RedScarf: a User-friendly Multi-platform Network-on-Chip Simulator

E. A. da Silva, D. Menegasso, S. Vargas Jr. and C. A. Zeferino
Laboratory of Embedded and Distributed Systems
University of Vale do Itajaí
Itajaí, SC, Brazil 88302-202
eas@ univali.br, {daniel.menegasso, sergiovargas}@edu.univali.br, zeferino@univali.br

Abstract— **The design space exploration of Networks-on-Chip (NoC) requires tools to evaluate the network performance, tune its parameters and achieve the requirements of the target application. In this context, this paper presents RedScarf, a simulation environment with graphical user interface and a set of tools that automate the design space exploration of NoCs. It presents many resources that make RedScarf a powerful tool for Research and Education on NoCs. RedScarf is user friendly, platform-independent and offers a diversity of topologies and communication mechanisms, thus covering the main on-chip interconnect architectures described in the literature. Experimental results demonstrate how the tool is useful for performance analysis of on-chip interconnect architectures.**

Keywords—Multi- and Many-core Systems; Network-on-Chip; Performance Evaluation; Simulation.

I. Introduction

As the industry advances towards multi-core and many-core systems (or SoCs – Systems-on-Chip), Networks-on-Chip (NoCs) emerged as the best alternative to the traditional in chip interconnects: point-to-point channels and shared buses [1]. The NoC approach has advantages over both technologies, because it offers, simultaneously, reusability, performance scalability and communication parallelism [2].

However, the NoC design space is large, and meeting the performance and costs requirements needs considering several architectural aspects [3]. Different attributes, as the network topology, buffer sizes, the number of virtual channels, arbitration and flow control schemes, must be considered. The main goal of a designer is to find the best network configuration to achieve the requirements of the target application.

Designing NoCs requires specialized tools to evaluate the network performance, as well as to trade-off alternative implementations and tune parameters [4]. They facilitate the design space exploration by reducing the time to test different architectural alternatives, as shown in [5].

Since Networks-on-Chip became a hot topic, several researchers have proposed different tools to aid in the design space exploration. However, most of these tools have limitations that are addressed in this paper. Few of them are platform-independent software and offer a graphical user interface (GUI) to facilitate its use. Most of them are limited to few topologies and alternatives for the communication mechanisms of a NoC (*i.e.* flow control, routing, arbitration, switching and buffering). In addition, few tools offer a detailed documentation and means for collaborative work.

Addressing these issues, we developed a NoC simulation environment composed of a user-friendly interface and a set of tools that automate the design space exploration. This environment is named *RedScarf* and offers several resources that facilitate the traffic modeling, the network configuration and simulation, and the performance analysis. Also, the environment offers detailed documentation, supplementary material for classes and labs, and makes use of a web tool for collaborative development. Such resources make *RedScarf* a powerful tool for Research and Education on NoC.

The remainder of this paper is organized as follows. Section II discusses related work. Section III describes the architecture and features of *RedScarf*. Following, Section IV presents experimental results that demonstrate its use for design space exploration. Section V presents the final remarks.

II. Related Work

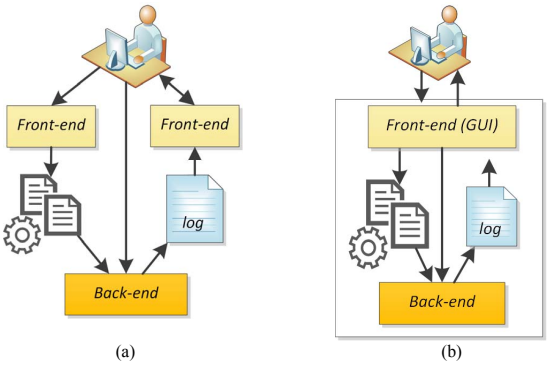
This section presents similar tools in order to identify the basic features of NoC simulators. Only public academic tools that are currently available for download were considered, including Nirgam [6], NoC Simulator [7], OCCN [8], BookSim [9], Noxim [10], HeMPS [11] and gem5 (with GARNET models) [12]. Commercial products or academic tools that are no longer available were not evaluated.

A. Tool architecture

Generally, simulation environments can be structured in two layers: the user interface (*front-end*) and the simulator (*back-end*). The user can see these layers as separated applications or as a single application.

In the first approach (Fig. 1.a), the front-end is composed of a set of separate tools, which are visible to the user. Text editors are used to edit configuration files and visualize data log files generated during the simulation. Spreadsheets or graphic viewers are typically used to analyze the results and generate graphics. The simulator is usually invoked by command-line. Nirgam, NoC Simulator, OCCN, BookSim, Noxim, and gem5 are based on this approach.

2324-7894/17 $31.00 © 2017 IEEE
DOI 10.1109/SBESC.2017.16



(a) (b)

Fig. 1. Simulator architecture: (a) separated tools; (b) integrated tools.

In the second approach (Fig. 1.b), the user interacts only with the front-end, which is implemented as a GUI that facilitates the simulator usage. The user configures the parameters, invokes the simulator and visualizes the results by means of the graphic components of the interface. HeMPS is the only one of the analyzed tool that follows this approach.

