

**UNIVERSIDADE DO VALE DO ITAJAÍ
CENTRO DE CIÊNCIAS TECNOLÓGICAS DA TERRA E DO MAR
CURSO DE CIÊNCIA DA COMPUTAÇÃO**

REDSCARF
AMBIENTE PARA AVALIAÇÃO DE DESEMPENHO DE REDE-EM-CHIP

por

Eduardo Alves da Silva

Itajaí (SC), novembro de 2014

**UNIVERSIDADE DO VALE DO ITAJAÍ
CENTRO DE CIÊNCIAS TECNOLÓGICAS DA TERRA E DO MAR
CURSO DE CIÊNCIA DA COMPUTAÇÃO**

REDSCARF
AMBIENTE PARA AVALIAÇÃO DE DESEMPENHO DE REDE-EM-CHIP

Área de Sistemas Embarcados

por

Eduardo Alves da Silva

Relatório apresentado à Banca Examinadora
do Trabalho Técnico-científico de Conclusão
do Curso de Ciência da Computação para
análise e aprovação.

Orientador: Cesar Albenes Zeferino, Dr.

Itajaí (SC), novembro de 2014

*Dedico este trabalho a todas as pessoas envolvidas direta e indiretamente nesta realização.
Tanto para o trabalho quanto para a graduação.
Em especial aos meus familiares que me deram o suporte na vida, necessário para chegar até
aqui.*

*E ao meu orientador, que me acolheu durante todo o período da faculdade, me orientou,
ensinou e fez com que meu conhecimento não só fosse ampliado, mas que fosse ampliado com
qualidade, sempre exigindo mais de mim.*

AGRADECIMENTOS

Agradeço a Deus pela vida que me foi concedida, pelos problemas que me foram impostos e principalmente pela capacidade e intelecto que me foi dado para resolução destes problemas e para meu crescimento como humano.

Agradeço a minha família, pelo apoio, incentivo e adoração que tiveram por mim na realização deste curso. Em especial aos meus pais que foram a fonte de suporte, energia e motivação para a realização desta conquista.

Ao meu orientador que sempre esteve ao meu lado para ensinar-me mais e mais. Para ser meu amigo, resolver minhas dúvidas, jogar aquele futebol semanal e inspirar-me a sempre buscar a excelência naquilo que faço.

Aos mestres que sempre transmitiram as informações necessárias na construção de meu conhecimento, sabedoria nas áreas da computação, e que fizeram parte do meu dia-a-dia durante os anos desta caminhada.

Aos colegas de laboratório, que sempre discutiram sobre os mais variados assuntos na vivência cotidiana e se fizeram amigos, fazendo com que minha concepção sobre as coisas se tornasse cada vez mais crítica e, também pelo auxílio na execução das tarefas complexas e aprendizagem diária que cada um proporcionou.

Aos colegas de aula, que fizeram parte de todo o processo de obtenção de conhecimento, contribuindo com cada colocação, questionamento e abordagem feita em classe e extraclasse.

Aos amigos que sempre me apoiaram nesta passagem, próximos ou distantes, fizeram parte disto.

O meu muito OBRIGADO!



Aprenda como se fosse viver para sempre. Viva como se fosse morrer amanhã.
(Mahatma Gandhi)

A mente que se abre a uma nova ideia jamais voltará ao seu tamanho original.
(Albert Einstein)

RESUMO

SILVA, Eduardo A. **RedScarf**: ambiente para avaliação de desempenho de Rede-em-Chip. Itajaí, 2014. 87p. Trabalho Técnico-científico de Conclusão de Curso (Graduação em Ciência da Computação) – Centro de Ciências Tecnológicas da Terra e do Mar, Universidade do Vale do Itajaí, Itajaí, 2014.

A era dos sistemas multi-core demanda agilidade no processo de validação dos projetos de sistemas integrados, em que se busca através da arquitetura de comunicação o design mais sofisticado para obtenção da relação custo-desempenho dos chips fabricados. A arquitetura de comunicação que vem ganhando destaque nesses sistemas é a de Redes-em-Chip. Para realizar a validação dos requisitos de desempenho da aplicação, nesta tecnologia, são necessárias ferramentas que agilizam o processo de verificação. Uma abordagem que possui flexibilidade e baixo custo é através da simulação. E para automatizar o processo de configuração, são criados ambientes de simulação. É neste contexto que este trabalho está inserido, em que se propôs a criar uma ferramenta, baseada num ambiente existente, afim de, melhorá-lo, evoluí-lo e torná-lo referência em simulação para avaliação de desempenho da Rede-em-Chip alvo deste trabalho. Foi desenvolvida uma nova versão desse ambiente, a qual possui como principais características o suporte multiplataforma e a execução *multi-thread* de modelos de simulação para avaliação do desempenho de uma Rede-em-Chip descrita em SystemC. O ambiente integra diversas ferramentas que automatizam a configuração e a execução de experimentos, oferecendo recursos para facilitar a análise de resultados.

Palavras-chave: Redes-em-Chip. Avaliação de desempenho. Simulação.

ABSTRACT

Multi-core systems era demand agility in the validation process of the integrated systems project, where they search through communication architecture the most sophisticated design to obtain the cost-performance ratio of chips manufactured. Communication architecture that is gaining prominence in these systems is the Networks-on-Chip. To perform the validation of application performance requirements, this technology, are necessary tools that speed up the process of verification. An approach that has the flexibility and low cost is through simulation. And to automate the configuration process, simulation environments are created. This work is inserted in this context and aimed at the development of a new version of an existing environment for performance evaluation of a Network-on-Chip using simulation. This new version is multiplatform and offers support for the multithreaded execution of SystemC-based simulation models. The environment integrates several tools that automate the configuration and the execution of performance evaluation experiments, providing resources that facilitate the analysis of results.

Keywords: Networks-on-Chip. Performance Evaluate. Simulation.

LISTA DE FIGURAS

Figura 1.	Sistema de coordenadas da SoCIN	23
Figura 2.	Organização do ParIS	24
Figura 3.	Terminais de instrumentação acoplados à rede	27
Figura 4.	Relação da vazão pelo tráfego oferecido, obtida de uma malha 8x8 com tráfego complemento.....	29
Figura 5.	Ferramentas e fluxo utilizado no ambiente BrownPepper.....	32
Figura 6.	BrownPepper: aba de configuração do sistema	33
Figura 7.	BrownPepper: aba de configuração dos experimentos de simulação	34
Figura 8.	BrownPepper: aba de visualização dos resultados da análise de desempenho.....	34
Figura 9.	Arquitetura do ambiente	45
Figura 10.	Fluxo de uso da ferramenta.....	48
Figura 11.	Fluxo de uso da ferramenta.....	49
Figura 12.	Aceleração da execução <i>multi-thread</i>	56
Figura 13.	Resultados RedScarf: (a) exemplo de gráfico; (b) resultados exportados para o MS Excel	59
Figura 14.	Topologias diretas: (a) malha, (b) toróide e (c) hipercubo	67
Figura 15.	Topologias Indiretas: (a) árvore gorda e (b) <i>butterfly</i>	68
Figura 16.	Roteamento numa malha 2D 3x3: (a) malha com o posicionamento dos nodos e (b) caminho percorrido no algoritmo XY com os passos enumerados do nodo “0,0” para o “2,2”	70
Figura 17.	Segmentação de uma mensagem em pacote, divisão do pacote em <i>flits</i> e divisão do <i>flit</i> em <i>phits</i>	71
Figura 18.	Estrutura geral de um roteador	73
Figura 19.	Diagrama dos Requisitos Funcionais.....	75
Figura 20.	Diagrama dos Requisitos Não Funcionais.....	76
Figura 21.	Diagrama das Regras de Negócio.....	77
Figura 22.	Diagrama de Classes (visão macro).....	78
Figura 23.	Diagrama de Casos de Usos	79
Figura 24.	Diagrama de Atividades	80
Figura 25.	Telas de configurações do sistema	81
Figura 26.	Tela de simulação do sistema	82
Figura 27.	Tela de análise de desempenho.....	83
Figura 28.	Telas de previsualização de tráfego	84
Figura 29.	Tela de configuração de tráfego	85
Figura 30.	Protótipo de tela de visualização das formas de ondas.....	86
Figura 31.	Tela de relatório	86
Figura 32.	Tela de visualização dos gráficos	87

LISTA DE QUADROS

Quadro 1.	Comparação das soluções concorrentes disponíveis no Qt.....	39
Quadro 2.	Exemplos de usos das tecnologias concorrentes disponíveis no Qt	39
Quadro 3.	Comparativo dos trabalhos relacionados	42
Quadro 4.	Comparativo das versões da ferramenta	60

LISTA DE EQUAÇÕES

Equação (1).....	25
Equação (2).....	25
Equação (3).....	25
Equação (4).....	26
Equação (5).....	26

SUMÁRIO

1 INTRODUÇÃO.....	13
<i>1.1 PROBLEMATIZAÇÃO</i>	<i>14</i>
<i>1.1.1 Formulação do Problema.....</i>	<i>14</i>
<i>1.1.2 Solução Proposta.....</i>	<i>14</i>
<i>1.2 OBJETIVOS</i>	<i>16</i>
<i>1.2.1 Objetivo Geral</i>	<i>16</i>
<i>1.2.2 Objetivos Específicos.....</i>	<i>17</i>
<i>1.3 METODOLOGIA</i>	<i>17</i>
<i>1.4 ESTRUTURA DO TRABALHO</i>	<i>18</i>
2 FUNDAMENTAÇÃO TEÓRICA	19
<i>2.1 REDES-EM-CHIP</i>	<i>19</i>
<i>2.2 A REDE SOCIN</i>	<i>21</i>
<i>2.3 AVALIAÇÃO DE DESEMPENHO EM REDES-EM-CHIP</i>	<i>26</i>
<i>2.4 O AMBIENTE BROWNPEPPER</i>	<i>30</i>
<i>2.5 O FRAMEWORK QT</i>	<i>35</i>
<i>2.5.1 Processos no Qt</i>	<i>37</i>
<i>2.5.2 Multi-threading no Qt.....</i>	<i>38</i>
<i>2.6 TRABALHOS RELACIONADOS</i>	<i>40</i>
3 Desenvolvimento.....	43
<i>3.1 VISÃO GERAL</i>	<i>43</i>
<i>3.2 ARQUITETURA DO SOFTWARE</i>	<i>44</i>
<i>3.2.1 Interface</i>	<i>45</i>
<i>3.2.2 Controle</i>	<i>46</i>
<i>3.2.3 Modelo</i>	<i>47</i>
<i>3.2.4 Fluxo das ferramentas</i>	<i>48</i>
<i>3.3 PROJETO.....</i>	<i>49</i>
<i>3.3.1 Requisitos funcionais</i>	<i>51</i>
<i>3.4 IMPLEMENTAÇÃO</i>	<i>53</i>
<i>3.5 RESULTADOS</i>	<i>55</i>
<i>3.6 FUNCIONALIDADES PARA A PRÓXIMA VERSÃO</i>	<i>60</i>
4 Conclusões.....	62
REFERÊNCIAS	64
Apêndice A. Detalhamento das propriedades das Redes-em-Chip	67
<i>A.1 TOPOLOGIA</i>	<i>67</i>
<i>A.2 ROTEAMENTO.....</i>	<i>68</i>
<i>A.3 CONTROLE DE FLUXO</i>	<i>70</i>
<i>A.4 MICROARQUITETURA DO ROTEADOR.....</i>	<i>72</i>
Apêndice B. Projeto de Software.....	74
<i>B.1 REQUISITOS FUNCIONAIS.....</i>	<i>75</i>
<i>B.2 REQUISITOS NÃO FUNCIONAIS.....</i>	<i>76</i>
<i>B.3 REGRAS DE NEGÓCIO</i>	<i>77</i>
<i>B.4 DIAGRAMA DE CLASSES</i>	<i>78</i>
<i>B.5 DIAGRAMA DE CASOS DE USO</i>	<i>79</i>
<i>B.6 DIAGRAMA DE ATIVIDADES.....</i>	<i>80</i>
<i>B.7 TELAS</i>	<i>81</i>
<i>B.7.1 Telas de configuração do sistema.....</i>	<i>81</i>

<i>B.7.2 Tela de simulação do sistema</i>	82
<i>B.7.3 Tela de análise de desempenho</i>	83
<i>B.7.4 Telas de previsualização de tráfego</i>	84
<i>B.7.5 Tela de configuração de tráfego</i>	85
<i>B.7.6 Tela de exibição das formas de ondas</i>	86
<i>B.7.7 Tela de relatório</i>	86
<i>B.7.8 Tela de visualização dos gráficos</i>	87
<i>ANEXO A. Manual do Usuário</i>	88

1 INTRODUÇÃO

O avanço das tecnologias de fabricação de circuitos eletrônicos viabilizou a integração de dezenas de componentes em um único chip de silício para a criação de sistemas computacionais integrados, os quais são denominados SoCs (Systems-on-Chip). Para atender as demandas de comunicação de tais sistemas, pesquisadores propuseram a adoção de redes de interconexões chaveadas para suportar a comunicação entre os componentes desses sistemas. Essas redes, quando implementadas no nível intrachip são denominadas Redes-em-Chip, do inglês NoCs – Networks-on-Chip (HEMANI et al., 2000). As NoCs provêm, dentre outras características, escalabilidade, reusabilidade e paralelismo de comunicação (ANDRIAHANTAINA et al., 2003; ZEFERINO, 2003; JERGER; PEH, 2009). Nesse contexto, Zeferino e Susin (2003) propuseram uma NoC que foi denominada SoCIN (System-on-Chip Interconnection Network), a qual caracteriza-se por ser baseada em roteadores parametrizáveis de baixo custo.

