Journal of Systems Architecture 99 (2019) 101633

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

RedScarf: an open-source multi-platform simulation environment for performance evaluation of Networks-on-Chip☆

Eduardo A. da Silva[a], Márcio E. Kreutz[b], Cesar A. Zeferino[a,*]

[a] University of Vale do Itajaí, Rua Uruguai, 458, Itajaí, SC, Brazil
[b] Federal University of Rio Grande do Norte, Campus Universitário Lagoa Nova, Natal, RN, Brazil

ARTICLE INFO

Keywords:
Multi- and many-core systems
Network-on-Chip
Performance evaluation
Simulation

ABSTRACT

The design space of Networks-on-Chip (NoCs) comprises a large number of architectural parameters. To comply with the performance requirements of target applications, NoC-based system designers need to employ tools to assess the impact of each parameter on the NoC and the performance of the application. In view of this, this paper presents a simulation environment named *RedScarf*, which was developed to facilitate the design space exploration of NoCs. *RedScarf* integrates a graphical user interface and a set of tools that automate the process of configuring and evaluating the network characteristics. By providing resources like multi-platform and multi-thread execution, among others, *RedScarf* is a powerful tool for Research and Education on NoCs. This work describes the *RedScarf* architecture and tools, and demonstrates by means of experiments how it can aid a designer in the task of assessing the performance of on-chip interconnect architectures.

1. Introduction

In the era of multicore systems, architectures present parallel behavior whereby many tasks can run simultaneously, leading to concurrency in communications. Since traditional on-chip interconnects (i.e., point-to-point channels and shared buses) [1] face limitations on either cost or performance/scalability, Networks-on-Chip (NoCs) emerged as the best alternative to cope with such constraints. The NoC approach has advantages over both technologies, offering simultaneously, reusability, performance scalability, and communication parallelism [2].

The design of NoC-based multiprocessing systems presents a large design space to be explored when optimized solutions are mandatory. In these designs, evaluations rely on analyzing several architectural aspects to meet performance and costs requirements [3]. Different attributes, such as network topology, the number of virtual channels, buffers size, and arbitration and flow control schemes must be considered. Therefore, designers are challenged to look for ways on finding network configurations that meet the requirements of target applications. This task can usually be accomplished using specialized tools to evaluate the network performance and find trade-off implementations with specific configurations [4]. These tools facilitate the design space exploration by reducing the time taken to test different architectural alternatives, as shown in [5].

Considering this scenario, several researchers have proposed different tools to aid in the design space exploration of NoC-based systems. However, most of these tools have limitations, which we address in this work. Few of them are platform-independent and offer a graphical user interface (GUI) to facilitate their use. Most of them are constrained to a few topologies and alternatives for communication mechanisms such as flow control, routing, arbitration, switching, and buffering. Also, few tools offer detailed documentation and are intended for collaborative work.

To address these issues, we have developed an open-source NoC simulation environment composed of a user-friendly graphical interface and a set of tools that automate the process of configuration and evaluation of several network characteristics. This environment, named *RedScarf* [6], offers several resources such as traffic modeling, network configuration, simulation, and assessment through performance analysis. The environment includes detailed documentation and supplementary material for classes and laboratories. It also uses a web tool for collaborative development. These resources make *RedScarf* a powerful tool for Research and Education on NoCs. Thus, the main contribution of this work is the development of a platform-independent simulation environment for performance evaluation of NoC architectures, which uses graphical resources and automation tools to facilitate the design space exploration. The simulation environment can also be used for scientific

☆ This work was supported by grants from Coordination for the Improvement of Higher Education Personnel (CAPES) – Finance Code 001 and National Council for Scientific and Technological Development (CNPq) – grants 315287/2018-7 and 436982/2018-8, Brazilian funding agencies for research and education.

* Corresponding author.
E-mail addresses: eas@univali.br (E.A. Silva), kreutz@dimap.ufrn.br (M.E. Kreutz), zeferino@univali.br (C.A. Zeferino).

https://doi.org/10.1016/j.sysarc.2019.101633
Received 22 February 2019; Received in revised form 19 August 2019; Accepted 21 August 2019
Available online 21 August 2019

and educational purposes, as it is GUI-oriented. The experimental results are presented using several types of graph, and different architectural configurations can be added through plug-ins. These features allow, for instance, students to argue of the meaning of a result when contrasted to architectural specifications. They also provide means that facilitate the work of a researcher when investigating the performance of new architectures.

This paper extends the work presented in [6], providing a more detailed description of the software architecture and discussing how a user can model and add new NoC models to *RedScarf*. It also presents a set of new experiments that better demonstrate the resources available in the simulation environment. The remainder of the paper is organized as follows. Section 2 discusses related work; and Section 3 describes the architecture and features of *RedScarf*. Following, Section 4 presents the experimental results that demonstrate the resources and their use for design space exploration. Finally, Section 5 concludes by presenting the final remarks.

2. Related work

Several tools for performance evaluation of NoCs have been presented in the literature. Some of them are designed for evaluation and comparison of a new architecture against a reference NoC, while others tools are intended to be generic and flexible to allow the modeling and evaluation of several architectures. However, not all of these tools are available to the public, their use being restricted to the researchers involved in their development.

As *RedScarf* was developed to be a public tool for the academic use, this section only examines other open-source tools that are currently available to download. After applying a systematic review protocol, we selected a set of tools that are similar to *RedScarf*. For our analysis, we have only considered the features included in the package available for installation. Any extra-features implemented by their authors or by third-parties, but not incorporated into the installation package, were not taken into account. It is also worth noting that commercial products or academic tools that are no longer available for public usage were not analyzed, as they do not comply with the goals of *RedScarf*.

Taking the above into consideration, we selected the following tools: Nirgam [7], NoC Simulator [8], OCCN [9], BookSim [10], Noxim [11], HeMPS [12], and gem5 (with GARNET models) [13]. For each tool, we examined its architecture, the measurement methods and metrics used for performance evaluation, the language used to model the network, the diversity of architectural parameters available for analysis, the operating system platforms supported by the tool, and the use of a versioning system for collaborative work.

2.1. Architecture

Simulation environments are generally structured in two layers: the user interface (front-end) and the simulator (back-end). These layers can be used as separated tools, or as a single application. In the separated tools approach, the front-end comprises a set of individual tools. As depicted in Fig. 1(a), each tool is visible to the user, who employs a text editor to describe the configuration files and analyze the logs that generated by the simulator. In order to create plots to exhibit the results, the user must apply third-party graphing utilities or spreadsheets. Furthermore, the simulator is usually invoked by command-line. Examples of simulators that follow this approach are Nirgam, NoC Simulator, OCCN, BookSim, Noxim, and gem5. In the second approach, the user interacts only with the front-end to configure the parameters, invoke the simulator, and view the results using a GUI, as illustrated in Fig. 1(b). HeMPS is the only one of the tools analyzed that follows this approach.

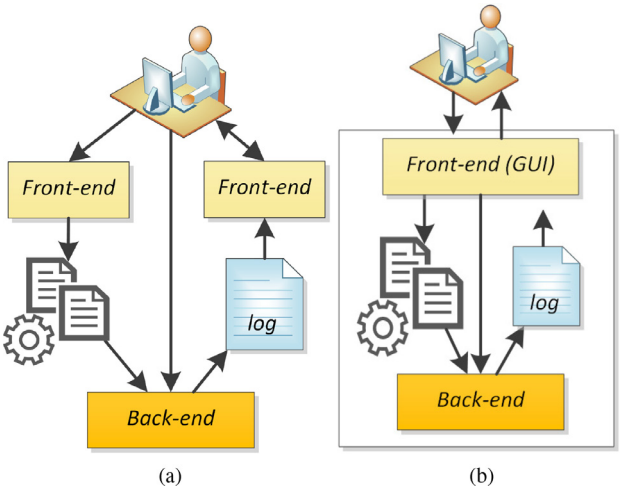

Fig. 1. Simulator architecture types: (a) separated tools; (b) integrated tools.

2.2. Measurement methods and metrics

Two methods can be employed to measure network performance: open- and closed-loop. The open-loop method controls the traffic parameters independently of the network itself; while in closed-loop measurement systems, the network influences the traffic [14]. Moreover, the open-loop method applies synthetic workloads following spatial and temporal distributions to evaluate the network performance on a specific traffic pattern. Nirgam, NoC Simulator, and BookSim are open-loop measurement tools. On the other hand, the closed-loop method is useful for measuring overall system performance. Tools that describe and simulate a complete SoC running some application – with an NoC interconnecting processors, memories, and other components – follow this approach. For this reason, they are also called full-system simulators. Examples include OCCN, gem5, and HeMPS.

The primary metrics used to characterize the performance of an NoC are throughput and latency [15]. Throughput is the maximum amount of data delivered by the network per time unit. It is also known as the accepted traffic. Latency is the time required for a packet to traverse the network from source to destination [14]. Full-system simulators, like HeMPS, also use the execution time as a performance metric.

2.3. Modeling language

The simulator is an executable file generated from a model that describes a system composed of an NoC and processing elements (i.e., instrumentation terminal modules). Most of the NoC simulators described in the literature use plain C++ as the entry method to model the network and system components. 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, adopting SystemC to describe the system components and VHDL to model the network.

2.4. Network parameters

Usually, the NoC simulators model a generic and parameterizable NoC architecture. The user can select a topology and 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, and flow control and switching techniques.

Table 1
NoC simulators characterization.

Tool	Architecturetype	Measurementmethod	Modeling language	Tested platforms	Versioning
NoC Simulator	Separated	Open-loop	C++	STL-compliant	*n.a.*
BookSim	Separated	Open-loop	C++	Unix andWindows (Cygwin)	Git
Nirgam	Separated	Open-loop	SystemC	SystemC-compliant	*n.a.*
Noxim	Separated	Open-loop	SystemC	SystemC-compliant	Git
OCCN	Separated	Closed-loop	SystemC	Unix andSun/Solaris	*n.a.*
gem5	Separated	Closed-loop	C++	Linux andOS X	Git
HeMPS	Integrated	Closed-loop	SystemC/VHDL	Linux	Git
RedScarf	Integrated	Open-loop	SystemC	Windows, Linux, and OS X	Git

Obs.: n.a. = information not available

Table 2
Architectural attribute coverage.

Tool	Topology	Routing	Switching	Flowcontrol	Arbitration	Buffering
NoC Simulator	1	1	1	1	1	F
BookSim	9	25	1	1	A	F
Nirgam	2	3	1	1	1	F
Noxim	1	8	1	1	1	D
OCCN	2	2	1	1	1	D
gem5	3	5	1	1	1	F
HeMPS	1	1	1	1	1	D
RedScarf	7	11	2	2	4	F

Obs.: A = use allocators; F = fully parameterizable (depth and number of virtual channels); D = partially parameterizable (only depth).