B. Measurement methods and metrics

There are two methods to measure the network performance: *open-loop* and *closed-loop*. Open-loop measurement enables the traffic parameters to be controlled independently of the network itself; while in the closed-loop, measurement the network influences the traffic [13].

The first method applies synthetic workloads following spatial and temporal distributions used to evaluate the network performance on a specific traffic pattern. Nirgam, NoC Simulator, and BookSim are based on this method.

The second method is useful for measuring overall system performance. It is used in tools that describe and simulate a complete SoC, with a NoC interconnecting processors, memories and other components, running some application. For this reason, they are also called full-system simulators. This method is used in OCCN, gem5, and HeMPS.

The main metrics used to characterize the performance of a NoC are throughput and latency [14]. Throughput is the maximum amount of data delivered by the network per time unit. It is also called the accepted traffic. Latency is the time required for a packet to traverse the network from source to destination [13]. Full-system simulators, like HeMPS, also use the execution time as a performance metric.

C. Modeling language

The simulator is an executable file generated from a model that describes a system composed of a NoC and processing elements (*i.e.* instrumentation terminal modules). Most of the NoC simulators presented in the literature use plain C++ as entry method to model the network and system components. In order to capture hardware specificities, several of these simulators use SystemC, a standard ANSI C++ library for hardware and software design. Other works use a hybrid approach and mix SystemC and VHDL.

NoC Simulator, BookSim, and gem5 use C++. Nirgam, OCCN, and Noxim are SystemC-based. HeMPS applies a mixed approach, using SystemC to describe the system components and VHDL to model the NoC.

D. Network parameters

Usually, NoC simulators model a generic and parameterizable NoC architecture. One can select a topology, customize the channel width, buffers depth and the number of virtual channels. Also, it is possible to select alternatives for the communication mechanisms, including the routing algorithm, arbitration policy, flow control and switching techniques.

E. System platform

Most of the NoC simulators are implemented in C++ (or SystemC) and are command-line-oriented. These features enable the execution on different platforms. On the other hand, GUI-oriented tools usually require that the graphical framework is compiled for different platforms (*e.g.* Linux, Windows and Mac OS). Most of the tools analyzed in this work were tested only on Unix-like platforms, mainly Linux.

F. Discussion

Table I summarize the features of the aforementioned NoC simulators. Only one of them has integrated architecture with a GUI to facilitate its use. However, this tool is a closed-loop simulator and focuses on the system performance evaluation. None of these tools were tested in the main three operating system platforms: Windows, Linux, and Mac OS. In this context, this paper presents *RedScarf*, an integrated, GUI-oriented, open-loop, SystemC-based and multi-platform NoC simulator that covers a broad set of architectural attributes for design space exploration of on-chip interconnects.

TABLE I. NOC SIMULATORS CHARACTERIZATION

Tool	*Architecture*	*Measurement metric*	*Modeling language*	*Architectural attributes coverage*						*Tested platforms*	*Versioning*	*Ref.*
				T	*R*	*SW*	*FC*	*AR*	*M*			
NoC Simulator	Separated tools	Open-loop	C++	1	1	1	1	1	F	STL-compliant	*n.a.*	[7]
BookSim	Separated tools	Open-loop	C++	9	25	1	1	A	F	Unix, Windows (Cygwin)	Git	[9]
Nirgam	Separated tools	Open-loop	SystemC	2	3	1	1	1	F	SystemC-compliant	*n.a.*	[6]
Noxim	Separated tools	Open-loop	SystemC	1	8	1	1	1	D	SystemC-compliant	Git	[10]
OCCN	Separated tools	Closed-loop	SystemC	2	2	1	1	1	D	Unix and Sun/Solaris	*n.a.*	[8]
gem5	Separated tools	Closed-loop	C++	3	5	1	1	1	F	Linux and OS X	Git	[12]
HeMPS	Integrated tools	Closed-loop	SystemC/VHDL	1	1	1	1	1	D	Linux	Git	[11]
RedScarf	Integrated tools	Open-loop	SystemC	7	11	2	2	4	F	Windows, Linux, and OS X	Git	This

OBS: T: topologies, R: routing algorithms, SW: switching techniques, FC: flow controls, AR: arbiter policies, M: memory; F: fully parameterizable (depth and virtual channels); A: use allocators; D: parameterizable in depth

III. REDSCARF

RedScarf is a simulation environment with emphasis on the performance evaluation of NoCs. It offers a graphical interface that facilitates its usage and enables its execution in the main desktop operating system platforms.

A. Simulator architecture

RedScarf follows the integrated approach. The GUI was implemented in Qt and the simulator in SystemC. Qt is a cross-platform application framework for software development. Fig. 2 presents the software architecture of *RedScarf*.