O espaço de projeto das NoCs é bastante amplo e vários aspectos arquiteturais precisam ser considerados para atender os requisitos de desempenho e custo. Para facilitar e acelerar a exploração desse espaço de projeto são necessários ambientes e ferramentas especializadas, conforme relatam Benini e De Micheli (2006). Esses autores também afirmam que ferramentas de análise são importantes para avaliar o desempenho de uma rede e buscar um melhor *tradeoff*¹ por meio do refinamento de parâmetros.

Para facilitar a exploração do espaço de projeto da rede SoCIN, Zeferino, Bruch e Pizzoni (2009) desenvolveram um ambiente de avaliação de desempenho da SoCIN denominado BrownPepper. O BrownPepper é um ambiente integrado que permite a avaliação de desempenho da rede SoCIN para diferentes configurações. Esse ambiente é baseado em modelos SystemC dos roteadores da rede e de geradores e medidores de tráfego. O BrownPepper também integra um conjunto de ferramentas desenvolvidas para automatizar as tarefas relacionadas à configuração, execução automática e análise de diferentes cenários de tráfego, com o mínimo de intervenção do usuário. Essas funcionalidades são suportadas por

¹ Um *tradeoff* é uma relação de alternativas a ser exploradas por um critério em relação a outro. Por exemplo, melhorar o desempenho muitas vezes resulta em maior custo, ou seja, beneficiar um critério à custa de outro (VAHID, 2008).

uma interface gráfica baseada em GTK+ (GIMP² Toolkit) e por ferramentas de visualização de gráficos (GNU Plot) e de diagramas de formas de onda (GTK Wave).

1.1 PROBLEMATIZAÇÃO

1.1.1 Formulação do Problema

O BrownPepper foi desenvolvido sem planejamento prévio, com código-fonte usando uma abordagem estruturada (ou seja, não orientada a objetos), sem projeto de software e com documentação restrita aos artigos publicados e aos extensivos comentários incluídos no código-fonte. Tais aspectos dificultam a manutenção do software e a adição de novas funcionalidades, em particular por outros desenvolvedores.

Além disso, o ambiente só é suportado por plataformas Linux, o que restringe o seu uso, enquanto o SystemC, biblioteca C++ base do simulador do BrownPepper, é suportado em outras plataformas (Windows e OS X).

Com o uso da ferramenta, novas funcionalidades e documentação mostraram-se necessários para a evolução deste ambiente. Porém, foram encontradas dificuldades nas tentativas de evoluir o ambiente. Portanto a ferramenta ficou estagnada e as novas necessidades não foram supridas.

Considerando as limitações mencionadas e a demanda para uso do BrownPepper em ensino e pesquisa, identificou-se a necessidade de se reconstruir o ambiente sob um novo paradigma e com novas tecnologias. Este é, portanto, o problema alvo deste trabalho.

1.1.2 Solução Proposta

Para resolver o problema descrito acima, foi iniciado, por este autor, um processo de reengenharia de software do ambiente BrownPepper no ano de 2013. Esse processo teve como finalidade melhorar a organização do código-fonte, disponibilizar uma versão estável, criar artefatos de projeto para facilitar a compreensão dos usuários e programadores que venham realizar incrementos na ferramenta, e realizar a troca do *framework* gráfico para uma

² GIMP – GNU Image Manipulation Program.

tecnologia que facilitasse as alterações de leiaute quando necessário e fosse suportado em múltiplas plataformas.

O presente trabalho buscou concluir o processo de reengenharia de software para a criação da nova versão do ambiente BrownPepper que contemplasse todos os recursos da primeira versão, trouxessem melhorias funcionais ao ambiente e disponibilizasse uma documentação completa para usuários e desenvolvedores que venham realizar incrementos e dar continuidade no processo de atualização do ambiente de simulação criado. Por questões de projeto esta nova versão recebeu o nome de RedScarf.

No processo de reengenharia da ferramenta, algumas decisões foram tomadas para a criação do RedScarf, como: (i) adoção do paradigma de programação Orientado a Objetos (OO) para melhorar a organização do código-fonte e torná-lo reusável; (ii) uso de diagramas da UML (Unified Modeling Language) na modelagem e na documentação; (iii) adequação do sistema ao modelo de software OO em camadas MVC (Model-View-Controller); e (iv) a mudança do *framework* gráfico GTK+ para o Qt. Esse *framework* é multiplataforma (com suporte a Linux/X11, Mac OS X, Windows e sistemas operacionais para dispositivos móveis³), e totalmente orientado a objetos. Além disso, o Qt possui todos os recursos necessários para a reconstrução da interface existente, tornando, portanto a nova versão OO por inteira.

Durante esse processo de reengenharia, as inconsistências encontradas foram corrigidas e algumas melhorias incorporadas ao ambiente ao longo de 2013 e do primeiro semestre de 2014 (durante o desenvolvimento da primeira parte deste TTC – Trabalho Técnico-científico de Conclusão de Curso). Além destas, novas funcionalidades foram identificadas, sendo que algumas implementadas também na primeira etapa do TTC, incluindo:

- Execução *multi-thread* das simulações para reduzir o tempo de execução dos experimentos;
- Geração de *feedback* para o usuário acompanhar o estado da rede durante as simulações;

³ Android, BlackBerry, iOS e Windows CE.

- Parada da simulação com base no tempo decorrido (não apenas nos pacotes entregues);
- Refinamento automático dos experimentos no ponto de saturação;
- Execução da ferramenta em plataformas Windows, Linux e OS X;
- Eliminação da dependência do GNUPlot e do GTKWave na análise dos resultados para execução em diferentes plataformas;
- Armazenamento das configurações dos experimentos; e
- Armazenamento das simulações geradas para ser possível restaurar/visualizar os resultados posteriormente.

Além dessas, pretendeu-se incorporar algumas funcionalidades desenvolvidas ou em desenvolvimento no Laboratório de Sistemas Embarcados e Distribuídos (LEDS – Laboratory of Embedded and Distributed Systems), incluindo:

- Integração de modelos de núcleos de processamento para simulação de aplicações reais;
- Suporte à geração de fluxos de comunicação maliciosos que emulem ataques às propriedades de segurança da rede SoCIN;
- Estimativa do consumo de energia com base no chaveamento de elementos de memória (*buffers* FIFO e registradores) ou em modelos de analíticos;

As contribuições deste TTC incluem: documentação do projeto do ambiente e dos seus recursos, mudança do paradigma de programação estruturado para orientado a objetos (OO), mudança do *framework* gráfico para aumentar a flexibilidade das mudanças de leiaute, redução do tempo de realização de experimentos pela execução de múltiplas simulações simultâneas, acompanhamento em tempo de execução das simulações correntes e suporte a novas funcionalidades derivadas de outros trabalhos de pesquisa do LEDS.

1.2 OBJETIVOS

1.2.1 Objetivo Geral

Disponibilizar nova versão da ferramenta BrownPepper, que denominada RedScarf, com ferramentas e recursos adicionais para melhoria do seu desempenho na execução de

experimentos, melhorando a visualização dos dados para análise de resultados e incorporando trabalhos de pesquisas do LEDS relacionados a SoCIN.

1.2.2 Objetivos Específicos

- Disponibilizar documentação completa de projeto e uso da ferramenta;
- Adicionar novas funcionalidades à ferramenta;
- Acelerar a execução dos experimentos de simulação;
- Gerar recurso visual para ampliar a análise dos resultados; e
- Testar e validar o ambiente proposto.

1.3 METODOLOGIA

Para a realização deste trabalho e cumprimento dos objetivos propostos, as atividades foram divididas nas etapas de: (i) conclusão do processo de reengenharia em que o objetivo foi atingir a versão atual da ferramenta com o novo modelo de software OO e com Qt utilizando a IDE (Integrated Development Environment – ambiente de desenvolvimento integrado) QtCreator; (ii) estudos com o levantamento da bibliografia, leitura e confecção do documento, abordando os assuntos Redes-em-Chip, SoCIN, avaliação de desempenho em Rede-em-Chip, BrownPepper, Qt, SystemC e um levantamento de ferramentas similares; (iii) o projeto através da definição dos requisitos, diagramas UML com o auxílio de uma ferramenta CASE (Computer-Aided Software Engineering – ferramenta de auxílio na engenharia de software); (iv) o desenvolvimento e testes da aplicação; e (v) a documentação.

A documentação é em nível de projeto de software, manual do usuário e documentação do código-fonte, além do documento textual do TTC e o artigo. E para isso, como mencionado, foi utilizada uma ferramenta CASE, um editor de texto e todos os artefatos produzidos na primeira parte do TTC que serviu de base para a confecção dos documentos propostos.

O desenvolvimento contou com a implementação das funcionalidades projetadas. Para isto, foi necessário um ambiente de programação e a documentação do projeto de software. Como resultado foi entregue o produto proposto contemplando todas as funcionalidades descritas na fase de projeto, excetuando-se as funcionalidades desejáveis e o acompanhamento gráfico da simulação, devido à complexidade e ao tempo disponível para a

realização. Também foram feitos testes unitários a fim de verificar o correto funcionamento das características implementadas, bem como testes de integração para validar a ferramenta no todo e verificar o correto funcionamento do ambiente. Parte dos testes foi realizada baseando-se na primeira versão do ambiente, sendo que os resultados obtidos foram idênticos aos da versão predecessora. Os testes das novas funcionalidades foram realizados com acompanhamento do orientador deste trabalho. Não foi feito nenhum plano de testes específico, pois algumas funcionalidades do projeto ainda não eram totalmente dimensionadas.

1.4 ESTRUTURA DO TRABALHO

Este documento é organizado em quatro capítulos. O Capítulo 1 apresentou a motivação do trabalho e seus objetivos. O Capítulo 2 apresenta o referencial teórico do trabalho, incluindo conceitos sobre NoCs e avaliação de desempenho de NoCs, uma descrição das características da SoCIN e do ambiente BrownPepper, além do posicionamento deste com os trabalhos similares da literatura. No Capítulo 3 é apresentado o desenvolvimento das funcionalidades acrescidas à ferramenta com a apresentação dos seus requisitos funcionais, aspectos de implementação e resultados. O Capítulo 4 apresenta as conclusões com uma discussão sobre o processo de realização deste trabalho, resultados atingidos, estratégias adotadas durante a execução deste projeto e pretensões futuras. Em apêndice, são apresentados uma revisão bibliográfica mais aprofundada sobre NoCs e o projeto de software do ambiente desenvolvido.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os temas mais pertinentes à realização deste trabalho, que são: aspectos gerais das Redes-em-Chip; SoCIN, que é a Rede-em-Chip alvo deste trabalho; avaliação de desempenho em Redes-em-Chip, quais métricas e estimativas de desempenho nesta aplicação; o ambiente BrownPepper como referência para melhorias e potencial de aplicabilidade deste trabalho; recursos do ambiente de simulação, que são as bibliotecas e recursos necessários utilizados na criação do ambiente, como Qt e SystemC; e uma análise sobre trabalhos relacionados, com o posicionamento deste trabalho em relação aos demais.

2.1 REDES-EM-CHIP

Com os avanços nos processos de fabricação de CIs (circuitos integrados), veio à miniaturização dos componentes, que viabilizou a construção de sistemas computacionais completos em um único chip, integrando vários componentes reutilizáveis chamados de núcleos ou blocos de propriedade intelectual. Esses sistemas são denominados de sistemas integrados ou SoCs (Systems-on-Chip) (ZEFERINO, 2003).

Para realizar a integração dos núcleos, é necessário um mecanismo de comunicação que interconecte os blocos e atenda os requisitos da aplicação. Em 1992, Tewksbury, Uppuluri e Hornak já discutiam as limitações das arquiteturas de interconexão tradicionais, como o barramento, para suportar a interconexão dos elementos de processamento em um SoC. Em 1999, Guerrier e Greiner discutiram sobre os problemas de desempenho nas tecnologias baseadas em barramentos para sistemas em chip, como, o aumento da capacidade à medida que o fio do barramento aumenta. Zeferino (2003) também apontou que os requisitos de desempenho em comunicação, como largura de banda escalável e baixa latência, dificilmente serão atendidos pelas arquiteturas baseadas no barramento.