2.5. 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 the graphical framework to be compiled for different platforms (e.g., Linux, Windows, or Mac OS). Most of the tools analyzed were tested only on Unix-like platforms, mainly Linux.

2.6. Discussion

Tables 1 and 2 summarize the features of the aforementioned NoC simulators. Only HEMPS follows an integrated architecture and provides a GUI to facilitate its use. However, this tool is a closed-loop simulator and focuses on system performance evaluation. None of these tools were tested in all three of the current main operating system platforms mainly in use: Windows, Linux, and Mac OS. In view of this, this paper presents *RedScarf*, an integrated, GUI-oriented, open-loop (the focus is to evaluate the NoC, not the system), SystemC-based, and multi-platform NoC simulator that covers a broad set of architectural attributes to help the designer in the task of exploring the design space of on-chip interconnects. As we highlighted in the introduction, *RedScarf* excels for the fact that it is platform independent, and because it provides tools that facilitate the setup and execution of experiments and analysis of their results. The plug-in-based implementation also makes it easier to model new architectures. Moreover, as Table 2 shows, *RedScarf* provides a diversity of architectural attributes that are ready for use as a baseline reference when evaluating the new proposals. These features make *RedScarf* a useful tool for both Research and Education.

It is worth noting that *RedScarf* was originally designed as a tool for performance evaluation of Networks-on-Chip. Its current version does not include power models for energy consumption analysis, or technology information for silicon costs evaluation. Moreover, this version of *RedScarf* was developed targeting conventional wired NoCs. Emerging technologies like photonic [16–18] and wireless [19,20] are not currently covered.

3. RedScarf

3.1. RedScarf *architecture*

RedScarf follows an integrated approach. We have implemented the GUI 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*.

We structured the front-end according to the Model-View-Controller (MVC) software architecture design pattern. The Model layer includes a set of tools to model the system and the traffic to be applied. It also includes a tool to analyze the logs generated by the simulator. The View layer encompasses the windows that compose the interface and a set of tools to view the results. Finally, the Control layer interconnects the other layers and manages the environment settings and the simulations. It also invokes a VCD-based waveform viewer (e.g., GTKWave, Scansion), which is the only third-party tool in *RedScarf*.

The back-end includes a library of SystemC models that can be combined to form different architectural alternatives. The library follows the plug-in based approach, which prevents the need to recompile the simulator in order to add a new component to the library.

RedScarf supports the use of multithreading. Each thread runs a simulator to evaluate a system configuration working under a given traffic load. Thus, it is possible to speed up the overall simulation by running multiple simulators at the same time, over different cores on the host machine.

3.2. Measurement and metrics

RedScarf uses the open-loop measurement method. The simulator relies on a system model that includes an NoC and a terminal instrumentation module attached to each router in the NoC, as shown in Fig. 3.

The terminal instrumentation module comprises a traffic generator (TG) and a traffic monitor (TM). TG creates packets and injects them into the NoC according to the traffic model that the user has defined. TM collects information from the incoming packets for performance evaluation. The system model also includes a centralized module, named

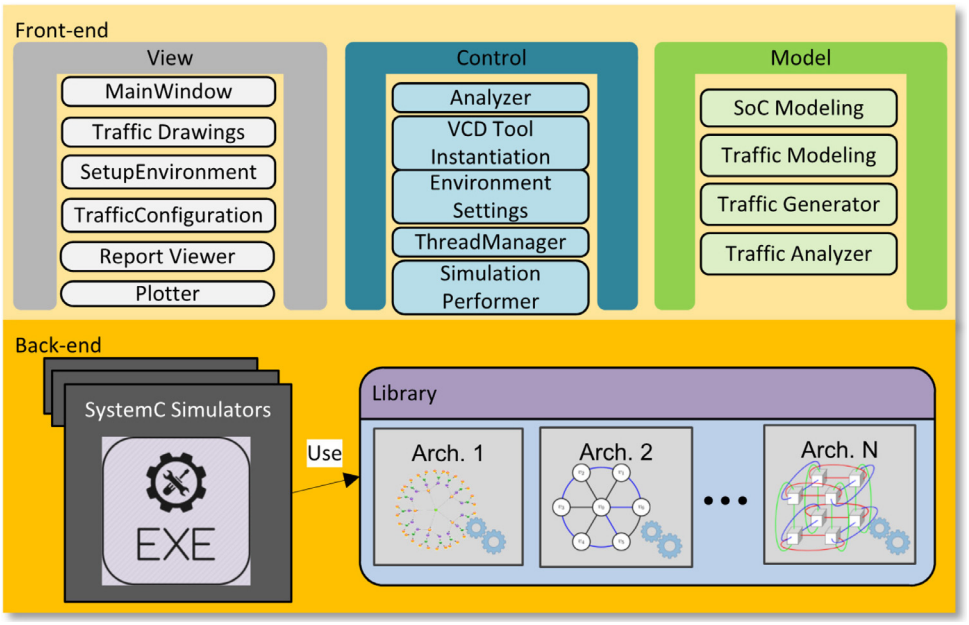

Fig. 2. Software architecture.

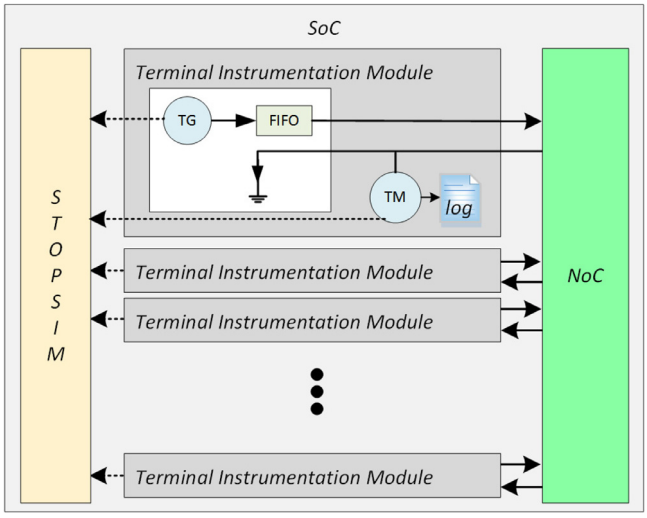

Fig. 3. Simulator architecture.

StopSim, which counts the packets injected and received by each terminal instrumentation module. This information enables the user to identify when the simulation has reached its stop condition.

A packet comprises a header, a payload of unlimited-length, and a trailer, which may be the last payload flit. The header contains the destination address that the network uses to forward the 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 stores information about it in an internal data structure. This information comprises the required bandwidth (in Mbps), the communication deadline (in nanoseconds), and a 64-bit timestamp that identifies when the TG created the packet. This approach avoids the use of an unbounded FIFO to decouple the traffic source from the NoC.

When a packet arrives at a terminal instrumentation, a TM collects the following information about the 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. This information is then stored in a log file to be processed by a traffic analysis tool that calculates the performance metrics (i.e., offered traffic, accepted traffic, latency, and deadline miss ratio).

3.3. Modeling languages

We implemented the simulator of *RedScarf* in SystemC and described the library of components using cycle-accurate RTL (Register-transfer Level) modeling. For this, we have applied a synthesizable subset of SystemC [21] and each component is a one-to-one implementation of its VHDL counterpart. Thus, we guarantee that the simulation model is as accurate as a synthesizable description. On the other hand, we have modeled the terminal instrumentation and StopSim modules using higher-level (non-synthesizable) structures of C++.

3.4. Router architecture

RedScarf uses a distributed and modular router architecture with a parameterizable number of input and output channels. The input channel comprises the circuitry that regulates the flow of incoming packets, stores their flits, runs the routing function, and requests an output channel to forward the flits. The output channel has an arbiter that schedules the requests received from the input channels and commands the internal switches to establish a path with the selected input channel. It is also responsible for regulating the flow of outgoing flits.

This modular architecture is extendable and allows different types of on-chip communication architecture to be implemented. Considering a system with N nodes, *RedScarf* emulates a shared bus by instantiating an $N \times 1$ router (with N input channels and a single output channel). A crossbar comprises a single $N \times N$ large router, while an NoC consists of a set of N $m \times m$ small routers, where m depends on the topology and router position in the NoC.

3.5. Adding an NoC to RedScarf

The process to model and add a new NoC to *RedScarf* comprises three main steps:

1. *Topology plug-in*: The developer needs to implement the topology plug-in of its new component and must describe: (*i*) *A topology class*, which comprises the NoC interface and the topology layout; and

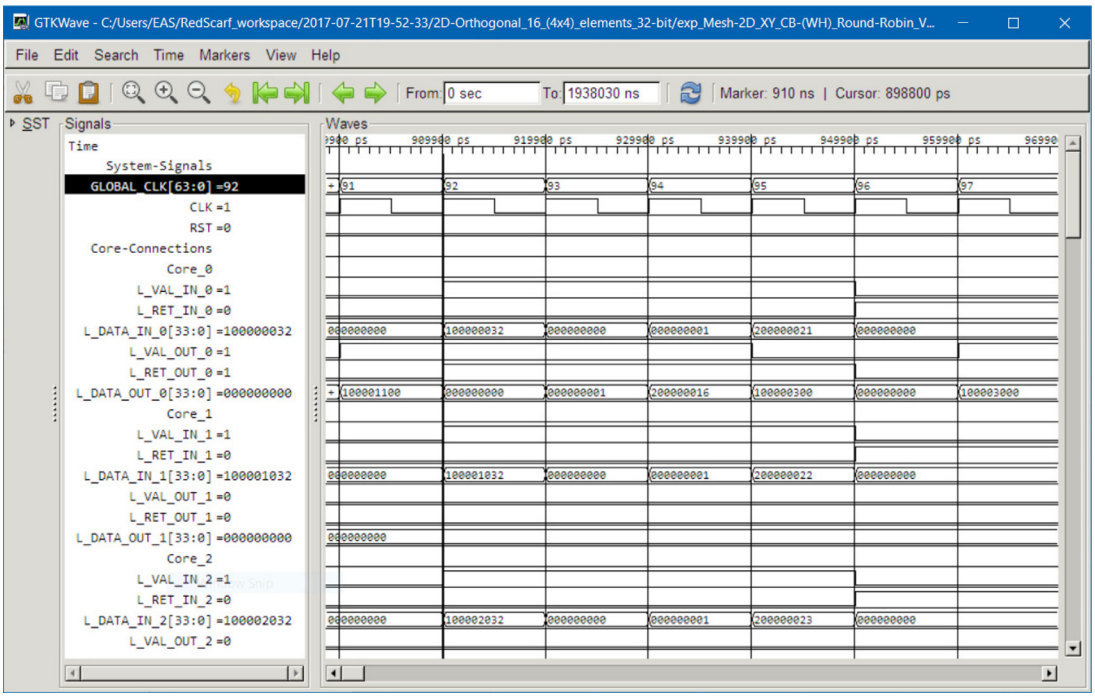

Fig. 4. Waveform view.