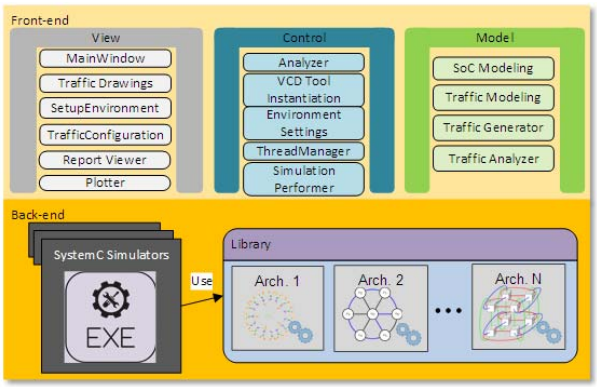


Fig. 2. Software architecture

The front-end is structured according to the Model-View-Controller (MVC) software architecture design pattern. The Model layer includes a set of tools to model the System and Traffic. It also comprises a tool to analyze the traffic reports resulted from the simulation. The View layer encompasses the windows that compose the interface and a set of tools to visualize the results. Finally, the Control layer interconnects the other layers and manages the environment settings and the simulations. It also invokes the VCD-based waveform viewer (*e.g.* GTKWave, Scansion), which is the only third-party tool.

The back-end is composed of a library of simulation components that model several NoC architectures. The library follows the plug-in-based approach. The inclusion of new components does not require the simulator be recompiled. Simulators are executed in parallel to speed up the simulations. The number of parallel executions is defined according to the number of architectures configured to be evaluated and the operation frequencies to experiment, and the number of cores on the host machine.

B. Measurement and metrics

RedScarf uses the open-loop measurement method. The simulator is based on a system model composed of a NoC and terminal instrumentation modules, as depicted in Fig. 3.

The terminal instrumentation module is composed of a set of modules, each one with a traffic generator (TG) and a traffic monitor (TM). The traffic generator creates packets and injects them into the NoC according to the traffic model defined by the user. The traffic monitor collects information from the incoming packets that are used for performance evaluation. The system model also includes a centralized module, named *StopSim*, which counts the packets injected and received by each terminal instrumentation module. This information allows identifying when the simulation stop condition is reached.

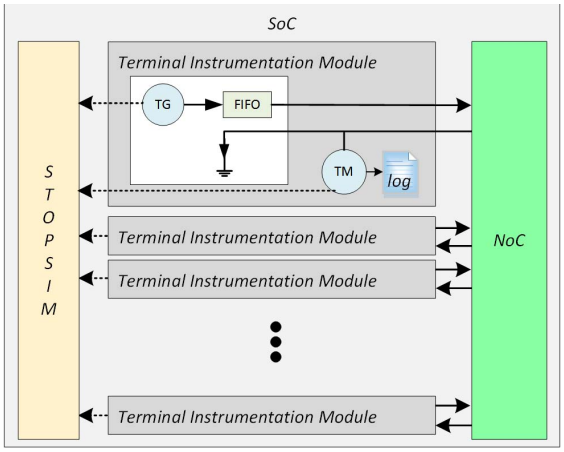


Fig. 3. Simulator architecture

A packet is composed of a header, an unlimited-length payload, and a trailer, which can be the last payload flit. The header contains the source and destination address, which are used by the network to forward a packet to its destination. It also includes the source address and tags that identify the communication flow and the traffic class to which the packet belongs.

When a TG creates a packet, it saves in a structure a set of information for performance evaluation, including the required bandwidth (in Mbps), the communication deadline (in ns) and a 64-bit timestamp. This timestamp identifies the clock cycle at which the packet was created, according to the injection rate defined in traffic model. With this solution, it is not necessary to use an unbounded FIFO to decouple the traffic source from the NoC. Therefore, the shorter packet is composed of a header and a trailer flit.

The TMs at each terminal instrumentation module collect the following information of each incoming packet: (*i*) source address; (*ii*) flow identifier; (*iii*) class identifier; (*iv*) communication deadline; (*v*) cycle at which the packet was created; (*vi*) cycles at which the header and the trailer were received; (*vii*) payload length; and (*viii*) required bandwidth. All these data are saved in a log file that is processed by a traffic analysis tool to calculate the following performance metrics: offered traffic, accepted traffic, latency and deadline miss ratio.

C. Modelling languages

The system is implemented in SystemC. The network models are cycle-accurate and described at RTL (Register-transfer Level) by using a synthesizable subset of SystemC [15]. Each component is a one-to-one implementation of its VHDL counterpart. The terminal instrumentation and StopSim modules use higher level structures of C++.

D. Network parameters

The simulator of *RedScarf* is based on an extensible and parameterizable library of components that models a set of interconnect architectures. Since *RedScarf* is intended to be used in Research and Education, it comes with a set basic reference NoCs with the following ready-to-use alternatives:

- *Topologies*: shared bus, crossbar, ring, chordal ring, 2-D torus, and 2-D and 3-D meshes;
- *Flow control*: credit-based and handshake;
- *Switching*: wormhole and circuit switching;
- *Routing*: according to the topology. Obs.: For 2-D mesh, there are these alternatives: deterministic (XY) and partial adaptive algorithm based on the Turn model;
- *Buffering*: input and output FIFO buffers;
- *Arbitration*: static, random, rotate and Round-Robin; and
- *Virtual channel*: none (*i.e.* a single channel), two, four, eight, sixteen or thirty-two channels.

Researchers can use this library as a reference to implement and evaluate new techniques they are proposing. In Education, students can use this subset as introductory experiments about NoC performance evaluation, and then extend them to explore other alternatives in the NoC design space.