Esses são alguns aspectos que despertaram o interesse e a necessidade de uma alternativa de arquitetura de comunicação para sistemas integrados. Nesse contexto, a solução proposta pela comunidade científica está no uso de redes de interconexão chaveada, como as encontradas em computadores paralelos. Essas redes chaveadas, quando aplicadas à comunicação intrachip, recebem múltiplas denominações na literatura (ZEFERINO, 2003; JERGER; PEH, 2009), Micronetworks, On-Chip Networks (OCNs), On-Chip Interconnection

Network (OCIN) e Networks-on-Chip (NoC) Esta última, proposta por Hemani et al. (2000), tem sido a denominação de maior aceitação nas comunidades científica e industrial.

As Redes-em-Chip ou NoCs surgiram da necessidade de uma arquitetura de comunicação para sistemas integrados, que provesse, dentre outras características, escalabilidade, reusabilidade e paralelismo de comunicação (ANDRIAHANTENAINA et al., 2003; ZEFERINO, 2003; JERGER; PEH, 2009).

Nas redes de interconexão, os projetistas estão condicionados a buscar alternativas de topologia, roteamento e controle de fluxo da rede para atender às especificações das aplicações para quais são projetadas. Características estas que estão diretamente relacionadas ao desempenho da rede. (DALLY; TOWLES, 2004).

Assim como em um projeto de rede de interconexão, as Redes-em-Chip também são caracterizadas pela topologia, pelos mecanismos de roteamento e controle de fluxo, e pela microarquitetura dos roteadores (JERGER; PEH, 2009). Uma breve descrição de cada uma destas características é feita por Jerger e Peh (2009):

- Topologia: Uma Rede-em-Chip é composta de canais e nodos de chaveamento (roteadores). A topologia da rede determina o leiaute físico e a conexão entre os nodos e canais na rede;
- Roteamento: Dada uma topologia, o algoritmo de roteamento determina o caminho através da rede que a mensagem deve realizar para chegar ao destino. Um algoritmo de roteamento que é capaz de balancear o tráfego (ou carga) tem impacto direto na vazão e desempenho da rede;
- Controle de fluxo: O controle de fluxo determina como os recursos são alocados para a transmissão das mensagens através da rede. O controle de fluxo é o mecanismo responsável por alocar (e desalocar) *buffers* e a largura de banda dos canais para os pacotes (partes das mensagens) em espera. Os recursos podem ser alocados para os pacotes no todo (feito nos controles de fluxo do tipo *store-and-forward* e *virtual cut-through*), o que requer *buffers* muito grandes tornando essa abordagem impraticável para comunicação intrachip. Por isso, nas NoCs, o mais comum é o uso de controle de fluxo no nível de *flit* (*flow control unit* – unidade de controle de fluxo, uma subdivisão do pacote). Nesta abordagem, os *buffers* e a largura de banda do canal são alocados em uma menor granularidade (*flits*) do que

no controle de fluxo em nível de pacotes. Como resultado, os roteadores podem ser projetados com *buffers* menores;

- **Microarquitetura do roteador:** A microarquitetura genérica do roteador é composta dos seguintes componentes: *buffers* de entrada, estado do roteador, lógica de roteamento, alocadores e o *crossbar* (ou *switch* – comutador). As funcionalidades do roteador são executadas, frequentemente, em *pipeline*⁴ para melhorar a vazão. O atraso em cada roteador da Rede-em-Chip é o contribuinte primário para a latência de comunicação. Como resultado, pesquisadores têm dedicado um esforço significativo para diminuir o tempo despendido nos estágios de *pipeline* e aumentar a vazão.

O detalhamento destas propriedades pode ser visto no Apêndice A deste trabalho.

2.2 A REDE SOCIN

Esta seção baseia-se na tese realizada por Zeferino (2003) para introduzir alguns conceitos sobre a SoCIN e no artigo apresentado por Zeferino, Santo e Susin (2004), para detalhar as características do roteador utilizado na versão atualizada da rede.

A SoCIN (System-on-Chip, Interconnection) utiliza topologia direta, podendo ser configurada como uma malha ou toróide. Possuía em sua primeira versão controle de fluxo do tipo *handshake*, roteamento do tipo fonte e determinístico, chaveamento por pacotes do tipo *wormhole*, arbitragem dinâmica distribuída e memorização de entrada.

A primeira versão do roteador da rede SoCIN foi o RASoC (Router Architecture for Systems-on-Chip), que é um *soft-core*⁵ parametrizável descrito em VHDL (VHSIC⁶ Hardware Description Language). Os parâmetros do RASoC incluem a largura do canal de dados e a profundidade dos *buffers* de entrada. O RASoC utiliza roteamento fonte, em uma implementação em que o cabeçalho do pacote é criado com as quantidades de roteadores a

⁴ Em português, canalização, mas por questões técnicas da área optou-se pela utilização do termo original. Tendo uma analogia com paralelização (PATTERSON; HENNESSY, 2014).

⁵ Um *soft-core* é um modelo de hardware descrito numa HDL (Hardware Description Language – linguagem de descrição de hardware) que pode ser customizado para uma dada aplicação e sintetizado em ASIC ou FPGA (tecnologias de fabricação de CIs) (TONG; ANDERSON; KHALID, 2006).

⁶ Very High Speed Integrated Circuit (Circuito integrado de velocidade muito alta).

serem percorridos nas direções X e Y. Ou seja, o nodo origem executa o algoritmo de roteamento e define a rota a ser utilizada.

O segundo roteador utilizado na SoCIN foi o ParIS (Parameterizable Interconnect Switch), que estendeu a capacidade de parametrização do RASoC, incluindo alternativas para os mecanismos de roteamento, arbitragem, controle de fluxo, chaveamento e memorização, ampliando o escopo para exploração do espaço de projeto. Para suportar diferentes algoritmos de roteamento, o local da realização do roteamento foi modificado do nodo origem para os roteadores. Ao invés de incluir o caminho até o destinatário, o cabeçalho passou a transportar o endereço do destinatário apenas, com base em suas coordenadas X e Y. Ou seja, o roteamento deixou de ser local e passou a ser distribuído.

Com a evolução do roteador e da rede, a SoCIN possui dentre outras características:

- Roteamento: Distribuído (determinístico ou parcialmente adaptativo);
- Arbitragem: Round-Robin, estática e randômica;
- Controle de fluxo: *Handshake* ou baseado em créditos;
- Chaveamento: Wormhole, transpasse virtual e por circuitos;
- Memorização: nas entradas, podendo ter nas saídas com *buffers* FIFO parametrizáveis; e
- Informações de roteamento no cabeçalho: coordenadas do roteador de destino na rede.

Assim como o RASoC, o ParIS também é um *soft-core* VHDL baseado em uma biblioteca de blocos parametrizáveis, incluindo diferentes abordagens/implementações para cada técnica usada para encaminhar as mensagens. O roteador possui cinco portas de comunicação, sendo elas: L (*local* – porta local), N (*north* – porta norte), S (*south* – porta sul), E (*east* – porta leste) e W (*west* – porta oeste). A porta *Local* é reservada para a conexão com um núcleo ou um subsistema para a rede, enquanto as demais portas (nomeadas com os pontos cardinais) são para a conexão com os demais roteadores na rede. Dependendo da posição do roteador na rede, nem todas as portas são necessárias, como os roteadores de borda em uma malha 2D.

Os pacotes são transferidos na SoCIN utilizando a abordagem sem perda de informação, ou seja, os pacotes não são descartados em nenhuma hipótese e, os algoritmos de roteamento são limitados aos usados para topologias de duas dimensões ortogonais, como a malha. Para isso, foi estipulado um sistema de coordenadas em que cada roteador na rede é identificado por um par de coordenadas cartesianas (X_{id}, Y_{id}), mostrado na Figura 1, sendo o eixo horizontal representado por “X” e o vertical por “Y”. A posição em X e Y determina o endereço do roteador na rede, informação que é utilizada na transmissão dos pacotes.

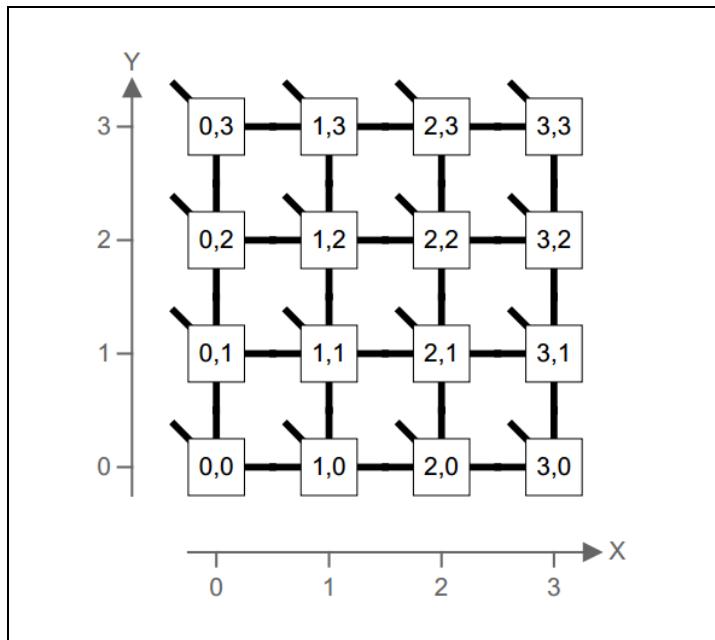


Figura 1. Sistema de coordenadas da SoCIN

Fonte: Zeferino, Santo e Susin (2004).

Quando um núcleo envia uma mensagem pela rede, esta mensagem deve ser dividida em pacotes, em cada pacote deve conter o sistema de coordenadas do destino no cabeçalho do pacote (X_{dest}, Y_{dest}). Essa informação é utilizada pelos roteadores para determinar para qual porta de saída do roteador o pacote deve ser encaminhado. Nas abordagens de roteamento total ou parcialmente adaptativo, o roteador também deve considerar as informações sobre a disponibilidade dos canais de saída para realizar o roteamento.

Na Figura 2, é apresentada a organização do roteador ParIS. Sendo que é dividida em módulos de entrada e saída (*in* e *out*) e de acordo com a porta de comunicação (L, N, E, W, S). Os módulos apresentados na Figura 2 são os referentes às portas *Local* e *West*, nota-se que as estruturas desses módulos são compostas pelos mesmos blocos, sendo que a única diferença é à qual porta se refere.

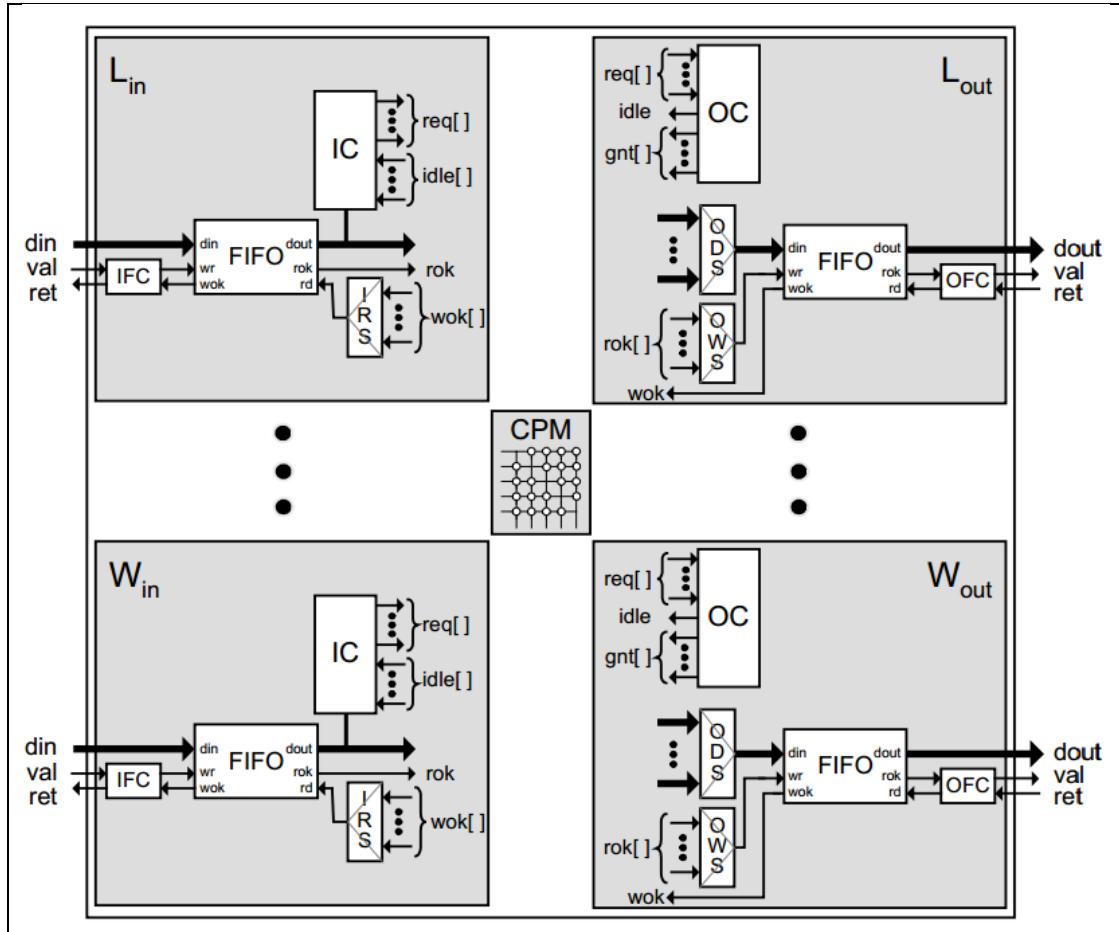


Figura 2. Organização do ParIS

Fonte: Zeferino, Santo e Susin (2004).