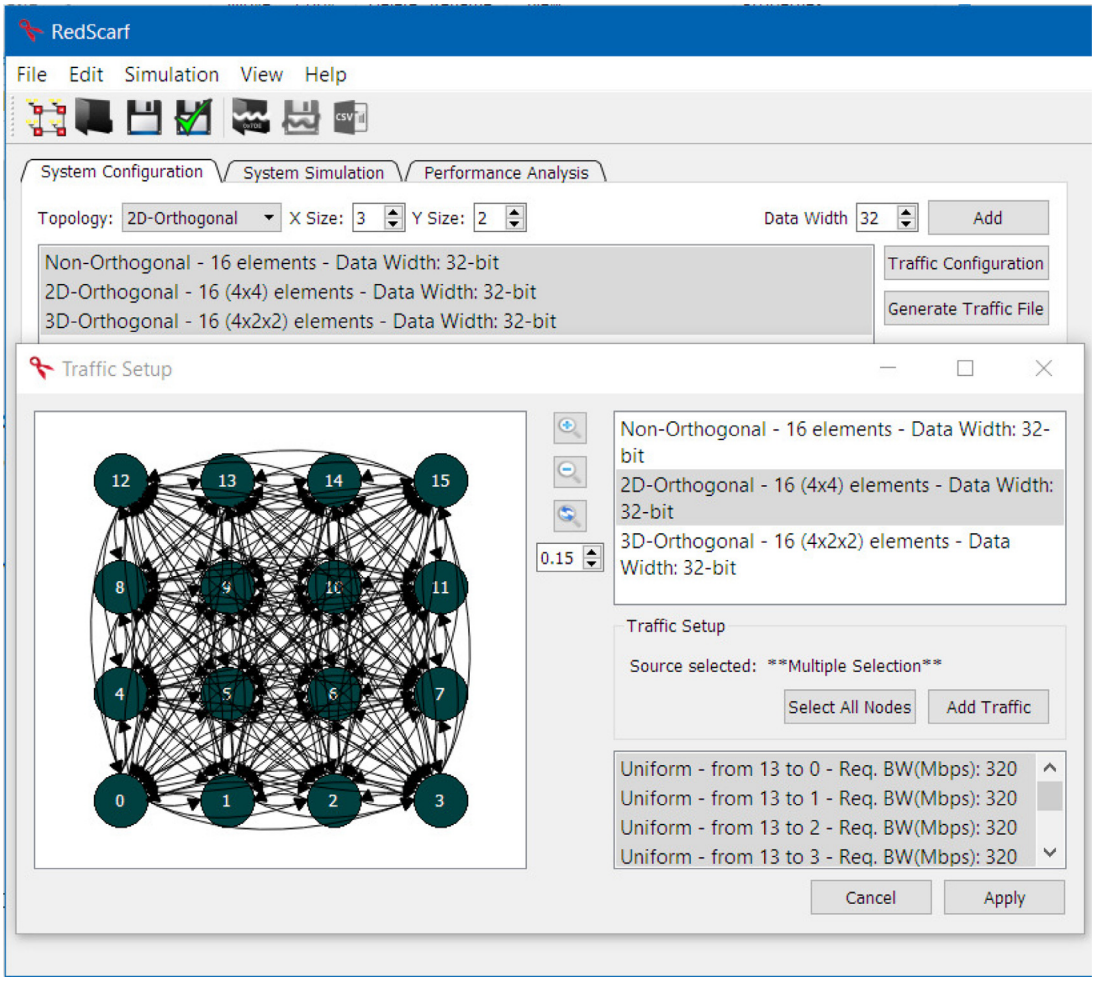

Fig. 5. System Configuration tab (running on Windows OS).

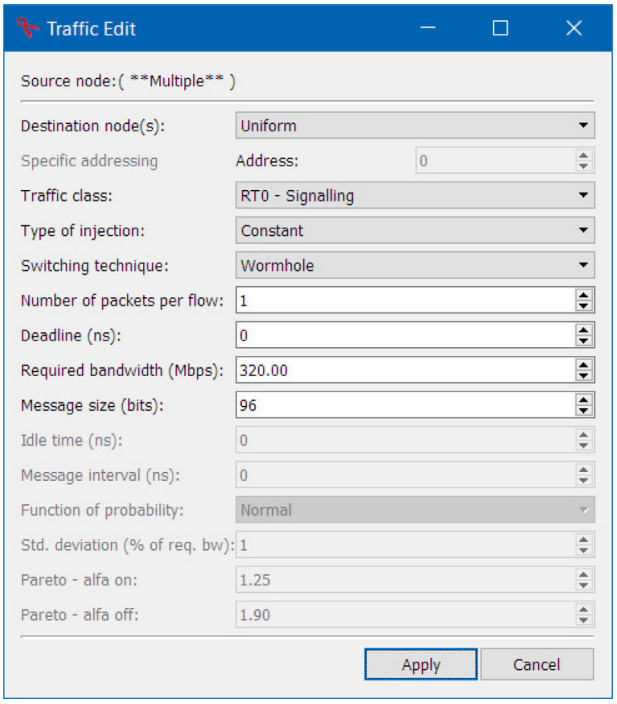

Fig. 6. Traffic Pattern dialog window (running on Windows OS).

(*ii*) *the factory methods*, which are the procedures to instantiate and deallocate the topology object.

2. *Routing algorithm plug-in*: The developer needs to implement the routing algorithm plug-in for its network. This step comprises the description of: (*i*) *A routing class*, which must be compliant with the topology modeled; and (*ii*) *the factory methods*, which implement the procedures to instantiate and deallocate the routing objects.
3. *Front-end updating*: the developer must change the system configuration file of *RedScarf* to include the new topology and the routing algorithm added to the plug-ins directory.

The developer then must compile the classes implemented in the first two steps and move the output files to the plug-ins directory. Following, the user then must restart *RedScarf* for the changes to take effect. It is suggested starting simple simple experiments to validate the implementations.

One resource that helps the developer in the verification of a new model is the use of some third-party waveform viewer to exhibit the VCD (Value Change Dump) file generated by *RedScarf*. With this resource, the developer can observe the incoming and outgoing packets at the network terminals. For instance, Fig. 4 shows a zoom of a waveform with three packets being injected at the terminals of routers 0, 1, and 2. The packet injected by node 0 comprises the following flits: [0x100000032, 0x000000000, 0x000000001, 0x200000021], which are, respectively: The header, the first payload flit, the second payload flit, and the trailer. By analyzing a waveform like this one, it is possible to verify whether the network has delivered each packet to its

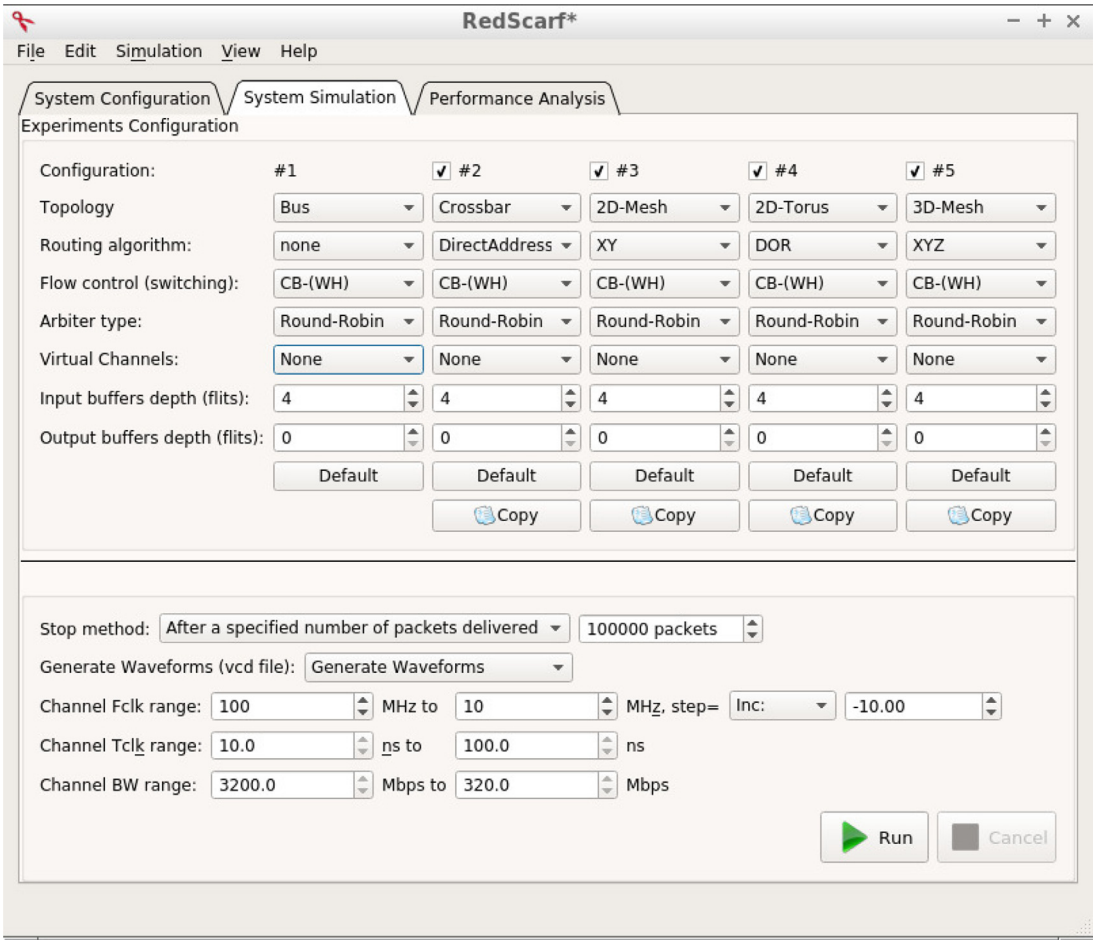

Fig. 7. System Simulation tab (running on Linux OS).