E. The graphical user interface

RedScarf interface is structured in three tabs that correspond to its toolchain: (*i*) *System Configuration*; (*ii*) *System Simulation*; and (*iii*) *Performance Analysis*.

In the *System Configuration* tab (Fig. 4), the user can define the system size (number of tiles) by topology classification (Non-Orthogonal, 2D-Orthogonal, 3D-Orthogonal), the channel width (in bits), and the traffic pattern of the communication flows to be generated by each terminal instrumentation module. The user can configure an unlimited set of traffic patterns by clicking on the corresponding *Add Traffic* button in the *Traffic Configuration* dialog accessible from the main window. The *Add Traffic* button open the *Traffic Pattern* modal dialog with a form by which the user enters the traffic parameters (Fig. 5).

RedScarf uses the traffic generation method presented in [16]. In this method, a traffic pattern is characterized by a spatial distribution, its traffic class, the type of injection, and the corresponding parameters. For instance, according to the traffic pattern depicted in Fig. 5, the terminal instrumentation module from the "Multiple" sources will inject one 96-bit packet per batch to each one of all the other nodes in the system at a constant bitrate of 320 Mbps. This configuration will be replicated to all the selected nodes in the *Traffic Setup*. This facilitates the traffic modeling applying the same traffic pattern to several nodes. A batch is restarted each time it sends all the packets defined in the traffic model.

After defining the traffic model, the user can configure and start the experiments in the *System Simulation* tab (Fig. 6). This

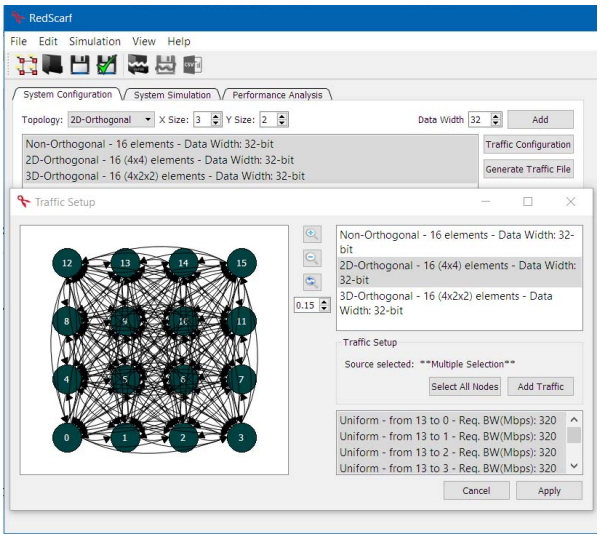


Fig. 4. *System Configuration* tab (running on Windows OS)

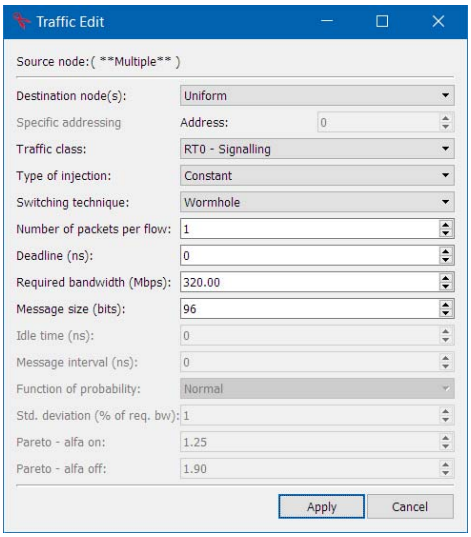


Fig. 5. *Traffic Pattern* dialog window (running on Windows OS)

tab has a panel to configure up to five experiments in which the user can select different parameters to define one or more NoC configurations for evaluation and comparison. In the bottom of this tab, the user can select a range of operating frequencies to vary the relationship between the traffic patterns requirements and the network capacity. In other words, in *RedScarf,* we consider that the communication requirements of the target application do not vary (*e.g.* always 320 Mbps). What we vary is the network capacity (*e.g.* 3200 Mbps at 100MHz or 320 Mbps at 10 MHz, for a 32-bit data channel).

The example in Fig. 6 has five NoC configurations with clock frequency varying from 100 to 10 MHz (in steps of -10 MHz). Considering that the NoC channels are 32-bit wide, the bandwidth of each channel varies from 3.2 Gbps to 320 Mbps (*i.e.* 3200 Mbps, 2880 Mbps,…, 320 Mbps). If all the nodes inject packets at 320 Mbps, the normalized injection rate varies from 0.1 (for 100 MHz) to 1.0 (for 10 MHz).