Os módulos de entrada são compostos pelos blocos: IFC, FIFO, IC e IRS. Já os módulos de saída incluem os blocos: OC, ODS, OWS, FIFO (opcional) e OFC. Os blocos IFC (Input Flow Control) e OFC (Output Flow Control) são os responsáveis pelo controle de fluxo no roteador. Os blocos de FIFO são os mesmos nos módulos de entrada e saída, sendo responsáveis pela memorização dos pacotes. Os blocos IC (Input Controller) e OC (Output Controller) são responsáveis pelo roteamento e pela arbitragem, respectivamente. Os blocos ODS (Output Data Switch), OWS (Output Write Switch) e IRS (Input Read Switch) são multiplexadores que compõem o *crossbar* do roteador e realizam a comutação entre os sinais de dado e de controle dos módulos de entrada e de saída. Por fim o CPM (Crosspoint Matrix) é um bloco de configuração responsável por definir as conexões permitidas entre os módulos de entrada e de saída do roteador de acordo com o algoritmo de roteamento utilizado. Destaca-se que os diferentes algoritmos de roteamento estabelecem diferentes restrições de conexão entre as entradas e as saídas do roteador. Por exemplo, no roteamento XY, os módulos de entrada N_{in} e S_{in} não podem ser conectados aos módulos de saída E_{out} e W_{out} .

No roteador ParIS, o tempo necessário para o cabeçalho do pacote atravessar o roteador, ou seja, o tempo de atraso do roteador, é de 2 ou 3 ciclos, sendo que, é gasto um ciclo para roteamento, um ciclo de arbitragem e um ciclo no *buffer* de saída (se ele for implementado). O controle de fluxo impacta no desempenho dos enlaces, portanto fora do roteador, no baseado em créditos é necessário um ciclo para a transferência de um *flit* e no *handshake* devido aos sinais de controle são necessários quatro ciclos de relógio. Na transferência do cabeçalho, o *flit* é escrito no *buffer* de entrada na terceira fase do *handshake* e a quarta fase (*reset* do sinal de reconhecimento – *ret*) ocorre em paralelo com o ciclo de roteamento.

Do exposto acima, considerando-se o envio de um pacote com N flits (incluindo o cabeçalho) entre dois nodos separados por D enlaces (incluindo os enlaces entre os núcleos e os roteadores) e, portanto, por $D - 1$ roteadores (sem *buffers* de saída), a latência para envio do pacote, na ausência de contenção é dada por (1), ou seja, o tempo gasto para o *flit* de cabeçalho atravessar os D enlaces e os $D - 1$ roteadores mais o tempo para o restante do pacote ($N - 1$) chegarem ao destinatário após o cabeçalho. Considerando o número de ciclos gastos na transferência de um *flit* em cada técnica, quando é utilizado o controle de fluxo baseado em créditos, a latência é dada por (2). Já no controle de fluxo *handshake* a latência é dada por (3). O tempo de roteamento e arbitragem (*tra*) são menores no *handshake* por causa da sobreposição mencionada previamente, dado que a fase de roteamento (primeiro dos dois ciclos) ocorre em paralelo ao último ciclo do *handshake*. Dessa forma:

$$t = t_{Link} * D + tra * (D - 1) + t_{Link} * (N - 1) \quad (1)$$

Como:

$$\begin{aligned} t_{Link}_{credit} &= 1 \\ tra_{credit} &= 2 \\ t_{Link}_{handshake} &= 4 \\ tra_{handshake} &= 1 \end{aligned}$$

Então:

$$t_{credit} = 1 * D + 2 * (D - 1) + 1 * (N - 1) \quad (2)$$

$$t_{handshake} = 4 * D + 1 * (D - 1) + 4 * (N - 1) \quad (3)$$

Desenvolvendo essas equações, obtém-se (4) e (5):

$$t_{credit} = 3 * D + N - 3 \quad (4)$$

$$t_{handshake} = 5 * D + 4 * N - 5 \quad (5)$$

Essas equações, entretanto, não permitem modelar o efeito da concorrência entre pacotes e estimar a latência dos pacotes quando múltiplos fluxos competem pelos recursos da rede. A modelagem matemática desse efeito é alvo de outra pesquisa em andamento no LEDS e está fora do escopo deste trabalho, o qual tem como foco o uso de simulador para a obtenção de métricas de desempenho da rede, conforme segue.

2.3 AVALIAÇÃO DE DESEMPENHO EM REDES-EM-CHIP

A avaliação de desempenho está relacionada com a mensuração e a verificação da aplicabilidade de uma solução para determinado sistema. No projeto das Redes-em-Chip os requisitos de desempenho norteiam as definições e a caracterização da rede e suas particularidades. O desempenho de uma NoC é medido através da observação do tempo e quantidade de informações que a rede suporta, isto é feito através da análise do tráfego de pacotes na comunicação entre os nodos. As principais métricas utilizadas na avaliação do desempenho de uma rede são a vazão (também referenciada como tráfego aceito) e a latência (DALLY; TOWLES, 2004; OGRAS; MARCULESCU, 2013; DUATO; YALAMANCHILI; NI, 2003; JERGER; PEH, 2009).

A avaliação de desempenho nas Redes-em-Chip tradicionalmente é realizada por meio da simulação, que, apesar de ser extremamente lenta, permite determinar o impacto das diferentes configurações dos parâmetros de projeto no desempenho da rede (OGRAS; MARCULESCU, 2013). Por extremamente lenta, entende-se que o tempo de obtenção das métricas de desempenho na simulação é muito maior do que comparado à aplicação de modelos analíticos, os quais, no entanto, apresentam apenas uma estimativa do desempenho, com algum erro de aproximação.

Para realizar a análise do tráfego são utilizados padrões de distribuição de carga pela rede, como a distribuição uniforme, na qual cada terminal da rede envia pacotes para todos os demais terminais da rede com a mesma probabilidade. (DUATO; YALAMANCHILI; NI, 2003).

Os geradores de tráfego (padrões de distribuição de tráfego sintético) podem ser implementados como terminais de instrumentação, que são acoplados na porta dos nodos terminais da rede. Esses terminais são compostos dos seguintes módulos: fonte de pacotes, temporizador e contador de entrada, fila e, temporizador e contador de saída (Figura 3) (DALLY; TOWLES, 2004).

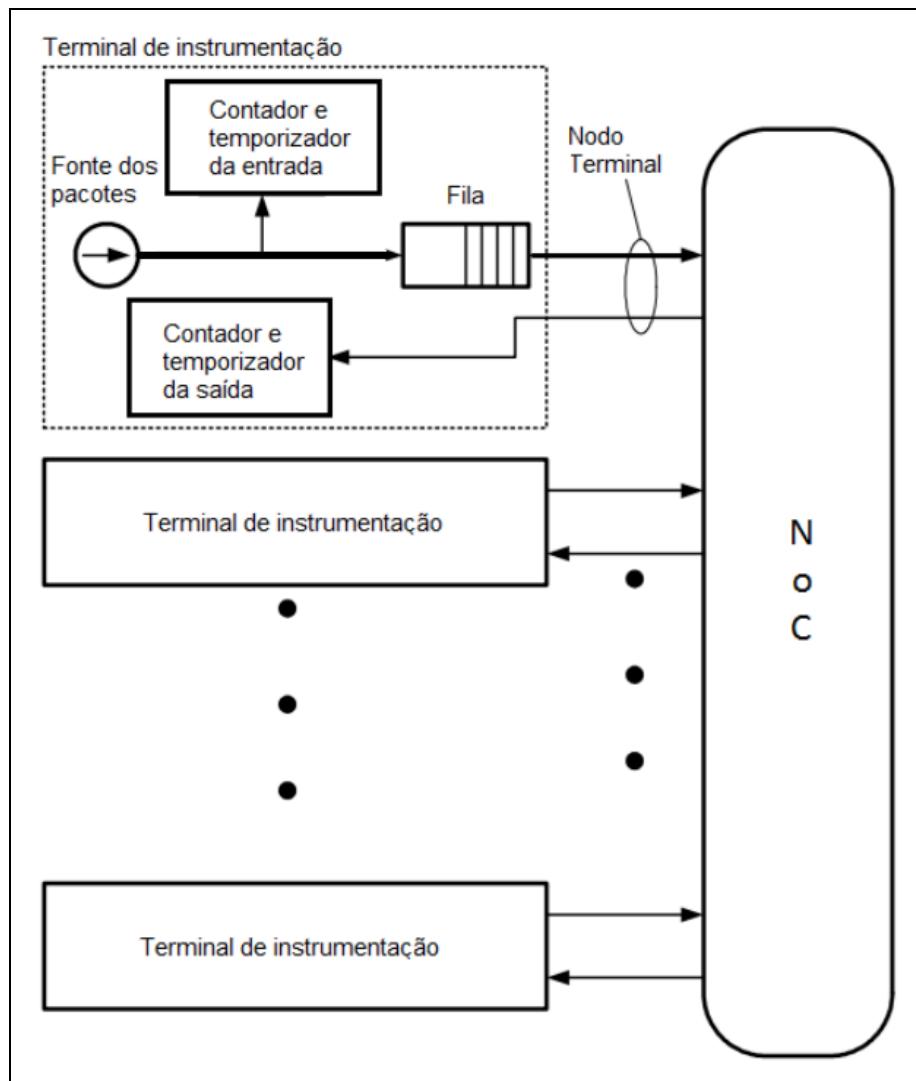


Figura 3. Terminais de instrumentação acoplados à rede

Fonte: Adaptado de Dally e Towles (2004).

Existem alternativas de organização do terminal de instrumentação, além da apresentada na Figura 3, sendo que, na arquitetura ilustrada, a medição das informações somente é realizada fora da rede. No entanto, ela pode ser estendida para uma análise mais detalhada com a inclusão de terminais de instrumentação dentro da rede, possivelmente nos enlaces, e com uma organização diferente.

A fonte dos pacotes é responsável pela geração e definição das informações de tráfego dos pacotes, de acordo com um padrão de distribuição definido para o gerador sintético de tráfego. O módulo que é contador e temporizador de entrada é responsável por contabilizar a quantia de pacotes sendo injetada na rede e o instante (ciclo) em que cada pacote é injetado. A fila de entrada é necessária para que não haja descarte de nenhum pacote gerado, isso devido à possibilidade de congestão na rede e, portanto, não sendo possível a injeção de pacotes. O módulo que é contador e temporizador de saída é responsável por contar a quantia de pacotes recebidos pelo terminal assim como salvar o momento da recepção.

Os módulos de contagem e temporização são as estruturas responsáveis pela coleta dos dados que serão utilizados na avaliação do desempenho. Por exemplo, a latência de um pacote é calculada pela diferença entre o tempo que um pacote é completamente recebido pelo terminal de instrumentação do destinatário e o tempo em que o módulo da fonte escreve o pacote na fila do terminal de instrumentação origem (momento da criação do pacote).

A vazão é dada pela quantidade máxima de informação entregue por unidade de tempo. Em outras palavras, é a taxa em que os pacotes são entregues aos nodos destinos. Com pouca carga de informação na rede, a taxa de entrega dos pacotes é igual à taxa de injeção, pois pacotes não enfrentam contenção e o tráfego aceito é igual ao tráfego oferecido. Contudo à medida que a carga de trabalho oferecida à rede aumenta e a contenção começa aparecer nos roteadores, a taxa de entrega fica constante ou cai, pois a rede não consegue entregar os pacotes na mesma taxa em que os pacotes são inseridos. Assim, é possível fazer uma relação do tráfego aceito (vazão) pelo tráfego oferecido (demanda) (OGRAS; MARCULESCU, 2013; DALLY; TOWLES, 2004), conforme é ilustrado na Figura 4.