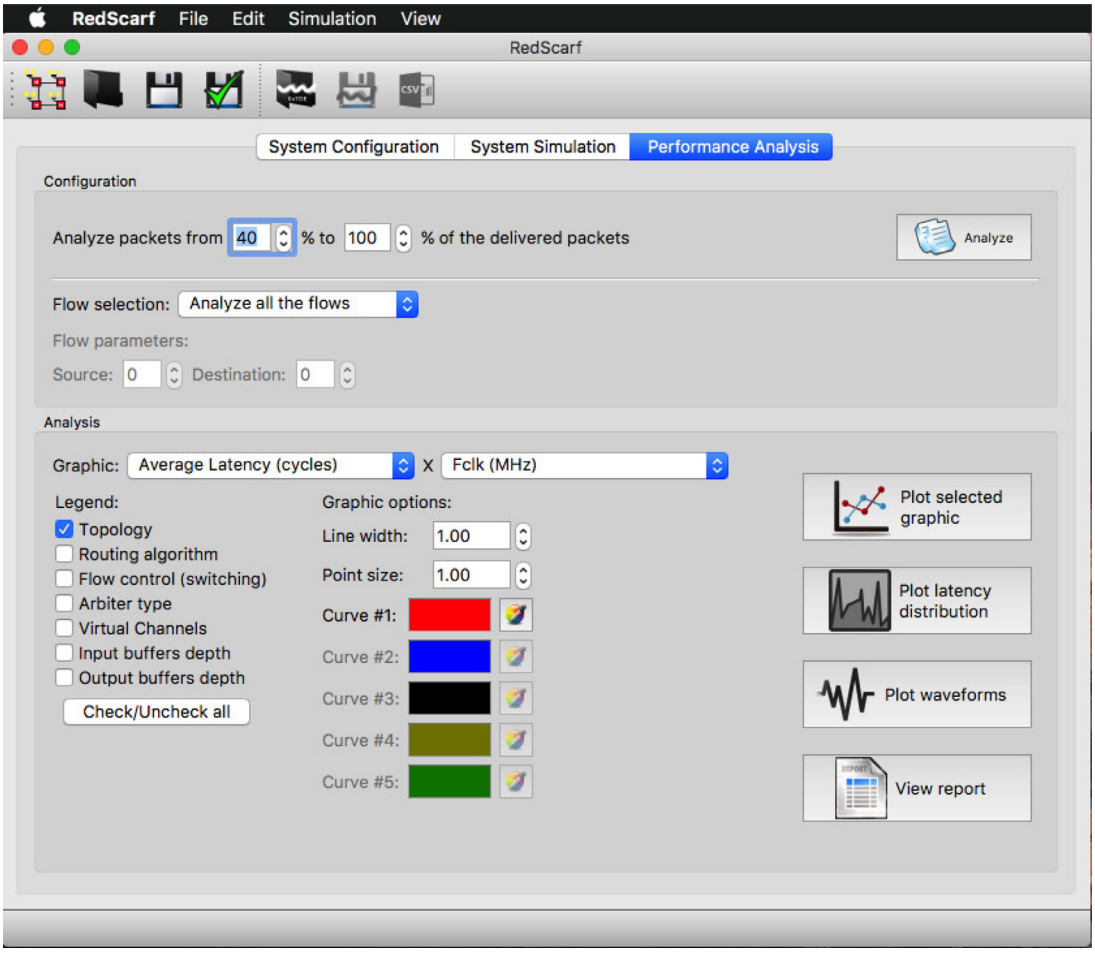

Fig. 8. Performance Analysis tab (running on Mac OS X).

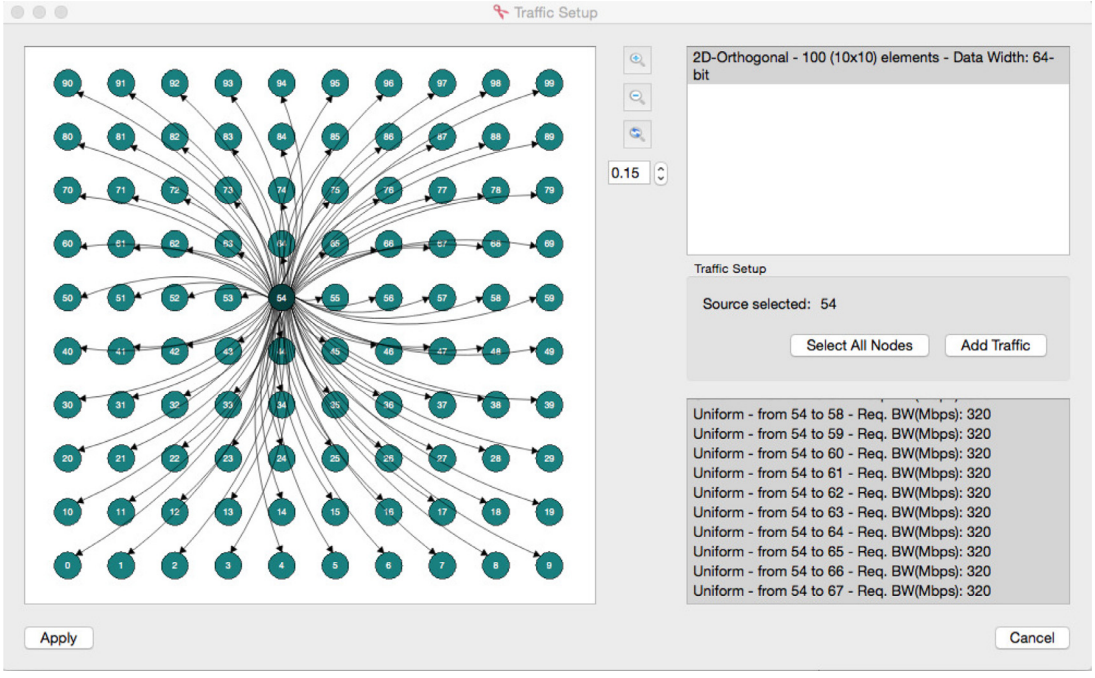

Fig. 9. Representation of the traffic model for node 54.

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
11	70048	0.4545	0.1405	35	33305.8	9274	70978	13131.1
13	70047	0.3734	0.1393	35	28749.3	5929	68848	13693.6
16	70049	0.3051	0.1394	35	23103.8	451	66030	14770.1
20	70048	0.2500	0.1357	35	17814.5	19	63254	15664.6
25	70045	0.2041	0.1276	35	15529.7	19	63990	15989.2
30	70045	0.1673	0.1153	35	13090.1	19	62914	15388.0
37	70046	0.1370	0.1029	35	11535.0	19	63388	14619.3
45	70047	0.1119	0.0925	36	8605.4	19	60253	12080.4
55	70049	0.0917	0.0811	36	5554.7	19	60604	9516.1
67	70046	0.0751	0.0687	36	3681.6	19	52375	6682.9
81	70042	0.0614	0.0565	36	2192.4	19	38367	3988.0
99	70014	0.0503	0.0451	36	1234.0	19	20566	1721.8
122	70011	0.0412	0.0369	36	973.8	19	15387	1206.7
148	70008	0.0337	0.0301	36	802.4	19	9469	826.1
181	70011	0.0276	0.0247	36	703.5	19	7735	678.3
221	70011	0.0226	0.0202	36	629.6	19	7735	560.5
270	70009	0.0185	0.0165	36	584.2	19	6210	487.0
330	70010	0.0151	0.0135	36	554.9	19	5073	444.4
403	70010	0.0124	0.0111	36	536.0	19	4960	418.5
493	70010	0.0101	0.0091	36	520.4	19	4714	384.1
602	70010	0.0083	0.0074	36	510.8	19	1680	351.9
735	70009	0.0068	0.0061	36	510.9	19	1489	351.9
898	70009	0.0056	0.0050	36	510.9	19	1489	351.9

Experiment: XY

Fig. 10. Simulation report window for XY routing.

destination node. It is also possible to measure the packet latency given by the number of cycles between the header injection and the trailer ejection (in this case we are not taking into account the interval between the packet creation and its injection into the NoC). This resource is also useful in Education to illustrate intra-chip communication at a low level.

3.6. NoC parameters

RedScarf is intended for use in Research and Education and comes with a set of basic reference NoCs with the following ready-to-use alternatives:

- *Topologies*: Shared Bus, Crossbar, Ring, Chordal Ring, 2D Torus, 2D Mesh, and 3D Mesh.
- *Flow control*: Credit-based and Handshake.
- *Switching*: Wormhole and Circuit switching.
- *Routing*: According to the topology.
- *Buffering*: Input and output FIFO buffers.
- *Arbitration*: Static, Random, Rotate, and Round-Robin.
- *Virtual channel*: None (i.e., a single channel), 2, 4, 8, 16, or 32 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 first experiments about NoC performance evaluation, and then extend them to explore other alternatives in the NoC design space.

3.7. The graphical user interface

As mentioned above, *RedScarf* is a user-friendly and multi-platform NoC simulator. It has a graphical user interface (GUI) that facilitates the use of its resources. Furthermore, it runs on the main desktop platforms: Windows, Linux, and Mac OS X (see Figs. 5–8). The interface of *RedScarf* is organized in three tabs that correspond to its toolchain: (*i*) System Configuration; (*ii*) System Simulation; and (*iii*) Performance Analysis.

In the System Configuration tab (Fig. 5), the user must add at least one class of topology, and define the system size and the channel width. Also, it is necessary to configure the traffic patterns that he/she wants to apply to the system. In other words, the user must define the communication flows that will be generate by the terminal instrumentation modules. The example illustrated in Fig. 5 presents a scenario in which the user added three classes of topology (Non-Orthogonal, 2D Orthogonal, and 3D Orthogonal) for a system with 16 nodes and 32-bit channels to compare different topologies under a Uniform traffic pattern.

The user can configure a non-limited set of traffic patterns by clicking on the corresponding `Add Traffic` button in the Traffic Setup dialog box. *RedScarf* uses the traffic generation method proposed in [22], which describes a traffic pattern by defining a spatial distribution, its traffic class, and the type of injection (constant or variable). The graphical resources available in *RedScarf* facilitates the description of a traffic model, mainly when it relies on a spatial distribution that is applied to all the nodes of the system. In this case, the user needs to click on the `Select All Nodes` button in the Traffic Setup dialog box before describing a new traffic pattern to apply it to all the nodes. The usefulness of this feature becomes more evident as the number of nodes increases because it allows describing the traffic model in few steps, independently of the system size.

Fig. 6 illustrates the configuration of a traffic pattern for multiple nodes, simultaneously. According to this pattern, each node will generate communication flows to all the other nodes in the system (i.e., Uniform distribution) and inject one 96-bit packet per batch at a 320 Mbps constant injection rate. The example demonstrates how simple and fast traffic modeling in *RedScarf* can be.