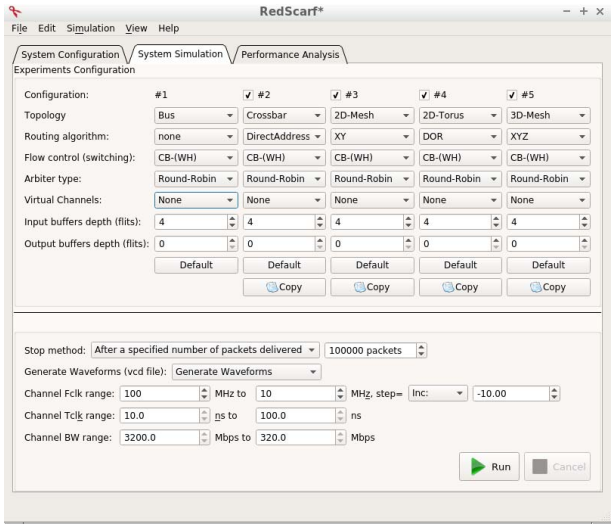


Fig. 6. *System Simulation* tab (running on Linux OS)

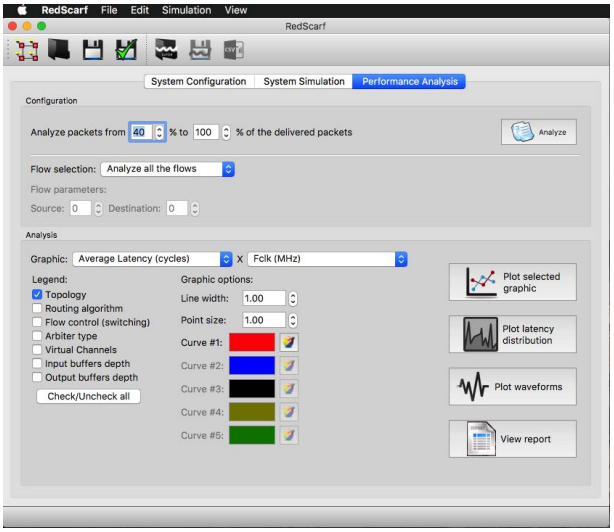


Fig. 7. *Performance Analysis* tab (running on Mac OS X)

After configuring the experiments, the user must click on the *Run* button to start the simulations. *RedScarf* generates the configuration files for each experiment configuration and runs the simulators for each clock frequency. In the example of Fig. 6, *RedScarf* generates five different config files and runs 10 simulations for each configured simulator, one for each clock frequency. Then, 50 experiments are executed.

Three criteria are available to stop a simulation: (*i*) when all the packets are delivered; (*ii*) when a specified simulated time (in cycles or ns) is reached; (*iii*) when a specified number of packets is delivered. In the first criterion, each node injects packets until the number of packets defined in the traffic model is reached (the batches are not restarted). This implies that the terminal instrumentation modules stop the packet injection before the end of the simulation. In the second and third, the nodes inject packets continuously, independently of the predefined number of packets to be injected, and the packet generation does not stop until the *StopSim* module interrupts the simulation.

When a simulation is concluded, the log files generated by the terminal instrumentation modules are read by a traffic analysis tool that calculates the performance metrics. The user can access these metrics using the *Performance Analysis* tab (Fig. 7). Results can be viewed in different ways: (*i*) tables; (*ii*) charts; and (*iii*) waveforms. The user can select a range of packets to be analyzed to consider the warm-up and drain phases.

F. *System platforms and resources*

RedScarf can be executed in the three main desktop platforms, what was illustrated in the previous figures: Windows (Fig. 4), Linux (Fig. 6) and Mac OS X (Fig. 7).

It has a complete manual to facilitate its usage by new users. The manual includes the system requirements, the description of each feature, and a tutorial to guide the user in its first experiments. It also offers supplementary material for usage in Education, including a presentation that summarizes the user manual, at labs for practical activities.

The development team of *RedScarf* uses Git for versioning and revision control, as well to share the source codes and documentation among its developers.

G. *Additional features*

RedScarf has several additional features that facilitate its use in Research and Education, including:

1. *Multithreading:* depending on the numbers of cores available in the host computer, the time necessary to run all the simulations can be reduced by running more than one simulation at the same time. The maximum number of cores to be used can be defined by the user in the *RedScarf* settings;
2. *Internationalized interface:* the GUI of *RedScarf* is currently available in English and Portuguese. However, it was designed and implemented to be easily migrated to different languages without the need of recompiling the software;
3. *Results exporting*: result reports can be exported in CSV (Comma Separated Value) for analysis in other tools, like spreadsheets and graphics utilities;
4. *Setup saving and loading:* the system configuration and the traffic model can be saved in an XML (eXtensible Markup Language) file for later reuse; and
5. *Experiments archiving and restoring:* all the log files of an experiment can be archived into a single file and later restored for additional analysis in *RedScarf*.

IV. EXPERIMENTS

This section presents results of experiments that demonstrate the use of *RedScarf*. We try to establish a configuration similar to the ones used in some similar works. A comparison with results of similar tools was out of the scope of this work, but it will be performed in a future study.

A. Configuration

We first defined three system configurations, one for each classification (2-D, 3-D and non-orthogonal) (*i*) a 2-D 4x4 mesh topology with 32-bit data channel as base architecture (as the one shown in Fig. 4); (*ii*) a 16-node Non-Orthogonal configuration; and (*iii*) a 3D 4x2x2 mesh. The traffic model was based on the configuration presented in Fig. 5. Each one of the 16 terminal instrumentation modules was configured to send 96-bit packets to all the other nodes in the system (*i.e.* a Uniform distribution) at a constant bitrate of 320 Mbps.