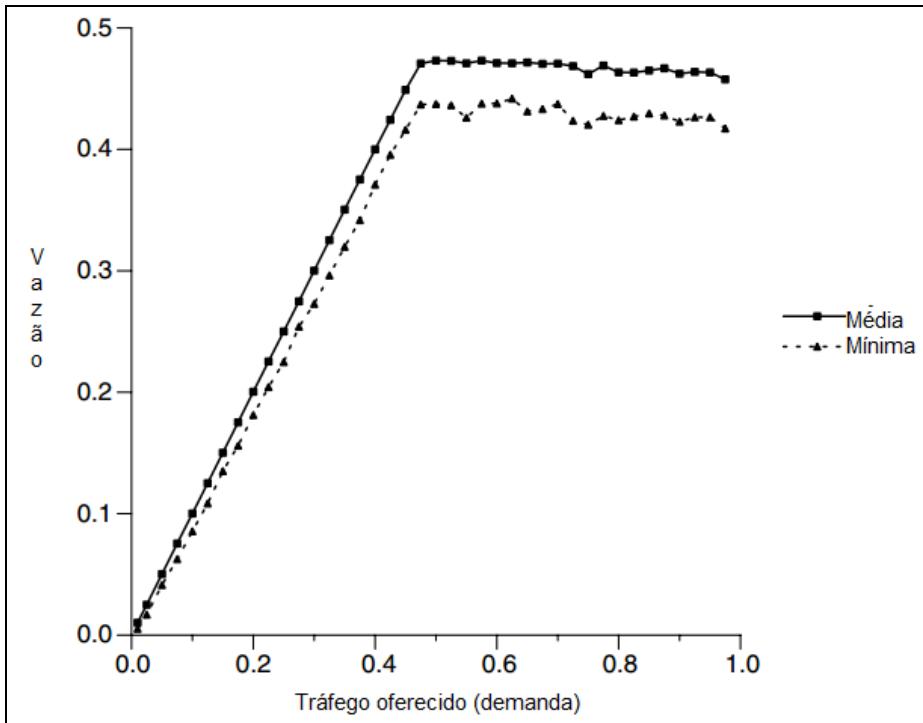


Figura 4. Relação da vazão pelo tráfego oferecido, obtida de uma malha 8x8 com tráfego complemento

Fonte: Adaptado de Dally e Towles (2004).

Nota-se, na Figura 4, que para a rede utilizada, o tráfego aceito (vazão) até o ponto de saturação é o mesmo que o tráfego oferecido. O ponto de saturação é o momento em que a taxa de entrega fica quase que constante (em uma rede estável) ou decai (em uma rede instável). Após a saturação, que ocorreu com cerca de 40% da capacidade de transferência, o aumento do tráfego oferecido (demanda de comunicação) continua, mas a rede não suporta entregar as informações na mesma taxa que recebe. Isso provavelmente ocorre devido à alta carga de informações e à contenção nos roteadores. Portanto, para dar uma noção mais intuitiva do desempenho da rede, a vazão pode ser vista como uma fração da capacidade que a rede suporta (DALLY; TOWLES, 2004).

Para a avaliação de desempenho por meio da simulação, é importante considerar a execução dos experimentos em três fases: aquecimento, medição e drenagem (esfriamento). A fase de aquecimento é o início da simulação, quando os primeiros pacotes são injetados na rede e não enfrentam nenhuma congestão, pois a rede está vazia. A fase de medição é o estado em que a rede atingiu uma estabilidade de comunicação após várias transmissões já efetuadas, e é o momento que de fato os pacotes devem ser medidos. E a fase de drenagem é o final da simulação, quando não são gerados pacotes e o aguardo é pela entrega de todos os pacotes ao seu destino. A fase de medição, como já é referida, é quando os pacotes devem ser medidos

de fato, pois é o momento que mais se aproxima da aplicação real (DALLY; TOWLES, 2004).

2.4 O AMBIENTE BROWNPEPPER

Um exemplo de ambiente para avaliação de desempenho de NoC baseado em simulação é o BrownPepper (ZEFERINO; BRUCH; PIZZONI, 2009), foco deste trabalho. O BrownPepper é um ambiente integrado que permite a avaliação de desempenho da rede SoCIN para diferentes configurações e baseia-se em modelos de simulação descritos em SystemC.

O SystemC é uma biblioteca de classes C++ padronizada pelo ANSI⁷ para projeto e implementação de hardware e software, indicado para projetistas e arquitetos que necessitam caracterizar sistemas complexos que utilizam abordagem híbrida entre hardware e software. O SystemC permite ao usuário escrever um conjunto de funções (processos) C++ que são executadas com o controle de um escalonador na intenção de imitar o passar do tempo simulado, sincronizando e comunicando de forma eficiente, de modo a reproduzir sistemas eletrônicos contendo hardware e software embarcado (IEEE COMPUTER SOCIETY, 2012). O núcleo do SystemC é um motor de simulação que contém o escalonador de processos. Os processos executados respondem por meio da notificação de eventos, os quais são pontos específicos no tempo simulado. O SystemC comporta uma série de módulos para a construção de hardware, como os módulos de: portas, canais, processos, eventos e outros (IEEE COMPUTER SOCIETY, 2012). Atualmente (2014), o SystemC encontra-se na versão 2.3.1 (ACCELLERA, 2014).

No BrownPepper, o roteador ParIS é descrito em SystemC no nível de transferência entre registradores (RTL – Register-transfer Level), em uma tradução 1-para-1 do seu modelo VHDL sintetizável. Já os terminais de instrumentação (geradores e medidores de tráfego) são descritos em uma combinação do RTL com o nível de transação (TL – Transaction Level), com o qual são utilizadas operações e funções mais abstratas.

Além dos modelos de simulação, o BrownPepper integra um conjunto de ferramentas para automatizar as tarefas relacionadas à modelagem dos cenários de tráfego, configuração

⁷ American National Standards Institute (instituto nacional americano de padronização).

de experimentos e análise dos resultados. O uso dessas ferramentas é facilitado por uma interface gráfica baseada em GTK+.

O BrownPepper trabalha com uma plataforma SoC composta de: rede SoCIN, geradores de tráfego (TGs – Traffic Generators) e medidores de tráfego (TMs – Traffic Meters) acoplados aos terminais da rede. Cada TG gera fluxos de comunicação (no nodo fonte) compostos de sequências de pacotes a serem enviados pela rede para um dado destinatário (outro TG retira os pacotes da rede). Cada TM é responsável por monitorar o tráfego no terminal da rede e coletar os dados dos pacotes que chegam aos TGs de destino.

Cada fluxo de comunicação gerado pelos TGs é caracterizado por um conjunto de parâmetros, sendo que os principais são: *(i)* largura de banda requerida; *(ii)* número de pacotes a serem enviados; *(iii)* endereço do destinatário ou uma distribuição espacial usada para definir um ou mais destinatários; e *(iv)* o tipo de taxa de injeção dos pacotes, que pode ser constante ou variável (também existe a possibilidade de injeção de rajadas).

O fluxo de projeto do BrownPepper é composto de um conjunto de ferramentas usadas para: *(i)* geração do simulador baseado em SystemC; *(ii)* geração o modelo de tráfego; *(iii)* execução das simulações; e *(iv)* análise dos resultados. A Figura 5 exibe essas ferramentas.

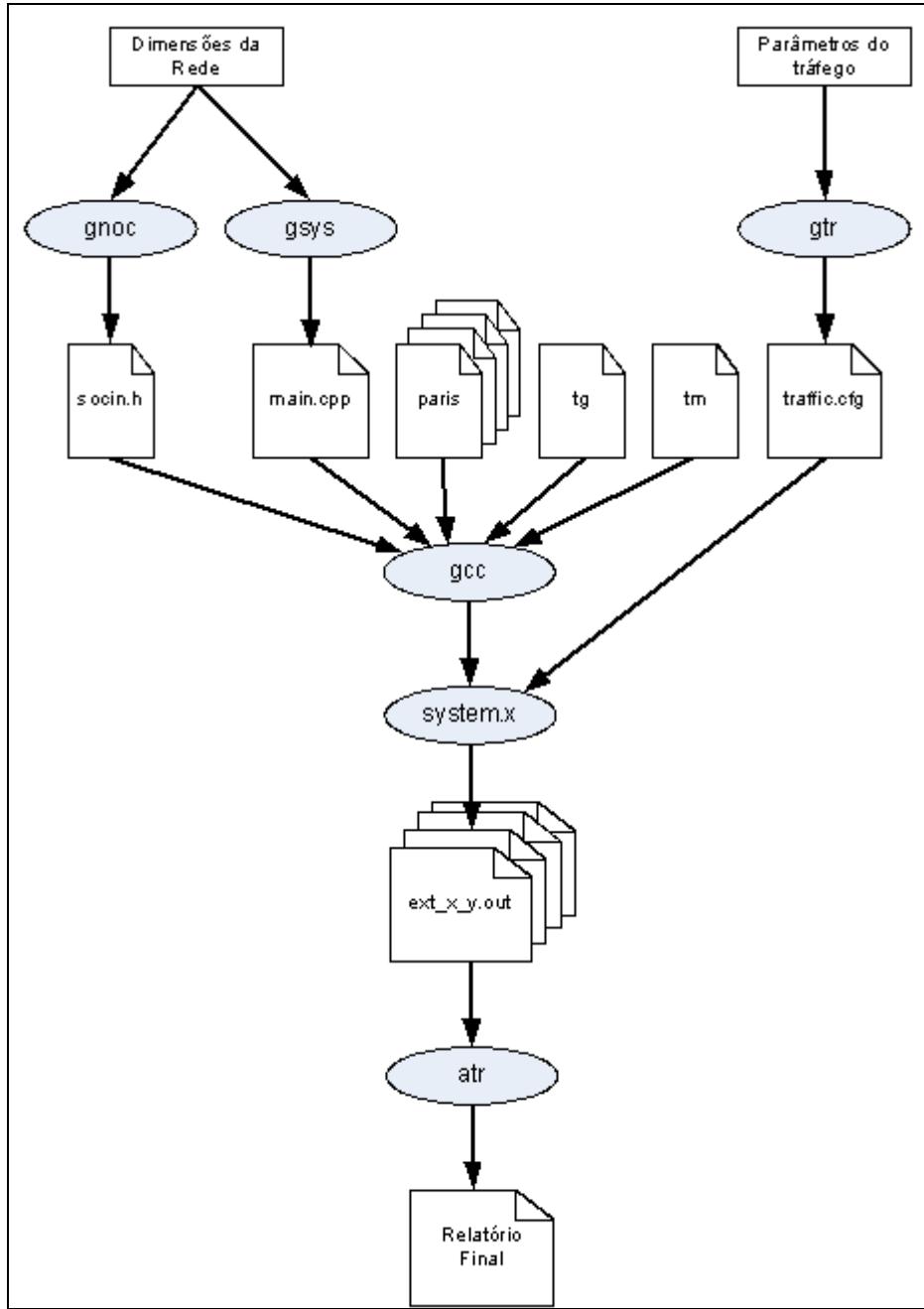


Figura 5. Ferramentas e fluxo utilizado no ambiente BrownPepper

Fonte: Adaptado de Zeferino et al. (2007).

As ferramentas relacionadas à geração do simulador são o *gnoc* e o *gsys*, que geram os arquivos do modelo da rede e do sistema, respectivamente. A ferramenta de geração do arquivo com o modelo de tráfego é o *gtr*. Já a ferramenta *atr* é responsável pela geração de relatórios a partir dos *logs* produzidos pelos TMs durante a execução do simulador.

A interface gráfica do BrownPepper é organizada em três abas; (*i*) Configuração do sistema; (*ii*) Simulação do sistema; e (*iii*) Análise de desempenho. Na primeira, são definidos alguns parâmetros do sistema e todos os parâmetros de tráfego. Na segunda, são definidos os

parâmetros dos experimentos a serem realizados. Já a terceira aba oferece opções para visualização dos resultados da avaliação de desempenho. As figuras a seguir apresentam essas abas da interface gráfica, as quais foram construídas com o uso da ferramenta Glade, um construtor gráfico de interfaces para GTK+.