After defining the traffic model, the user can configure and start the experiments in the System Simulation tab (Fig. 7). This tab has a

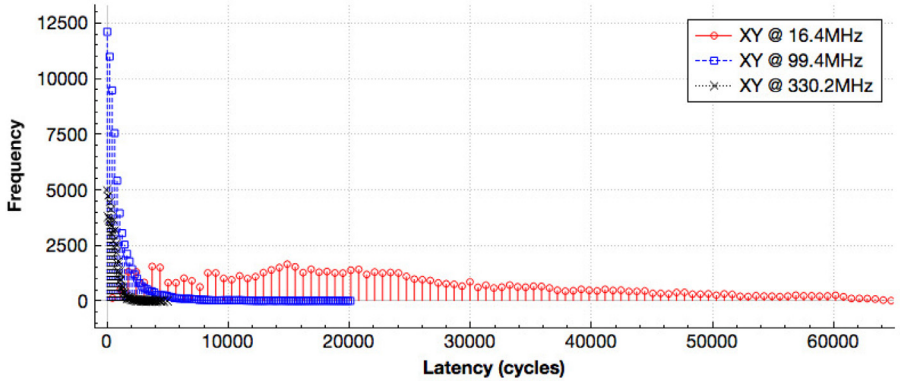

Fig. 11. Latency histogram for XY routing.

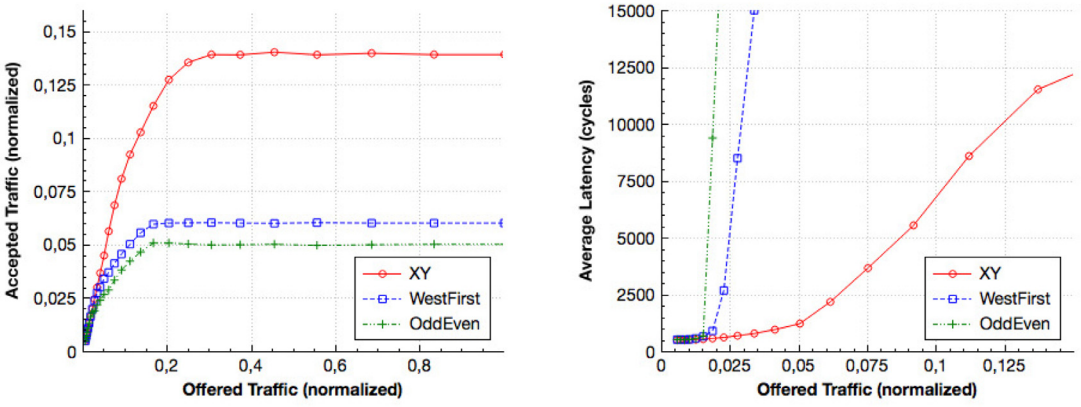

Fig. 12. Results: (a) throughput; and (b) average latency.

panel to configure up to five experiments, and the user can tweak the parameters to define different 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 pattern 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 for a communication flow). What we change is the network capacity (e.g., 3.2 Gbps at 100 MHz or 320 Mbps at 10 MHz, for a 32-bit data channel). This approach enables the user to evaluate whether a given configuration meets the application requirements.

The example in Fig. 7 depicts five on-chip communication architectures with clock frequencies ranging from 100 to 10 MHz (in steps of -10 MHz). Considering that the data 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).

After configuring the experiments, the user must click on the Run button to start the simulations. *RedScarf* then generates the configuration files for each experiment configuration and runs the simulators for each clock frequency. In the example shown in Fig. 7, *RedScarf* generates five *config* files and runs ten simulations for each configuration, one for each clock frequency. Therefore, it runs 50 experiments.

A simulation executes batches and a batch can be processed either once only, or repeated until the simulation stops, depending on the stop condition method chosen. Three criteria are available to stop a simulation: (*i*) When the network has delivered all the packets defined for all the batches, running each batch once; (*ii*) when a specified simulated time (in cycles or ns) is reached; (*iii*) when the network has delivered a specified number of packets. In the first criterion, each node injects packets until the number of packets defined in the traffic model is reached (the batches do not restart). This method implies that the terminal instrumentation modules stop the injection of packets before the end of the simulation. In the second and third criteria, the nodes inject packets continuously, independently of the predefined number of packets to be injected, and the creation of new packets does not stop until the *StopSim* module interrupts the simulation (the batches restart).

When a simulation finishes, a traffic analysis tool reads the log file of each terminal instrumentation module and calculates the performance metrics. The user can access these metrics using the Performance Analysis tab (Fig. 8). *RedScarf* exhibits the results in different ways: (*i*) Tables; (*ii*) charts; and (*iii*) waveforms. The user can select a range of packets for analysis, which enables the warm-up and drain phases to be considered.

3.8. Extra features

Besides the characteristics mentioned above, *RedScarf* provides some additional features, including:

1. *Collaborative development*: The development team of *RedScarf* uses Git [23] for versioning and revision control, as well for sharing source codes and documentation among its developers and users.
2. *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 user can define the maximum number of cores to employ in *RedScarf* settings.
3. *Internationalized interface*: The GUI of *RedScarf* is currently available in English and Portuguese. However, we have designed and implemented this GUI to facilitate its portability to other languages, without the need to recompile the software.
4. *Results exporting: RedScarf* can export the results to a CSV (Comma Separated Value) file for analysis in other tools, such as spreadsheets and graphics utilities.
5. *Setup saving and loading: RedScarf* allows the system configuration and the traffic model to be saved in an XML (eXtensible Markup Language) file for later reuse.
6. *Results archiving and restoring*: After running a simulation, the user can save (archive) the simulation results (logs and reports) for further analysis.

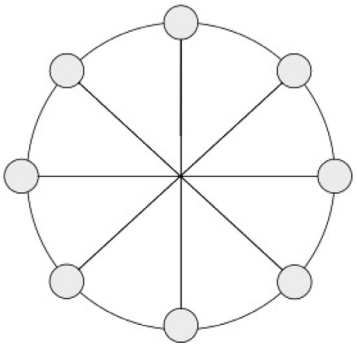

Fig. 13. The Chordal Ring topology.

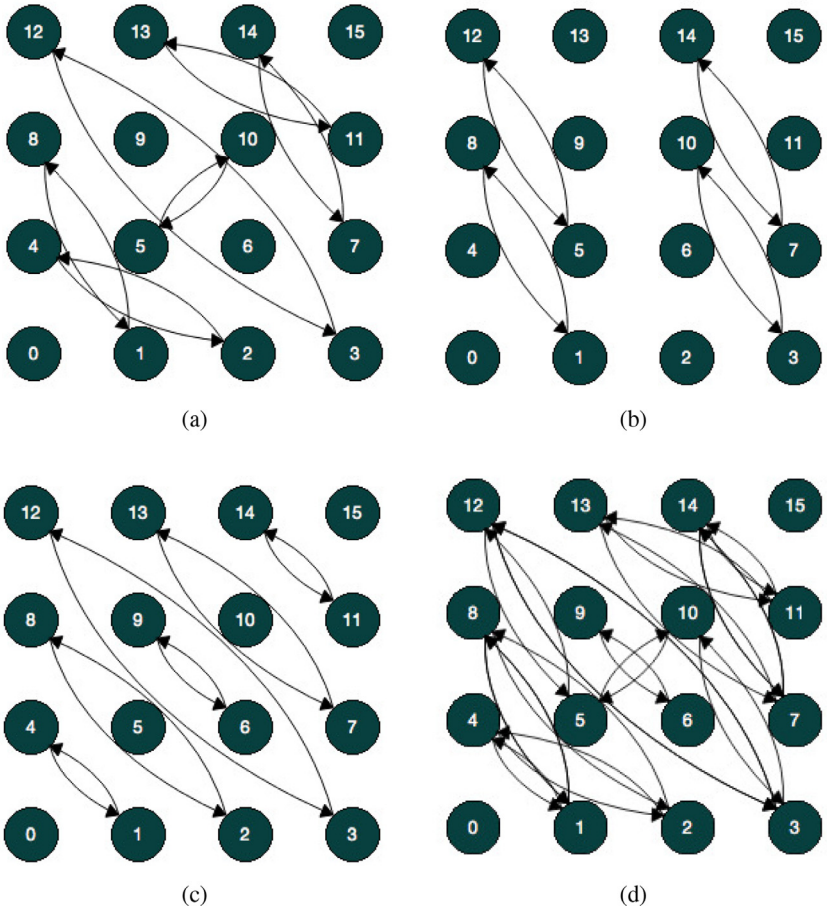

Fig. 14. Traffic patterns on the 2D Mesh: (a) Bit-Reversal; (b) Butterfly; (c) Matrix Transpose; and (d) Combined.

3.9. Implementation issues

Following the MVC approach, the front-end of *RedScarf* contains 18,500 lines of code, comprising 5.5 Klines in the Model, 4.2 Klines in the Control, and 5.5 Klines in the View. It also includes more 3.3Klines generated by Qt. It is worth noting that we used only well established free software and technologies in the development of *RedScarf*.

4. Experiments

This section presents experiments that demonstrate the use of *RedScarf* to evaluate the performance of on-chip communication architectures. The first experiment comprises the comparison of routing algorithms for a 2D Mesh topology and illustrates the basic resources available for performance analysis. The second experiment illustrates the process of adding a new topology to *RedScarf* with the comparison with reference topologies.

4.1. Experiment 1: Primary resources for analysis

This experiment runs over a 64-bit 10 × 10 2D Mesh interconnecting a system with 100 nodes. Each node sends 768-bit packets at 320 Mbps to all the other nodes in the system following a Uniform distribution.

Fig. 9 illustrates a resource of *RedScarf* that enables the traffic model defined for a single node to be viewed. It is also possible to select multiple nodes and display all the communication flows (as shown in Fig. 5). As we can see, this graphical resource is a useful tool to verify the configuration of the traffic pattern defined by the user, and, also, to identify potential hot spots for any system size (from few dozens up to hundreds of nodes). However, it is worth mentioning that it does not take into account the routes that will be taken by each packet because it depends on the topology and routing algorithm chosen.

For this experiment, we have defined different NoC configurations to compare the performance of three routing algorithms for a 2D Mesh topology: XY, West-First, and Odd-Even. In all the configurations, we have set up the routers with these parameters: Wormhole switching, Credit-based flow control, Round-Robin arbitration, and a single 4-flit FIFO buffer at each input channel.