Second, we defined five experiments for comparison like the ones presented in Fig. 6. We only varied the topology and its respective routing algorithm and fixed the other parameters as follows:

- *Routing algorithm*: Minimal path;
- *Flow control* (*Switching*): credit-based (wormhole);
- *Arbiter type*: Round-Robin;
- *Virtual channels*: none;
- *Input buffers (flits)*: 4;
- *Output buffers (flits)*: 0 (*i.e.* none).

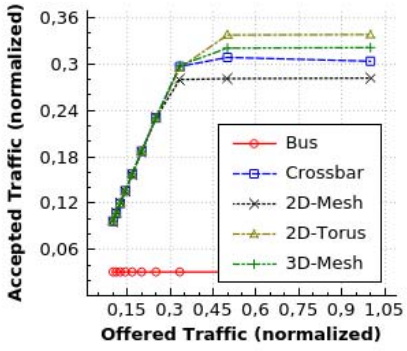
The clock operating frequency was varied from 100 to 10 MHz (in steps of -10 MHz). The simulations were configured to stop after the delivery of 100,000 packets.

B. Results

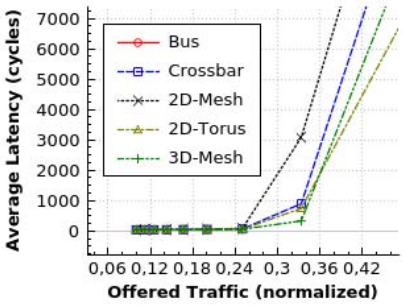
The experiments were carried out in a 4-core (Intel® Core™ i5-3210M CPU – 2.5 GHz) desktop. A total of 50 simulations were run (10 per interconnect architecture configuration) in about 30.3 minutes.

Fig. 8 shows as the accepted traffic and the average latency vary when offered traffic is increased (these charts were automatically plotted by *RedScarf*). Curiously, the crossbar (considered to be the best alternative in performance) does not present the best results in this scenario because it accepts less traffic than the 2-D torus and the 3-D mesh. As expected, the bus presented the worst performance, accepting much less traffic (0.0312 flits/cycle/node) than the other networks. As Fig. 8(a) depicts, the best topology was the 2-D torus, which accepted 0.3380 flits/cycle/node. In Fig. 8(b), one can see the behavior of the average latency when the networks reach their saturation point. At this point, none additional traffic is accepted by the network. As the bus saturates for all the injection rates used in this experiment, its latency is too high to be shown in Fig. 8(b).

The charts in Fig. 8 represent the traditional analysis used for performance evaluation of NoCs. In *RedScarf,* results can be also plotted in alternative ways that expand the analysis. For instance, the chart of Fig. 9 allows identifying the operating frequencies at which the different configurations are able to accept the offered traffic (in Mbps/node). This is useful to identify the minimal operating frequency to meet the communication requirements of a given application. For instance, most of the networks saturate when working at less than 40 MHz (the bus saturates at higher frequencies).



(a)



(b)

Fig. 8. Performance metrics: (a) accepted traffic x offered traffic; (b) average latency x offered traffic (bus omitted)

Another resource available for traffic analysis in *RedScarf* is a histogram that plots the latency fluctuation (*jitter*). Fig. 10 shows the effect of the network saturation to the latency. In the crossbar, there is a high concentration of packets delivered with low latencies, but the dispersion is high when the network is operating at 40 MHz. In the other architectures, the dispersion is smaller.

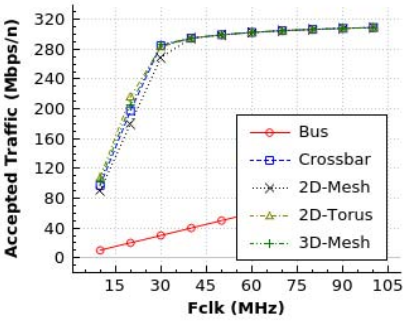


Fig. 9. Performance metrics: accepted traffic x operating frequency

RedScarf also allows visualizing the results in a table, which facilitates to identify the performance metrics. For instance, Fig. 11 presents the tables for the 2-D mesh and 2-D torus. One can see that the accepted traffic almost equals the offered traffic when the networks are not saturated. Also, at 40 MHz, the average latency equals 57.9 cycles in the 2-D torus, while the latency of the 2-D mesh is greater than 83 cycles, with a much higher standard deviation (the jitter).