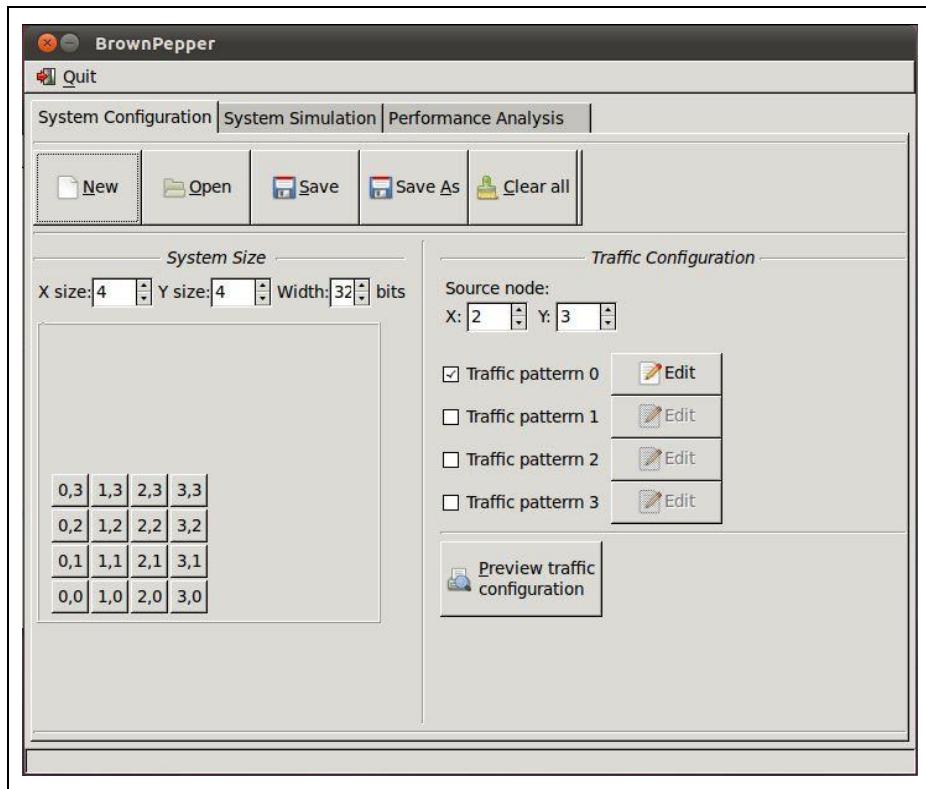


Figura 6. BrownPepper: aba de configuração do sistema

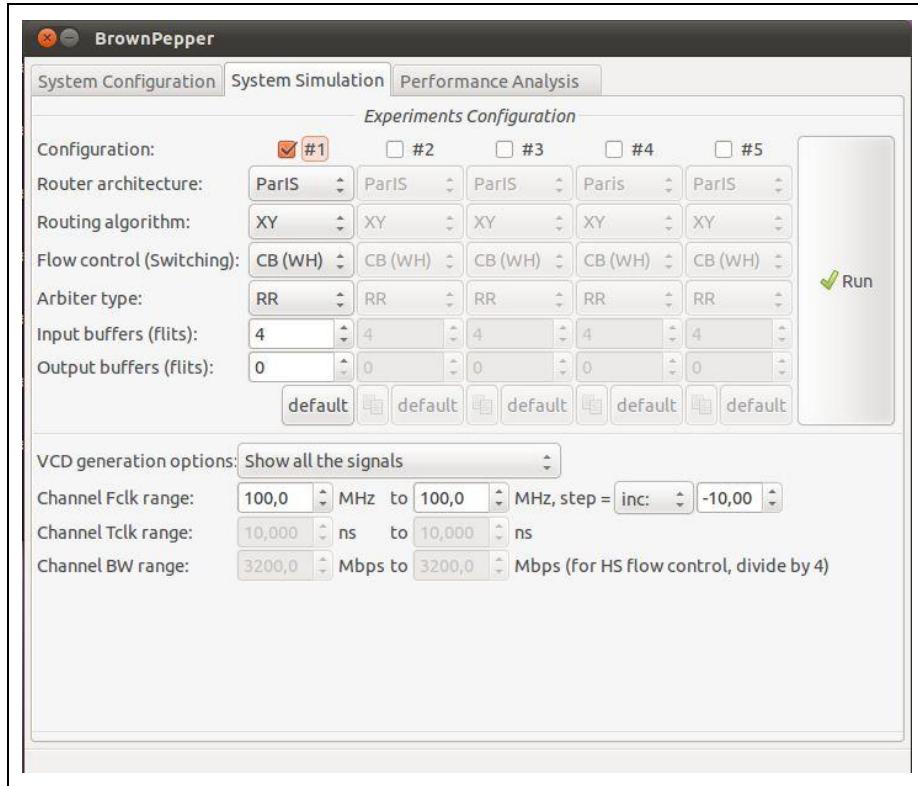


Figura 7. BrownPepper: aba de configuração dos experimentos de simulação

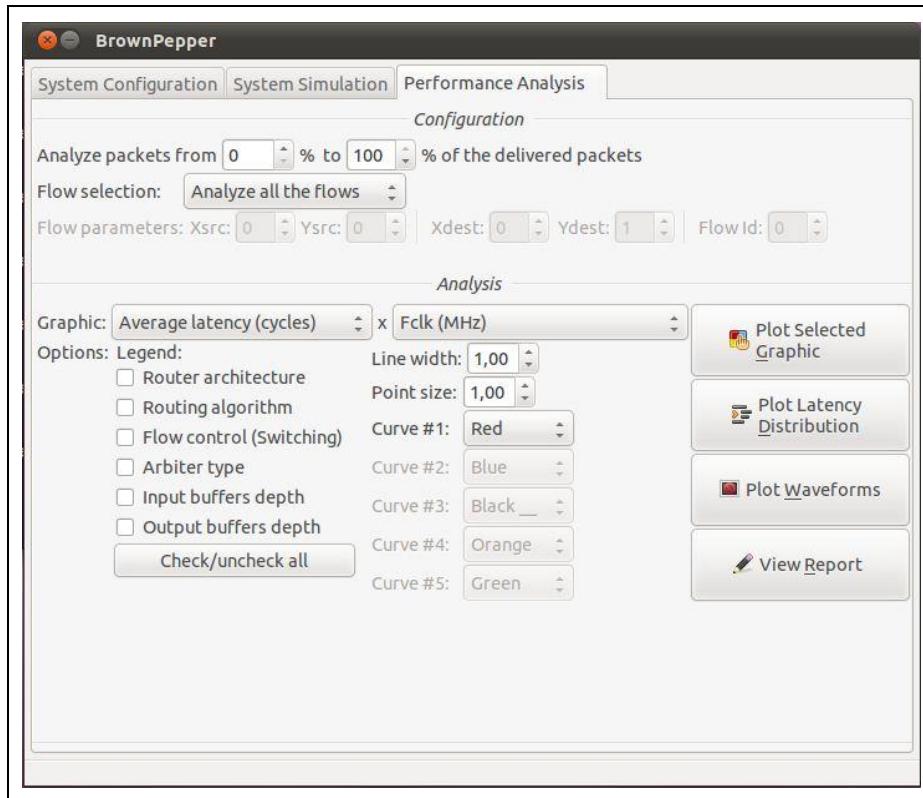


Figura 8. BrownPepper: aba de visualização dos resultados da análise de desempenho

Na aba de configuração do sistema (Figura 6), é possível definir a largura dos canais e o tamanho do sistema (quantidade *tiles* ou lajotas). Cada *tile* é composto de um roteador ParIS e de um par de terminais de instrumentação (TG-TM) e é possível definir até quatro padrões de tráfego a serem gerados por cada TG. Nessa aba, também é possível carregar uma configuração, salvar a configuração atual e limpar as configurações realizadas.

Na aba de configuração dos experimentos de simulação (Figura 7), são determinados os experimentos a serem realizados, com a opção de definir até cinco ajustes diferentes de roteador, podendo variar os seguintes parâmetros: arquitetura do roteador, algoritmo de roteamento, técnica de controle de fluxo, tipo de árbitro e profundidade dos *buffers* de entrada e de saída. Outro parâmetro a ser configurado é a frequência de operação, sendo que pode ser definido um intervalo de frequências e o passo de variação da frequência nesse intervalo. Após a configuração dos experimentos, o usuário aciona o botão “Run” e todos os experimentos são realizados automaticamente, sem intervenção do usuário.

Na aba de visualização dos resultados da análise de desempenho (Figura 8), são determinados os pontos de análise dos pacotes entregues. É nesta parte que são definidos os períodos de aquecimento e de drenagem, podendo ser desconsiderada a análise de um percentual inicial dos pacotes referentes ao aquecimento e um percentual final referente à drenagem. Também podem ser selecionados os fluxos específicos para serem analisados. A visualização dos resultados pode ser feita por meio de uma tabela, de diagramas de formas de onda (com o uso da ferramenta GTK Wave) e de gráficos gerados pela ferramenta GNU Plot. Podem ser selecionados diferentes tipos de gráficos para visualização (ex: Latência Média x Tráfego Oferecido, Latência Média x Frequência de Operação, Tráfego Aceito x Tráfego Oferecido, entre outros).

2.5 O FRAMEWORK QT

Conforme mencionado no Capítulo 1, para o RedScarf foi proposta a substituição do GTK+ pelo Qt (“cute” Q toolkit – kit de ferramentas Q, pronunciado como “quiute”), um framework C++ para desenvolvimento de aplicações multiplataformas com GUI (Graphical User Interface). O Qt utiliza a abordagem “escreva uma vez, compile em qualquer lugar” e permite aos programadores usar uma única estrutura de código-fonte para executar aplicações no Windows, Mac OS X, Linux, Solaris e muitas outras versões de Unix com X11 (BLANCHETTE; SUMMERFIELD, 2006).

Mas o Qt não é apenas para construção de interface gráfica, pois também possui uma API (Application Programming Interface) de funções para facilitar ao programador C++ desenvolver funcionalidades de baixo nível do sistema, como manipulação de arquivos, utilização de threads, execução de processos e outros (QT PROJECT, 2014).

O Qt é distribuído por meio de licenças como GPL para a criação de aplicações de código-fonte aberto. Para aplicações comerciais, é necessária a aquisição da licença comercial. O KDE (K Desktop Environment) é um exemplo de aplicação que utiliza o Qt (BLANCHETTE; SUMMERFIELD, 2006).

Atualmente a versão Open Source de desenvolvimento é mantida pela comunidade de desenvolvedores (QT PROJECT, 2014). E a versão comercial é suportada pela Digia (QT DIGIA, 2014). O Qt encontra-se na versão 5.2 (QT PROJECT, 2014; QT DIGIA, 2014). E foi adotado por ser totalmente orientado a objetos e por apresentar facilidade na alteração de leiaute gráfico.

O *framework* Qt foi escolhido para a criação da nova versão do sistema, por possuir ferramentas de criação de interfaces que são versáteis e facilitam as alterações de leiaute das aplicações gráficas. É multiplataforma e abstrai as APIs dos sistemas operacionais para facilitar a programação em C++ sem ter reescrita e adequação de código fonte para cada plataforma. Possui documentação completa, totalmente orientado a objetos, estável e além da comunidade de desenvolvedores ser ampla, pois o *framework* foi criado em 1991, vários softwares populares são desenvolvidos utilizando esta tecnologia. A seguir alguns exemplos⁸ de softwares que utilizam o Qt.

- Altera Quartus;
- Autodesk Maya;
- Adobe Photoshop;
- Google Earth;
- KDE;
- Need For Speed (últimas versões)

⁸ De conhecimento do autor.

- Skype;
- VirtualBox; e
- Vlc Media Player.

Alguns casos de soluções e companhias que utilizam o Qt são listados pela Digia (2014), atual mantedora do Qt na versão comercial, são elas:

- Barco;
- DAZ 3D;
- High End Systems;
- Navico Lowrence HDS marine eletronics range;
- Next limit technologies;
- Panasonic Avionics Inflight Entertainment;
- Sennheiser;
- Thinkbox Software; e
- VLC Player.

Além da disponibilização de uma suíte de aplicativos é integrado junto ao framework a IDE de programação, ferramenta de desenho, documentação e internacionalização. Na própria IDE de programação também está disponível a ferramenta de desenho de interface gráfica. E ainda é disponibilizado plug-in para desenho de interface com o Qt, para o Visual Studio.

Outra questão que será atacada com o Qt é a plotagem de gráficos, eliminando assim a dependência do GNUPlot. Quanto as formas de onda, será buscada uma opção para eliminar a dependência do GTKWave, mas não há garantia de que será bem-sucedida.

2.5.1 Processos no Qt

Devido ao fato da estrutura do BrownPepper ser dividida em *front-end* (interface) e *back-end* (simulador), é necessária a utilização dos processos e a comunicação entre estes para executar as simulações no ambiente. Portanto, será brevemente explanado como isso é feito utilizando o Qt.

A comunicação entre processos no Qt é provida pela classe QProcess, a qual é a responsável pela execução de programas externos, neste caso o simulador gerado pelo BrownPepper. A classe QProcess trabalha assincronamente, realizando seu trabalho em *background* (segundo plano) e mantendo a interface do usuário responsiva (BLANCHETTE; SUMMERFIELD, 2014). Porém, na implementação desta solução, foi utilizada a execução síncrona dos processos, por questões de controle, e para manter a interface responsiva, foi utilizada a abordagem *multi-threading*.

A comunicação entre os processos ocorre pelos canais de comunicação, como o *stdout* (*standard output channel*) (QT PROJECT, 2014, C++ Classes/QProcess). Portanto, toda a saída gerada na execução do simulador, é recebida pela aplicação e tratada para gerar o *feedback* durante a execução das simulações.

2.5.2 Multi-threading no Qt

Em aplicações convencionais de interface gráfica, uma *thread* é responsável por manter a interface responsiva. Se o usuário invoca uma operação que consuma bastante tempo do processador, como é caso da simulação, a interface congela durante o progresso dessa operação. A utilização de *multi-threads* soluciona esse problema, o qual ocorre na versão atual do BrownPepper.