Next, we configured the simulator to run 25 simulations for each configuration, with the clock frequency ranging from 1 MHz to 1 GHz, according to an incremental step defined by an exponential function (the exponent parameter equals -0.20). We then have set the simulation stop condition for the delivery of 100,000 packets. It is worth noting that the channel bandwidth varies from 64 Mbps (64 bits × 1 MHz) to 64 Gbps. Since each communication flow injects packets at 320 Mbps, at the lower operating frequencies, the offered traffic occupies 100% of the channel capacity. As the operating frequency rises, the offered traffic decreases. For instance, it equals 5.0% of the channel capacity when the network works at 1 GHz.

In *RedScarf*, after simulating all the configurations, the user can select a range of packets to analyze. For instance, he/she can disregard the packets transferred during the warm-up phase to evaluate the network performance in the steady state. In this experiment, we have ignored the first 30,000 packets.

RedScarf offers several resources for analysis of the experimental results. The user can visualize the results of a simulation using table-based

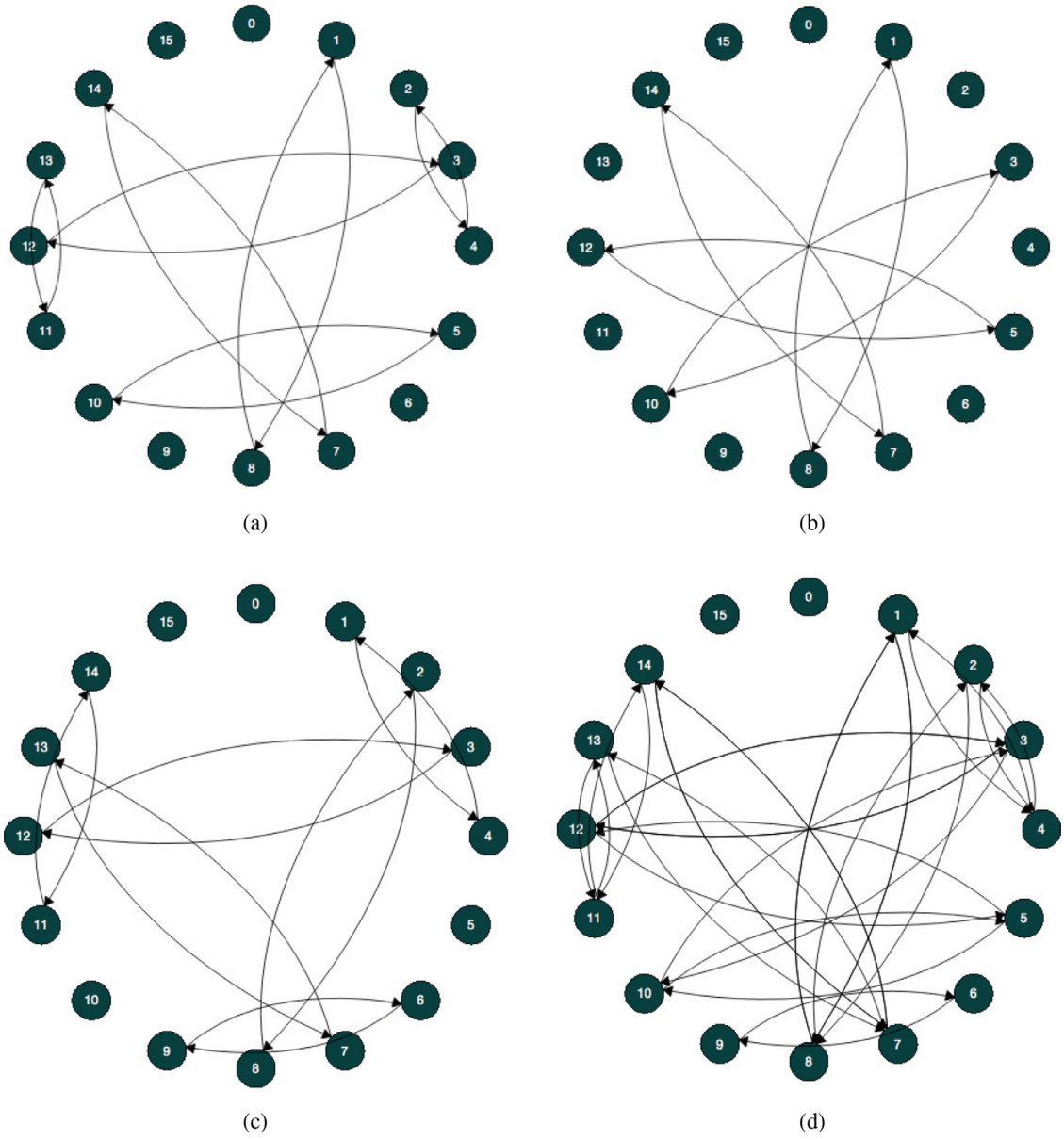

Fig. 15. Traffic patterns on the Chordal Ring: (a) Bit-Reversal; (b) Butterfly; (c) Matrix Transpose; and (d) Combined.

reports like the one of Fig. 10, which presents the performance metrics regarding the simulation of the XY routing. As the operating frequency decreases (read the table from bottom to top), the relative offered traffic increases. At higher clock frequencies, the network accepts almost all of the traffic injected by the nodes. On the other hand, at lower clock frequencies, the NoC is not able to admit much of the offered traffic. At 16 MHz, the network reaches its saturation point, and the normalized accepted traffic (i.e., the throughput) remains limited to 0.14 flits/cycle/node.

In *RedScarf*, the user can analyze the results using several types of graph. These graphs are generated automatically (no third-party viewer is necessary) and can be exported to PNG (Portable Network Graphics) files. These are some of the many features that make *RedScarf* a user-friendly tool.

One of the graphs available in *RedScarf* is the latency histogram, which exhibits the number of packets that go through each level of latency. For instance, Fig. 11 depicts the latency histogram for the XY routing for three different clock frequencies: 330.2 MHz, 99.4 MHz, and 16.4 MHz. At 330.2 MHz, we notice that the latencies (in clock cycles) are lower and concentrated. As the operating frequency decreases, the latencies become higher and more dispersed. At 16.4 MHz (the saturation point), the latencies are excessively high and quite sparse.

The user can also visualize graphs that illustrate how the throughput and the average latency vary with the offered traffic. Fig. 12 examines the performance of the three routing algorithms evaluated. In the graphs, we can clearly recognize the saturation point at which the NoC does not accept any further traffic (Fig. 12(a)) and the latency rises exponentially (Fig. 12(b)). As we can see, the graphs identify that the deterministic algorithm (XY) admits more traffic than the partially adaptive algorithms, which tend to concentrate the traffic, creating hot spots that increase contention in the network.

This experiment is similar to the one described in [11], in which the authors compared two NoC simulators: Noxim [11] and BookSim [14]. The results obtained were also similar.

4.2. Experiment 2: Adding a new topology

This experiment illustrates the process of adding and evaluating the Chordal Ring topology applied in the Spidergon NoC [24]. This topology is a bidirectional ring with a cross connection between each pair of nodes (Fig. 13), which reduces the network diameter compared to conventional rings. The Chordal Ring applies the Cross-First routing. Depending on the position of the destination node, it first uses the cross connection and then forwards the packets using the clockwise or the anti-clockwise route.

To implement the Chordal Ring, we applied the process presented in Section 3.5. We first described the topology and the routing algorithm plug-ins and then updated the system configuration file. After compiling the classes, we carried out the necessary tests to verify the simulation models.

Following, we examine the performance of the Chordal Ring against the 2D Mesh for a 16-node system. For this experiment, we configured both NoCs with this set of attributes: 32-bit channels, Wormhole routing, Credit-based flow control, Round-Robin arbitration, and 4-flit buffers at the input channels. The 2D Mesh uses the XY routing algorithm.

We then applied three permutation traffic models (Bit-Reversal, Butterfly and Matrix Transpose) and a combination of both (named Combined). Figs. 14 and 15 depict these traffic models on the 2D Mesh and the Chordal Ring. Each communication flow injects packets into the network at a constant bit rate of 2.56 Gbps. A packet comprises a header and a 128-bit payload and represents a cache line transfer.

In the experiment, we varied the operating frequency from 10 to 700 MHz (with an incremental step of 50 MHz) and set the simulation stop condition to the delivery of 10,000 packets. In the analysis, we discarded the first 3,000 packages to examine the network in the steady state. We must notice that the channel bandwidth ranges from 320 Mbps to 22.4 Gbps. Hence, the offered traffic varies from 100% to 11.4% of the channel capacity, respectively.

Table 3 and Figs. 16 and 17 present the simulation results. As we can observe, the Chordal Ring accepts more traffic than the 2D Mesh, with

Table 3
Performance comparison: Chordal Ring (CR) *vs* 2D mesh.

Traffic pattern	Bit Reversal		Butterfly		Matrix		Combined	
Topology	CR	Mesh	CR	Mesh	CR	Mesh	CR	Mesh
Max. Throughput (flits/cycle/node)	0.29	0.19	0.29	0.14	0.21	0.21	0.27	0.25
Average latency (cycles)	16.3	22.4	14.0	19.0	17.5	20.7	17.9	20.7
Minimal Fclk* (MHz)	260	410	160	260	260	410	310	460

* To fulfill the communication deadline requirements.