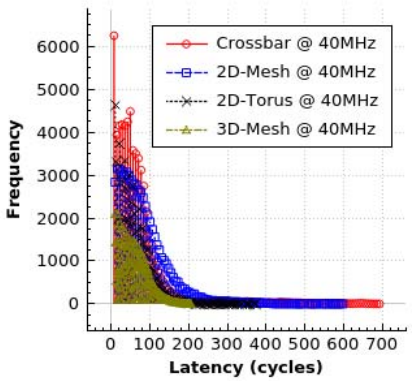
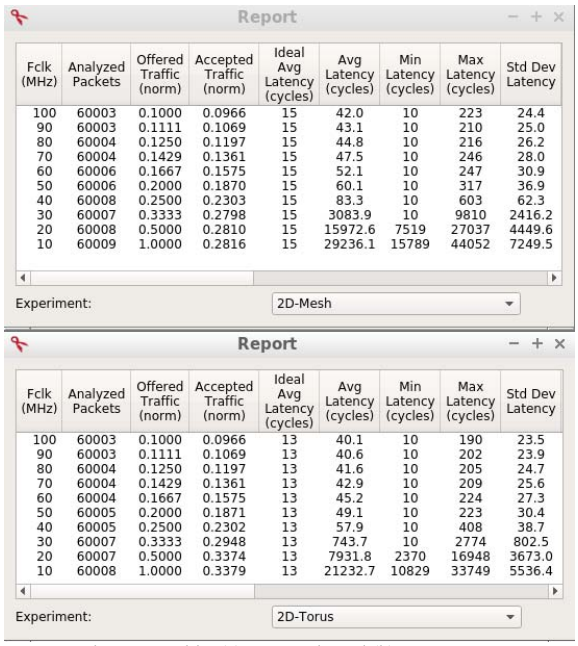


Fig. 10. Latency histogram

The user can also select a range of packets to be analyzed in order to consider the warm-up and/or drain phases. This selection disregards packets that usually suffer a smaller concurrency and present a low latency, what affects the minimal latency metric. For this experiment, the first 40% of delivered packets were discarded (Fig. 7) to avoid the systematic bias of warm-up period. The last packets delivered were not discarded because in the stop method used, the packet generation does not stop, what already avoid the bias in the drain phase (there is not a "drain" phase).



Report

Fclk (MHz)	Analyzed Packets	Offered Traffic (norm)	Accepted Traffic (norm)	Ideal Avg Latency (cycles)	Avg Latency (cycles)	Min Latency (cycles)	Max Latency (cycles)	Std Dev Latency
100	60003	0.1000	0.0966	15	42.0	10	223	24.4
90	60003	0.1111	0.1069	15	43.1	10	210	25.0
80	60004	0.1250	0.1197	15	44.8	10	216	26.2
70	60004	0.1429	0.1361	15	47.5	10	246	28.0
60	60006	0.1667	0.1575	15	52.1	10	247	30.9
50	60006	0.2000	0.1870	15	60.1	10	317	36.9
40	60008	0.2500	0.2303	15	83.3	10	603	62.3
30	60007	0.3333	0.2798	15	3083.9	10	9810	2416.2
20	60008	0.5000	0.2810	15	15972.6	7519	27037	4449.6
10	60009	1.0000	0.2816	15	29236.1	15789	44052	7249.5

Experiment: 2D-Mesh

Report

Fclk (MHz)	Analyzed Packets	Offered Traffic (norm)	Accepted Traffic (norm)	Ideal Avg Latency (cycles)	Avg Latency (cycles)	Min Latency (cycles)	Max Latency (cycles)	Std Dev Latency
100	60003	0.1000	0.0966	13	40.1	10	190	23.5
90	60003	0.1111	0.1069	13	40.6	10	202	23.9
80	60004	0.1250	0.1197	13	41.6	10	205	24.7
70	60004	0.1429	0.1361	13	42.9	10	209	25.6
60	60004	0.1667	0.1575	13	45.2	10	224	27.3
50	60005	0.2000	0.1871	13	49.1	10	223	30.4
40	60005	0.2500	0.2302	13	57.9	10	408	38.7
30	60007	0.3333	0.2948	13	743.7	10	2774	802.5
20	60007	0.5000	0.3374	13	7931.8	2370	16948	3673.0
10	60008	1.0000	0.3379	13	21232.7	10829	33749	5536.4

Experiment: 2D-Torus

Fig. 11. Results report table: (a) 2-D mesh; and (b) 2-D torus

C. *Additional features for perfomarnce analysis*

RedScarf has a useful feature for inspection and link-level performance analysis: the ability to generate VCD files for waveform visualization. By using a third-party viewer, one can observe the incoming and outgoing packets at the network terminals. For instance, Fig. 12 shows a zoom of a waveform with three packets being injected at the terminals of routers (0), (1) and (2). For instance, the packet injected by node (0) is composed of the following flits: [`0x100000032`, `0x000000000`, `0x000000001`, `0x200000021`], which are, respectively: the header, first payload flit, second payload flit, and the trailer. By analyzing a waveform like this one, one can obtain the latency of a single packet. He/she just needs to identify the cycle at which the packet trailer arrives at the destination terminal and subtract this value from the timestamp.

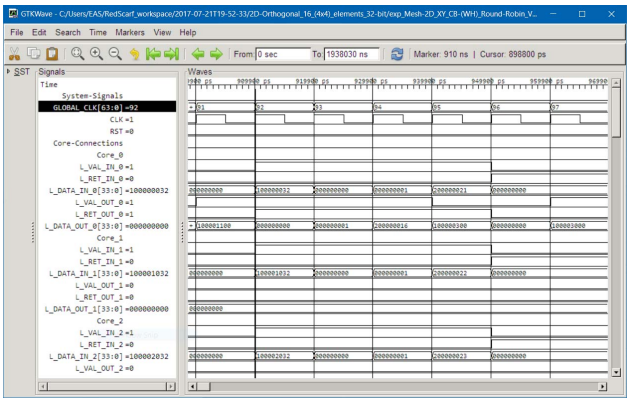


Fig. 12. Waveform view

RedScarf also verifies if a given NoC configuration meets the deadline requirements when it is specified in the traffic model. For each packet, it compares the required deadline with the packet latency and registers if the deadline met. It then offers metrics about the rate of deadlines that are met for each traffic class.