Outro benefício desta abordagem é a melhor utilização dos recursos do processador, como a execução e utilização das várias *threads* disponíveis nos processadores. Ou seja, diferentes instâncias da mesma aplicação (ex. um simulador) podem ser executadas ao mesmo tempo em diferentes núcleos do processador, resultando numa melhoria de desempenho (BLANCHETTE; SUMMERFIELD, 2006).

A classe QThread é responsável por providenciar a maneira de utilizar *threads*, de forma independente da plataforma para qual a aplicação será compilada. É possível criar uma classe para realizar um processamento qualquer e mover o objeto dessa classe para uma *thread* que não seja a principal (em que a GUI está sendo executada) (QT PROJECT, 2014, C++ Classes/QThread). No caso do RedScarf, essa abordagem é a utilizada para realizar a execução dos simuladores, sendo que, além de não congelar os controles da interface gráfica, permite que sejam executados vários simuladores em paralelo, sendo por padrão de acordo com o número de núcleos presentes no processador mas com a opção de o usuário definir quantas *threads* desejar a partir do menu de configuração.

Existem outras formas de realizar a programação concorrente com o Qt (QT PROJECT, 2014), mas para atender a necessidade de uso do RedScarf, foi utilizada a técnica de mover os objetos da classe de execução para *threads* separadas e realizar as execuções das simulações. O Quadro 1 apresenta uma comparação entre as tecnologias disponíveis para programação concorrente no Qt, e o Quadro 2 relata exemplos de usos que melhor se enquadram nas tecnologias.

Quadro 1. Comparação das soluções concorrentes disponíveis no Qt

Alternativas de implementação	Características						
	1	2	3	4	5	6	7
QThread	C++	Sim	Sim	Sim	Sim		
QRunnable e QThreadPool	C++	Sim					
QtConcurrent::run()	C++					Parc.	
Qt Concurrent (Map, Filter, Reduce)	C++				Sim	Sim	Sim
WorkerScript	QML			Sim			

Fonte: Adaptado de Qt Project (2014)⁹.

Onde:

1. API;
2. A prioridade da *thread* pode ser especificada;
3. A *thread* pode executar um *loop* de eventos;
4. A *thread* pode receber dados atualizados através de sinais;
5. A *thread* pode ser controlada usando sinais;
6. A *thread* pode ser monitorada através de um **QFuture**;
7. Possui suporte pronto para pausar/resumir/cancelar a operação.

Parc.: parcialmente.

Destaca-se, do Quadro 1, que as características 4 e 5 foram as principais contribuintes na escolha da abordagem com QThread, pois manteve a singularidade utilizando sinais para realizar a comunicação e controle com a *thread* principal (responsável pela GUI).

Quadro 2. Exemplos de usos das tecnologias concorrentes disponíveis no Qt

Ciclo de vida da <i>Thread</i>	Tarefa a ser executada	Solução
Uma chamada	Executar um método em outra <i>thread</i>	QtConcurrent::run(); herança QRunnable; herança QThread
Uma chamada	Operações em lotes	QtConcurrent
Uma chamada	Execuções longas informando a GUI	QThread
Permanente	Objeto vivendo em outra <i>thread</i> , várias tarefas	Objeto movido para QThread
Permanente	Objeto vivendo em outra <i>thread</i> , tarefas repetitivas	Objeto movido para QThread e utilizar um temporizador

Fonte: Adaptado de Qt Project (2014)⁹.

⁹ Caminho direto do local referenciado: <<http://qt-project.org/doc/qt-5/thread-technologies.html>>, acesso em 02 Maio 2014.

Do Quadro 2, o destaque é dado para a informação que possui o ciclo de vida de apenas uma chamada, sendo que, para cada simulador gerado, é necessária apenas uma execução. Também devido ao longo tempo despendido na simulação e em conformidade com a alternativa justificada no Quadro 1, a utilização da QThread é a solução que melhor se enquadra no contexto do RedScarf.

2.6 TRABALHOS RELACIONADOS

Nesta seção, serão apresentadas e comparadas ferramentas de simulação de Redes-em-Chip que exploram características relacionadas à avaliação de desempenho. Algumas das ferramentas discutidas foram selecionadas do trabalho de Achballah e Saoud (2013), no qual foi feita uma busca não exaustiva por ferramentas relacionadas com o projeto e simulação de NoCs e a caracterização das ferramentas levantadas.

O NoCGEN (CHAN; PARAMESWARAN, 2004) é um gerador de NoC usado para criar uma descrição de NoC simulável e sintetizável. A ferramenta permite configurar os parâmetros dos componentes da rede, como o uso de diferentes roteadores, portas de comunicação, algoritmos de roteamento, largura dos canais de dados e profundidade dos *buffers*. A simulação é realizada com uma abordagem híbrida de SystemC e VHDL, sendo que o SystemC é utilizado para produzir e consumir o tráfego da rede. Como resultado da avaliação de desempenho, são oferecidas as métricas, vazão e a latência de comunicação.

O Nirgam (2007) é um simulador de Redes-em-Chip baseado em eventos discretos, com acurácia em nível de ciclos. Não possui interface gráfica e é escrito em SystemC. Os parâmetros de topologia, frequência de operação, profundidade dos *buffers*, tamanho do *flit* e canais virtuais são configuráveis. Como resultado, disponibiliza a vazão e a latência da rede.

O NoC Simulator (2007) é um simulador escrito em C++ orientado a objetos, *open-source*, que modela uma topologia de Rede-em-Chip em malha. Componentes como canais virtuais, unidades de geração de tráfego e portas, são configuráveis. É dirigido a eventos, possui algoritmos de roteamento do tipo XY e XY adaptativo. Não possui interface gráfica e como resultado disponibiliza a latência e vazão da rede.

O OCCN – On-Chip Communication Network (2011) é um *framework open-source* para especificação, modelagem e simulação de arquiteturas de comunicação em chip. Com

uma API orientada a objetos em C++ e construído sobre o SystemC, o OCCN possui funções pré-definidas para a análise de desempenho oferecendo métricas de latência e vazão.

O trabalho realizado por Gottschling, Ying e Hofmann (2012) possui a característica de uma aplicação separada em *front-end* (interface gráfica do usuário), desenvolvido em Qt, e *back-end* (*framework* de simulação), desenvolvido em SystemC. Além dos resultados de latência e vazão disponibilizados ao fim das simulações, gera acompanhamento em tempo de execução das simulações correntes através de uma representação da rede em 3D utilizando OpenGL¹⁰.

O BookSim (2013) é um simulador com precisão no nível de ciclo (ciclo a ciclo) de rede de interconexão. Na sua versão mais recente, suporta uma larga quantidade de topologias como malhas, toróides e *flattened butterfly* (borboleta achatada). Também oferece diversos algoritmos de roteamento e inclui opções para customizar a arquitetura dos roteadores. É *open-source* (fonte aberto), escrito em C++ e não possui interface gráfica. Como resultado, disponibiliza a vazão e a latência da rede.

O Noxim (2014) é um simulador desenvolvido utilizando SystemC, *open-source* e utilizável por meio da linha de comandos. Os parâmetros configuráveis são o tamanho da rede, o tamanho dos *buffers*, tamanho dos pacotes, algoritmo de roteamento, taxa de injeção de pacotes, distribuição de tráfego, padrão de tráfego e distribuição de tráfego em *hot-spot* (pontos quentes). O simulador permite a análise da vazão, da latência e do consumo de energia.

O NNSE (Nostrum NoC *Simulation Environment* – Ambiente de simulação Nostum de NoC) é um simulador baseado em SystemC, possui uma GUI para configurar o tamanho da rede, a topologia, políticas de roteamento e padrões de tráfego. A ferramenta disponibiliza como resultado a vazão e a latência da rede e ainda está na versão de testes (NNSE, 2014).

O Atlas (2014) é um ambiente para geração e avaliação de NoCs escrito em Java que automatiza os processos relacionados ao fluxo de projeto, composto por: geração da NoC, geração de tráfego, simulação, avaliação de desempenho e consumo. Possui interface gráfica e na análise de desempenho os resultados são: (i) o número de pacotes recebidos; (ii) tempo

¹⁰ Ambiente de desenvolvimento de aplicações gráficas portáveis, em 2D ou 3D (OPENGL, 2014).

médio de entrega dos pacotes, em ciclos; (*iii*) tempo total para entregar todos os pacotes, em ciclos; e (*iv*) a média, o mínimo e máximo desvio padrão de tempo para um pacote, em ciclos de relógio.

O Arteris FlexNoC Interconnect IP (ARTERIS, 2014) é uma solução comercial que engloba várias ferramentas para o projeto de SoCs. É baseado em modelos SystemC e disponibiliza resultado da avaliação de desempenho com base na latência e na vazão da rede.

Um comparativo entre as ferramentas relacionadas nesta seção é realizado no Quadro 3. Todas as ferramentas têm suporte à análise de desempenho, tendo como métricas de avaliação a vazão e a latência. A utilização da abordagem *multi-thread* não é adotada ou informada nas ferramentas avaliadas. A principal tecnologia utilizada na construção dos simuladores é o SystemC. A plataforma mais utilizada para execução dos ambientes é o Linux. O código fonte de algumas das aplicações é disponibilizado. E a utilização de interface gráfica não é tão comum nos trabalhos com o propósito de simulação em NoCs. O Quadro 3 também inclui a versão atual, o BrownPepper (1.0) e a versão em desenvolvimento (RedScarf) de modo a posicionar o presente trabalho em relação aos demais. Nota-se, portanto, que o RedScarf será o único ambiente com suporte à execução *multi-thread* no *backend*.

Quadro 3. Comparativo dos trabalhos relacionados

Ferramentas	Propriedades				
	GUI	Tecnologias	<i>Multi-thread</i>	Plataforma	Distribuição
Booksim	não	C++	não	Unix, Windows (Cygwin)	<i>Open-source</i>
NoC Simulator	não	C++	não	Compatíveis com STL	<i>Open-source</i>
Noxim	não	SystemC	não	Compatíveis com SystemC	<i>Open-source</i>
NNSE	SIM	SystemC	<i>n.d.</i>	<i>n.d.</i>	<i>n.d.</i>
Nirgam	não	SystemC	não	Compatíveis com SystemC	<i>Open-source</i>
OCCN	não	SystemC	não	Unix, Sun/Solaris	<i>Open-source</i>
Atlas	SIM	Java	<i>n.d.</i>	Linux, Windows, Sun, OS X	<i>n.d.</i>
FlexNoC	SIM	SystemC	<i>n.d.</i>	<i>n.d.</i>	Comercial
NoCGEN	não	SystemC/VHDL	<i>n.d.</i>	<i>n.d.</i>	<i>n.d.</i>
GSNOC UI	SIM	Qt e SystemC	<i>n.d.</i>	Testado apenas em Linux	<i>n.d.</i>
BrownPepper	SIM	GTK e SystemC	não	Linux	<i>Restrita</i>
RedScarf	SIM	Qt e SystemC	SIM	Linux, Windows, OS X	<i>Freeware</i>

Onde *n.d.*: Informação não disponível.

3 DESENVOLVIMENTO

Neste capítulo são apresentadas as características do produto deste trabalho, as quais são baseadas nas funcionalidades existentes no BrownPepper, nas necessidades levantadas pelos pesquisadores do grupo de pesquisa que este projeto está inserido e nas tecnologias utilizadas para o desenvolvimento deste sistema.

3.1 VISÃO GERAL

O RedScarf é um ambiente de simulação de Rede-em-Chip com ênfase na avaliação de desempenho que facilita ao usuário o processo de parametrização das configurações da rede, tráfego e roteadores. E também automatiza a construção dos modelos da NoC, geração dos simuladores, execução da simulação e análise dos resultados.

Além do dimensionamento da rede, o RedScarf oferece de forma simples a configuração de tráfego para cada roteador, e ainda, possui modelos de distribuição espacial para facilitar a configuração de tráfego para múltiplos destinos ou com distribuições que exploram a localidade dos nodos na rede. Utilizando apenas a interface gráfica é possível definir diversos fluxos para cada endereço.

Oferece graficamente a possibilidade de definir diferentes experimentos variando as opções disponíveis dos modelos de roteadores acoplados à ferramenta. E permite que os experimentos sejam verificados em diferentes frequências de operação.

Na simulação, suporta a execução paralela de múltiplos experimentos (simuladores) e disponibiliza ao usuário a opção de definir o número de *threads* que deseja para a simulação.

Possui precisão em nível de ciclos de relógio, o que traz resultados exatos embora despenda tempo na execução, desvantagem que é tratada com a execução de múltiplos experimentos simultaneamente em computadores *multi-core*.