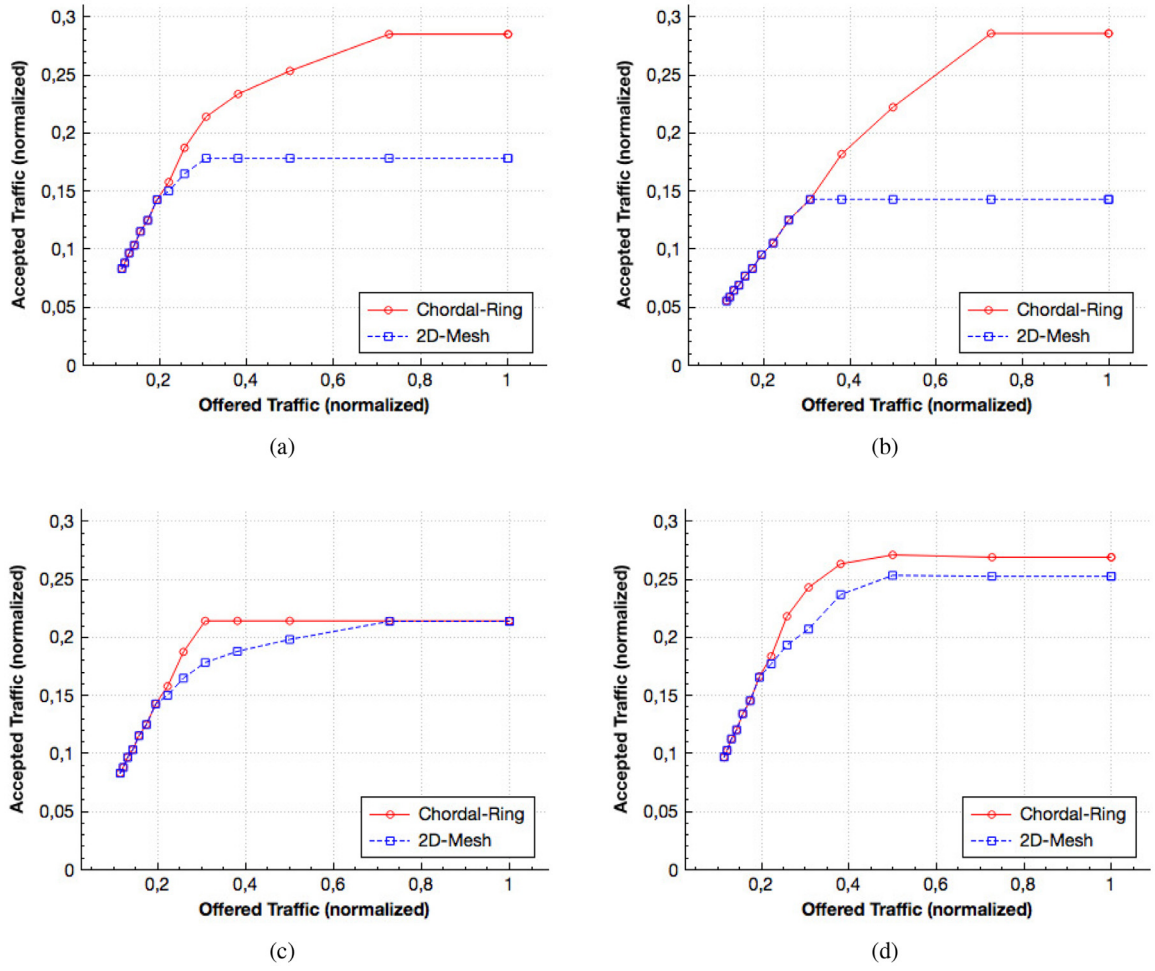

Fig. 16. Throughput for different traffic patterns: (a) Bit-Reversal; (b) Butterfly; (c) Matrix Transpose; and (d) Combined.

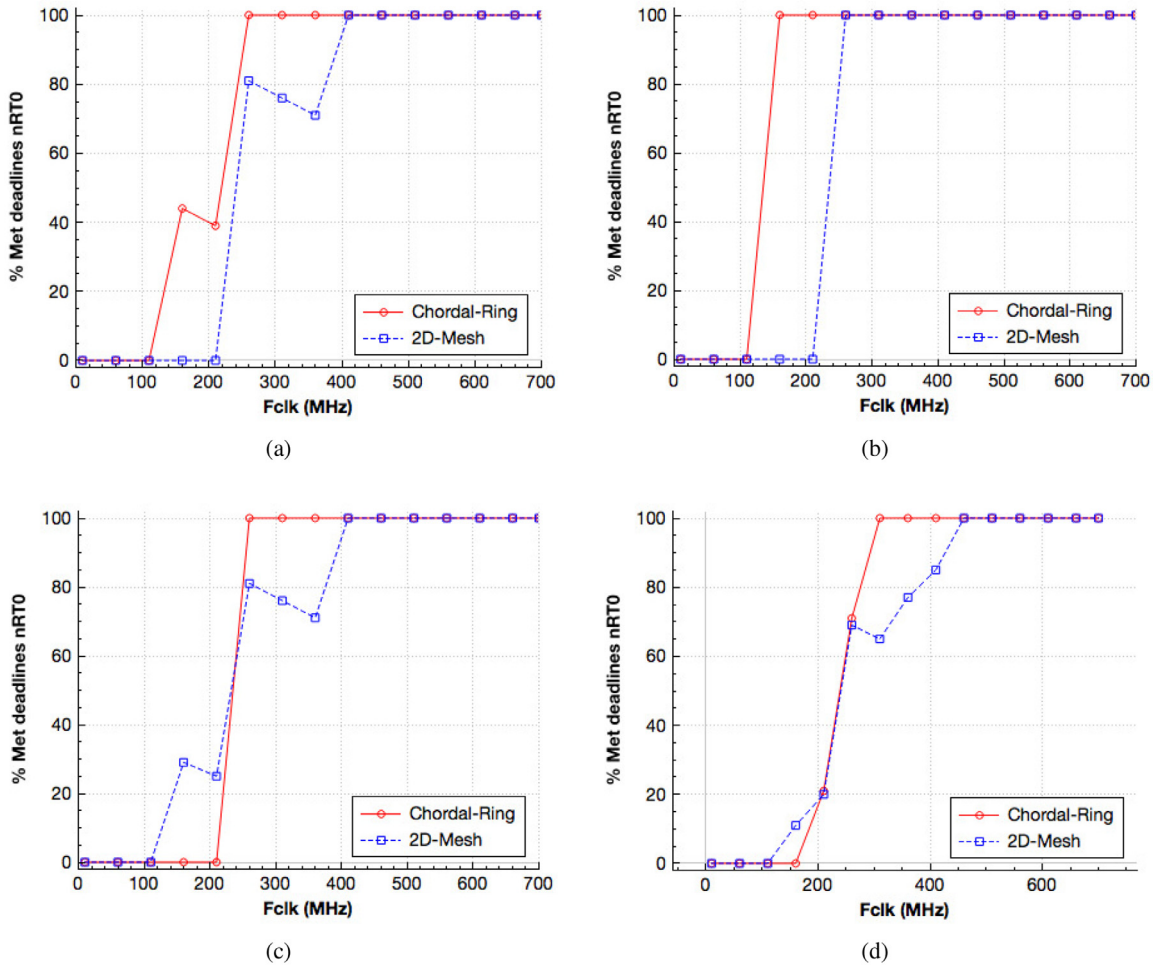

Fig. 17. Deadline fulfillment for different traffic patterns: (a) Bit-Reversal; (b) Butterfly; (c) Matrix Transpose; and (d) Combined.

lower average latency. The only traffic pattern in which the 2D Mesh accepts as much traffic than the Chordal Ring is the Matrix Transpose. Nevertheless, as we can see in Fig. 16(c), it starts to saturate earlier and has a higher average latency.

In this experiment, we introduce a further performance metric available in *RedScarf*: The deadline fulfillment at a given operating frequency. Considering that the user specified the required maximum end-to-end delay for the communication flows, this metric expresses the rate of packets the network is capable of delivering on time. For the traffic models described above, we set the maximum end-to-end delay to 150 ns, as the authors of [25] proposed for this type of traffic.

Fig. 17 depicts how the NoC fulfills the required deadlines as the clock frequency increases. Examining the graphs and Table 3, we can see that the Chordal Ring can operate at a clock frequency 36% lower, on average, than the 2D Mesh. Given that one significant contributor to the power consumption of a chip is the global clock tree [26], reducing the required clock frequency to meet the communication requirements contributes to energy saving.

To illustrate the reference architectures available in *RedScarf*, we also compared the Chordal Ring topology with the Crossbar, the 2D Torus, and the 3D Mesh, besides the 2D Mesh. Fig. 18 presents the results of a simulation that compares the performance of the topologies mentioned above in a 16-node system under the Matrix Transpose traffic. The communication flows use the same configuration as the previous simulations described in this experiment.

As we can see, for the Matrix Transpose traffic model, the Chordal Ring offers performance in between the other architectures. It accepts less traffic than the Crossbar and 2D Torus, but its latency and the minimum operating frequency to fulfill the deadline requirements is similar to the ones of the 2D Torus (their curves are superimposed on Fig. 18(b) and (c)). Because the Crossbar has limited scalability since its cost increases quadratically with the number of ports, and the 2D Torus uses long end-around connections that limit its performance, the experiment demonstrates that the Chordal Ring could be a good choice for the proposed scenario.

4.3. Discussion

The experiments presented state how powerful and versatile *RedScarf* is, as it can encompass several different assessments on NoC design parameters. Its graphical interface facilitates traffic modeling and the configuration of experiments, reducing the time necessary to conduct the design space exploration. The diversity of resources available to exhibit the simulation results provides the means required for data analysis, and are also useful for producing reports of the experiments performed.

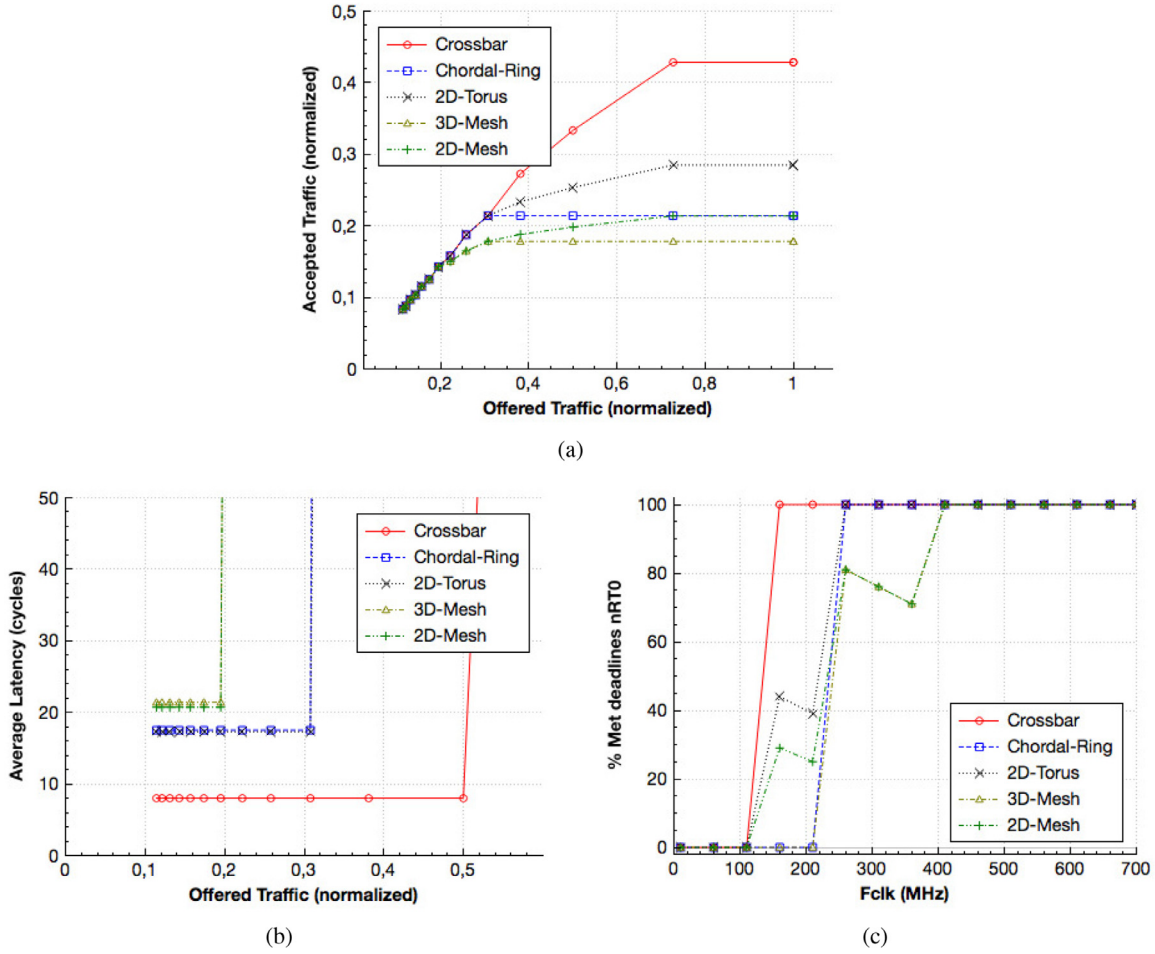

Fig. 18. Comparing the performance of the Chordal Ring with four reference NoCs: (a) Throughput; (b) latency; and (c) deadline fulfillment.