In addition, *RedScarf* allows the selection of a specific class or communication flow for analysis. For instance, one can analyze only one of the four classes of traffic (RT0, RT1, nRT0 and nRT1), or a specific communication flow identified by its source and destination addresses.

V. CONCLUSIONS

This paper presented *RedScarf*, a user-friendly multi-platform toolset for performance evaluation of NoCs. The design and implementation of *RedScarf* followed the best practices of software development. It is structured in *front-end* (the interface) and *back-end* (the simulator) layers. The *front-end* follows the MVC approach and contains 18.5Klines of source code (5.5K in the Model, 4.2K in the Control, 5.5K in the View, more 3.3K lines generated by the Qt framework). One important aspect is that it is totally based on free software and technologies that are well established in software development.

RedScarf has being used for Research and Education and is currently being evaluated for academic usage in other universities.

New tools and features are under development. The main goal is to integrate task mapping tools and benchmarks models. The user will be able to choose a different set of tasks or well-known benchmarks to evaluate its network model proposal front of the interconnect architectures available in the simulator. We also intend to add new traffic generation features, like the ability of to inject malicious traffic to attack the NoC security properties, as well as run traces of real

applications to evaluate real systems. Finally, we plan to integrate a multi-objective optimization tool to *RedScarf* for automatic exploration of the NoC design space.

All current and future features of *RedScarf* are intended to facilitate its usage and adoption by the community. We expect that it could be a reference tool for Research and Education in NoCs.

ACKNOWLEDGMENTS

The authors thank for the support of Capes – the Brazilian Federal Agency for Support and Evaluation of Graduate Education.

REFERENCES

[1] A. B. Kahng, B. Li, L.-S. Peh and K. Samadi, "ORION 2.0: A Power-Area Simulator for Interconnection Networks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, n. 1, 2012.

[2] A. Adriahantenaina, H. Charlery, A. Greiner, L. Mortiez and C. A. Zeferino, "SPIN: a scalable, packet switched on-chip micro-network," In *Design Automation and Test on Europe (DATE)*, 2003, p. 70-73.

[3] K. Macdonald, C. Nitta, M. Farrens and V. Akella, "PDG_GEN: A Methodology for Fast and Accurate Simulation of On-Chip Networks," *IEEE Transactions on Computers*, vol. 63, n. 3, 2014.

[4] L. Benini and G. De Micheli. *Networks on chips: technology and tools*. San Francisco: Morgan Kaufmann, 2006.

[5] H. Javaid, Y. Yachide, S. M. M. Shwe, H. Bokhari and S. Parameswaran, "FALCON: A Framework for HierarchicAL Computation of Metrics for CompONent-Based Parameterized SoCs," In *Design Automation Conference, 2014. DAC ACM/EDAC/IEEE,* p. 1-6, IEEE, 2014.

[6] *NIRGAM: A Simulator for NoC Interconnect Routing and Application Modeling*, 2007. Available in: http://nirgam.ecs.soton.ac.uk/.

[7] *NoC Simulator,* 2007. Available in: http://nocsim.blogspot.com.br.

[8] *OCCN: On-Chip Communication Architecture,* 2011. Available in: http://occn.sourceforge.net/.

[9] N. Jiang, J. Balfour, D. U. Becker, B. Towles, W. J. Dally, G. Michelogiannakis and J. Kim. "A detailed and flexible cycle-accurate Network-on-Chip simulator." In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS),* 2013. p. 86-96.

[10] V. Catania, A. Mineo, S. Monteleone, M. Palesi and D. Patti. "Noxim: An Open, Extensible and Cycle-accurate Network on Chip Simulator," In *IEEE 26th Int. Conf. Application-specific Systems, Architectures and Processors (ASAP)*, 2015, p. 162-163.

[11] E. A. Carara, R. P. de Oliveira, N. L. V. Calazans and F. G. Moraes. "HeMPS – A Framework for NoC-Based MPSoC Generation," In *IEEE Int. Symp. on Circuits and Systems (ISCAS)*, 2009. p. 1345-1348.

[12] N. L. Binkert, et al. "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, n. 2, 2011.

[13] W. J. Dally and B. P. Towles. *Principles and practices of interconnection networks*. Morgan Kaufmann, 2004.

[14] N. E. Jerger and L. S. Peh. *On-Chip Networks*. Madison: Morgan & Claypool, 2009.

[15] Synopsys. "Describing synthesizable RTL in SystemC™," Ver. 1.1, Jan. 2002.

[16] L. P. Tedesco et al. "Traffic Generation and performance evaluation for mesh-based NoCs," *Proc. of the 18th SBCCI*, 2005. p. 184-189.