5. Conclusions

This paper presents *RedScarf*, a simulation environment for performance evaluation of on-chip communication architectures. *RedScarf* is an open-source, user-friendly, and multi-platform software, and integrates a set of tools that provide the means necessary to facilitate the work of exploring the design space of on-chip interconnects. Despite the number of resources already available in *RedScarf*, new tools and features are under development. Our primary goal is to integrate task mapping tools and benchmarks models. Users will be able to choose a different set of tasks or well-known benchmarks to be evaluated on the reference interconnect architectures available in the environment as well as their own models. We also intend to add new traffic generation features, including the ability to inject malicious traffic to attack the NoC security properties, as well as run traces of actual applications to evaluate real systems. Moreover, we plan to integrate a multi-objective optimization tool [27] to *RedScarf* for automatic exploration of the NoC design space, integrating power models to investigate the energy efficiency of the different NoC configurations. The idea is to provide energy- and performance-aware application mapping tools, similar it is done in [28], to improve the design space exploration. Finally, to cover emerging architectures, we are considering developing models to evaluate the use of wireless and photonic technologies. All current and future features of *RedScarf* are intended to facilitate its usage and adoption by the community. We hope that it will become a reference tool for Research and Education in NoCs.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.sysarc.2019.101633.

References

[1] A.B. Kahng, B. Li, L.S. Peh, K. Samadi, ORION 2.0: a power-area simulator for interconnection networks, IEEE Trans. Very Large Scale Integr.(VLSI) Syst. 20 (1) (2012) 191–196, doi:10.1109/TVLSI.2010.2091686.

[2] A. Adriahantenaina, H. Charlery, A. Greiner, L. Mortiez, C.A. Zeferino, SPIN: a scalable, packet switched, on-chip micro-network, in: Design, Automation and Test in Europe, 2003, p. 4.

[3] K. Macdonald, C. Nitta, M. Farrens, V. Akella, PDG_Gen: a methodology for fast and accurate simulation of on-chip networks, IEEE Trans. Comput. 63 (3) (2014) 650–663, doi:10.1109/TC.2012.140.

[4] L. Benini, G. De Micheli, Networks on Chip: Technology and Tools, Morgan Kaufmann, San Francisco, 2006.

[5] H. Javaid, Y. Yachide, S.M.M. Shwe, H. Bokhari, S. Parameswaran, FALCON: a framework for hierarchical computation of metrics for component-based parameterized socs, in: 51st ACM/EDAC/IEEE Design Automation Conference, 2014, pp. 1–6, doi:10.1145/2593069.2593138.

[6] E.A. Silva, D. Menegasso, S. Vargas, C.A. Zeferino, *RedScarf*: a user-friendly multi–platform Network-on-Chip simulator, in: 2017 VII Brazilian Symposium on Computing Systems Engineering (SBESC), 2017, pp. 71–78.

[7] M. Gaur, B. Al-Hashimi, V. Laxmi, N. R, N. Choudhary, L. Jain, NIRGAM: asimulator for NoC interconnect routing and application modeling, 2007. http://nirgam.ecs.soton.ac.uk/.
[8] R.H. Moreno, NoC Simulator, 2007. http://nirgam.ecs.soton.ac.uk.
[9] M. Coppola, S. Curaba, M.D. Grammatikakis, R. Locatelli, G. Maruccia, F. Papariello, OCCN: a NoC modeling framework for design exploration, 2011. http://occn.sourceforge.net/occn_wpaper_pre_rev.pdf.
[10] N. Jiang, J. Balfour, D.U. Becker, B. Towles, W.J. Dally, G. Michelogiannakis, J. Kim, A detailed and flexible cycle-accurate Network-on-Chip simulator, in: IEEE Int. Symp. on Performance Analysis of Systems and Software, 2013, pp. 86–96, doi:10.1109/ISPASS.2013.6557149.
[11] V. Catania, A. Mineo, S. Monteleone, M. Palesi, D. Patti, Cycle-accurate Network on Chip simulation with Noxim, ACM Trans. Model. Comput. Simul. 27 (1) (2016) 4:1–4:25, doi:10.1145/2953878.
[12] E.A. Carara, R.P. de Oliveira, N.L.V. Calazans, F.G. Moraes, HeMPS - a framework for NoC-based MPSoC generation, in: IEEE Int. Symp. on Circuits and Systems, 2009, pp. 1345–1348, doi:10.1109/ISCAS.2009.5118013.
[13] N. Binkert, B. Beckmann, G. Black, S.K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D.R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M.D. Hill, D.A. Wood, The Gem5 simulator, SIGARCH Comput. Archit. News 39 (2) (2011) 1–7, doi:10.1145/2024716.2024718.
[14] W.J. Dally, B.P. Towles, Principles and Practices of Interconnection Networks, Elsevier, 2004.
[15] N.E. Jerger, L.-S. Peh, On-Chip Networks, 1st, Morgan and Claypool, Madison, 2009.
[16] A. Shacham, K. Bergman, L.P. Carloni, Photonic Networks-on-Chip for future generations of chip multiprocessors, IEEE Trans. Comput. 57 (9) (2008) 1246–1260, doi:10.1109/TC.2008.78.
[17] C. Sun, C.O. Chen, G. Kurian, L. Wei, J. Miller, A. Agarwal, L. Peh, V. Stojanovic, DSENT: a tool connecting emerging photonics with electronics for opto-electronic Networks-on-Chip modeling, in: 2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip, 2012, pp. 201–210, doi:10.1109/NOCS.2012.31.
[18] H. Temuçin, K.M. İmre, Scheduling computation and communication on a software-defined photonic Network-on-Chip architecture for high-performance real–time systems, J. Syst. Archit. 90 (2018) 54–71.
[19] D. DiTomaso, A. Kodi, D. Matolak, S. Kaya, S. Laha, W. Rayess, A-WiNoC: adaptive wireless Network-on-Chip architecture for chip multiprocessors, IEEE Trans. Parallel Distrib.Syst. 26 (12) (2015) 3289–3302, doi:10.1109/TPDS.2014.2383384.
[20] A. Ganguly, K. Chang, S. Deb, P.P. Pande, B. Belzer, C. Teuscher, Scalable hybrid wireless Network-on-Chip architectures for multicore systems, IEEE Trans. Comput. 60 (10) (2011) 1485–1502, doi:10.1109/TC.2008.86.
[21] Synopsys, Describing Synthesizable RTL in SystemC, 1.1, Synopsys, USA, 2002.
[22] L. Tedesco, A. Mello, D. Garibotti, N. Calazans, F. Moraes, Traffic generation and performance evaluation for Mesh-based NoCs, in: 18th Symp. on Integrated Circuits and Systems Design, 2005, pp. 184–189, doi:10.1109/SBCCI.2005.4286854.
[23] University of Vale do Itajai – Univali, *RedScarf*: a GUI-oriented simulation environment for the performance evaluation of Networks-on-Chip, 2019. https://leds-lab.github.io/redscarf.
[24] M. Coppola, R. Locatelli, G. Maruccia, L. Pieralisi, A. Scandurra, Spidergon: a novel on-chip communication network, 2004 International Symposium on System-on-Chip, 2004, doi:10.1109/ISSOC.2004.1411133. 15.
[25] E. Bolotin, I. Cidon, R. Ginosar, A. Kolodny, QNoC: QoS architecture and design process for Network-on-Chip, J. Syst. Archit. 50 (2–3) (2004) 105–128.
[26] D.J. Gebhardt, Energy-efficient Design of an Asynchronous Network-on-Chip, The University of Utah, 2011.
[27] J.V. Bruch, E.A.d. Silva, C.A. Zeferino, L.S. Indrusiak, Deadline, energy and buffer--aware task mapping optimization in NoC-based SoCs using genetic algorithms, in: 2017 VII Brazilian Symposium on Computing Systems Engineering (SBESC), 2017, pp. 86–93.
[28] M.O. Agyeman, A. Ahmadinia, N. Bagherzadeh, Energy and performance-aware application mapping for inhomogeneous 3d Networks-on-Chip, J. Syst. Archit. 89 (2018) 103–117.

Eduardo A. da Silva received his M.Sc. in Applied Computer Science from the University of Vale do Itajaí, Brazil, in 2017. He is assistant professor of the Computer Science Department of the University of Vale do Itajaí-Univali, Brazil, since 2018. His topics of interest are Networks-on-Chip, Digital Circuits Design, and Embedded Systems.

Márcio E. Kreutz received his Ph.D. in Computer Science from the Federal University of Rio Grande do Sul, Brazil, in 2005. He is adjunct professor of the Department of Informatics and Applied Mathematics of the Federal University of Rio Grande do Norte – UFRN, Brazil, since 2009. His topics of interest are Systems-on-Chip, Networks-on-Chip, Digital Circuits Design, and Embedded Systems.

Cesar A. Zeferino received his Ph.D. in Computer Science from the Federal University of Rio Grande do Sul, Brazil, in 2003, with a sandwich internship at the Universit Pierre et Marie Curie, Paris, France. He is full professor of the Computer Engineering Department of the University of Vale do Itajaí – Univali, Brazil, since 2002. He is the Research and Graduate Manager of Univali and head of the Laboratory of Embedded and Distributed Systems. His topics of interest are Networks-on-Chip, Digital Circuits Design, Hardware Accelerators, Embedded Systems, Systems-on-Chip, and Computer Science Education.