Na análise de desempenho, permite desconsiderar um percentual de pacotes que correspondem às etapas de *warm-up* e *drain* da simulação, sendo o usuário o responsável por definir estes percentuais. Como métricas, disponibiliza a vazão, latência da rede e prazos cumpridos, os quais podem ser visualizados via gráficos ou tabelas, sendo que essas podem ser exportadas na forma arquivo CSV (Comma Separated Value) para uso em outras

ferramentas (ex. MS Excel). Permite visualizar a simulação por diagramas de formas de ondas com o auxílio de uma ferramenta externa. Em termos de facilidade, oferece a leitura de arquivo em formato XML para a configuração dos parâmetros do ambiente.

E depois de feito todo o processo de simulação, dispõe ao usuário a possibilidade de armazenar os resultados gerados, com uma taxa de compressão de aproximadamente 85%, para economizar espaço em disco e realizar análises posteriores dos resultados da simulação.

Tudo isso rodando nas plataformas operacionais convencionais dos sistemas *desktop*, pois é multiplataforma e é totalmente funcional em Windows, Mac OS X e Linux, podendo ser portado para outras plataformas que suportem Qt, como, por exemplo, as para dispositivos móveis. Permite ainda ao usuário escolher o idioma que deseja para a ferramenta, tendo hoje entre os disponíveis o inglês e o português.

3.2 ARQUITETURA DO SOFTWARE

O ambiente de simulação é dividido em duas partes, o *front-end* e *back-end*. Este trabalho trata do *front-end*, que é responsável pela configuração dos diversos parâmetros necessários para a simulação, criação e construção dos simuladores, automatização das execuções das simulações, análise sobre os resultados da simulação, exibição dos gráficos e relatórios, instanciação de ferramenta para visualização das formas de ondas, arquivamento e carregamento de configurações e resultados das simulações. O *back-end* é o simulador em si e não foi objeto de trabalho na ferramenta, sendo que já havia sido implementado em trabalho anterior.

O *front-end* foi todo implementado e dividido em camadas utilizando o padrão MVC de construção de software orientado a objetos. Sua organização foi feita dividindo todo o trabalho em ferramentas para cada função, ou seja, quebrando o problema em pequenas partes, para facilitar a compreensão e a manutenção do software, quando necessária. A arquitetura do ambiente pode ser vista na Figura 9.

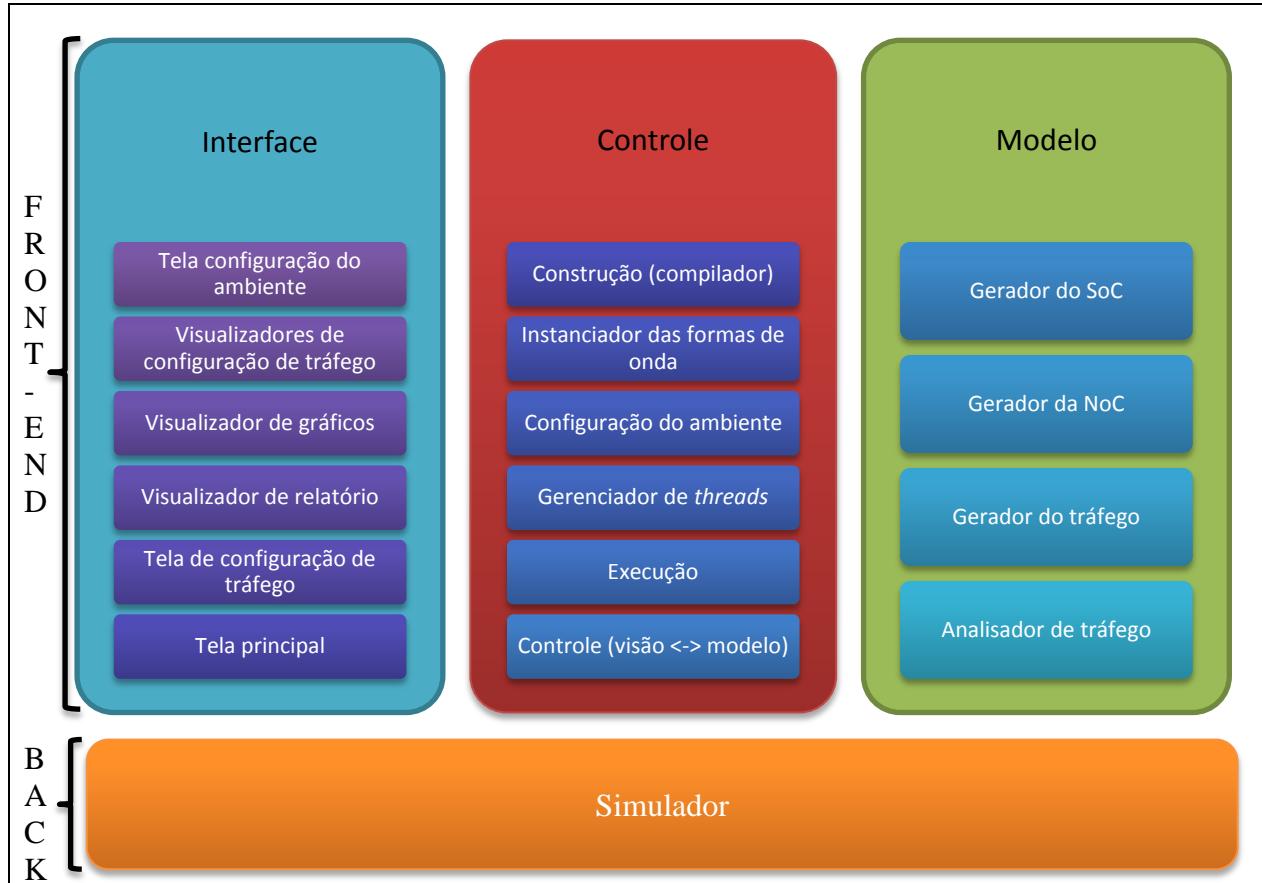


Figura 9. Arquitetura do ambiente

3.2.1 Interface

A interface foi toda feita em Qt, com o auxílio da ferramenta Designer do próprio *framework*. A maioria das funções está relacionada à tela principal e todas as caixas de diálogo estão implementadas separadamente.

A interface é multilíngue e permite o usuário definir o idioma que deseja para o ambiente. Os idiomas disponíveis até o momento são o inglês e português, mas pode facilmente ser expandido para outros idiomas, sem a necessidade de recompilar o código-fonte, pois apenas é necessário gerar o arquivo de tradução e adicionar informações ao arquivo de configuração do ambiente.

A tela principal é dividida em três abas que representam o fluxo de uso da ferramenta. Sendo a primeira aba de configuração da rede e dos parâmetros de tráfego, a segunda aba de configuração dos experimentos com a parametrização dos roteadores e a terceira aba é a de análise dos resultados, em que o usuário realiza a inferência sobre os dados gerados na

simulação, visualiza os gráficos e relatórios e pode instanciar uma ferramenta externa para a visualização das formas de ondas. As telas do sistema são apresentadas no Apêndice B.7 .

As ações realizadas pelo usuário na interface são repassadas para o controle e no controle é feito o tratamento adequado. As ações são enviadas ao controle via *signals* do Qt. E capturadas no controle pelos *slots* também do Qt. Alguns controles de tela são feitos na própria interface e não possuem intervenção do controlador da aplicação.

Toda a configuração de tráfego e experimentos é realizada pela interface gráfica, facilitando a parametrização do simulador para a simulação.

3.2.2 Controle

No controle estão todos os tratamentos de configuração do sistema, ambiente e funcionalidades que o sistema possui. Para isto o controle capture os eventos da interface emitidos pelos *signals* e realiza os tratamentos com métodos (*slots*) para cada tipo de evento. Utiliza do modelo para manter os dados necessários para o correto funcionamento da aplicação e para realizar operações com as ferramentas disponíveis no modelo. O controle é o meio de campo entre a interface e o modelo, é ele o responsável por tratar os dados da interface e quando preciso adequá-los para repassar ao modelo realizar o processamento.

É no controle que as estruturas utilizadas na aplicação são alocadas e mantidas, portanto qualquer operação realizada é de responsabilidade do controle solicitar realizar e monitorar o estado para sempre emitir *feedback* ao usuário. O paradigma de tratamento de eventos do Qt baseado em *signals* e *slots* é uma representação do padrão de projeto *observer*.

As ferramentas que fazem parte do controle são:

- *Builder*: responsável por realizar a construção do ambiente e geração dos modelos de tráfego com o auxílio da ferramenta de geração do modelo de tráfego disponível no modelo. O *builder* utiliza o gcc como compilador compatível com o SystemC em Unix, e o MinGW no Windows;
- *Instanciador de ferramenta para visualização das formas de ondas*: responsável por executar uma ferramenta externa através de um novo processo, desde que a ferramenta esteja devidamente configurada no ambiente;

- Configuração do ambiente: ferramenta que possui os dados de configuração do ambiente para o correto funcionamento da aplicação, como os diretórios de trabalho, do compilador e SystemC;
- Gerenciador de threads: responsável por executar as simulações de acordo com o número de *threads* que estão configuradas, além disso, monitora as *threads* e informa o final da execução de todas as *threads*, também permite a parada das *threads*;
- Execução: responsável por instanciar os simuladores e executá-los passando os parâmetros corretos para cada instância;
- Controle: o controle em si, é responsável por capturar os eventos da interface e utilizar as ferramentas do controle e modelo para operar todas as funcionalidades. É o núcleo da aplicação.

3.2.3 Modelo

O modelo possui as estruturas para manter todos os dados que a aplicação necessita, como a representação dos nodos da rede. Além disso, comporta ferramentas para a geração de modelos SystemC de componentes do sistema, da rede e do *testbench* da simulação, além do arquivo de modelo de tráfego. Também inclui ferramenta para análise dos dados com foco na avaliação de desempenho. A estruturação em diferentes ferramentas facilita a manutenção e atualização do ambiente.

Todo o modelo é programado utilizando apenas a biblioteca padrão do C++ (stl – *standard template library*). E para as ferramentas, alguns padrões de projeto foram adotados, como o *strategy* para a geração da rede e do analisador de tráfego, facilitando o acoplamento de novos geradores de redes e analisadores de tráfego.

A ferramenta de geração da rede é responsável por gerar o arquivo com o arranjo da rede-em-chip a ser gerada e o arquivo de parametrização da rede, sendo que uma das características da rede atacada é ser parametrizável por possuir roteadores parametrizáveis.

A ferramenta de geração do SoC é responsável pela criação do arquivo de *testbench* e do arquivo de ligação dos sinais da rede com os sinais do *testbench* e para a geração do arquivo de formas de ondas.

A ferramenta de geração do modelo de tráfego é responsável por ler os parâmetros da interface, reajustá-los quando necessário e gerar o arquivo de tráfego que serve como o estímulo de entrada do simulador.

A ferramenta de análise, que no momento está implementada apenas com o foco em desempenho, é responsável por ler os *logs* da simulação e realizar os cálculos de latência e vazão que serão disponibilizados ao usuário.

3.2.4 Fluxo das ferramentas

O fluxo principal de uso da ferramenta parte da definição dos parâmetros na interface, geração dos arquivos da rede e SoC, construção dos simuladores, execução da simulação, análise dos resultados e apresentação dos resultados para o usuário. Uma visão macro do fluxo é apresentada na Figura 10, enquanto a Figura 11 apresenta uma visão mais detalhada. O diagrama de atividades da ferramenta é mostrado no Apêndice B.6

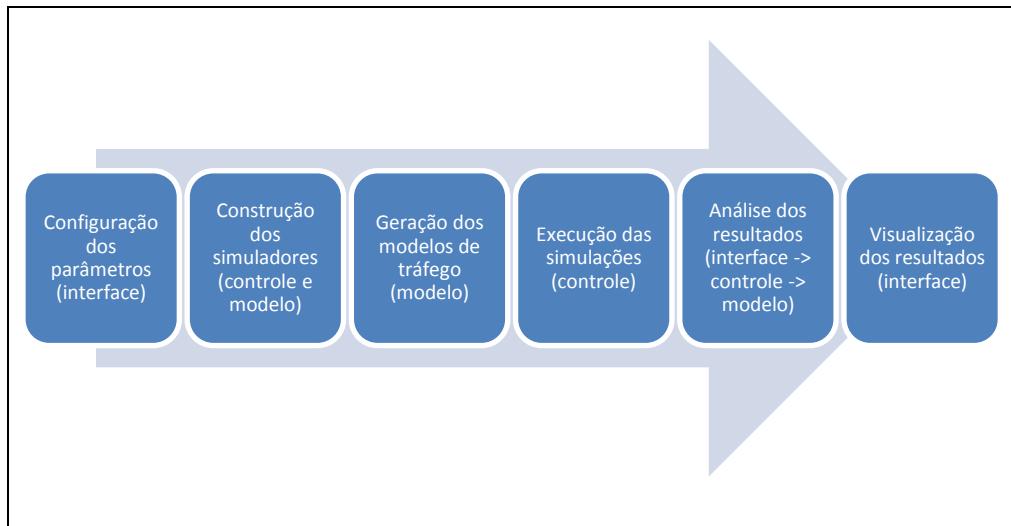


Figura 10. Fluxo de uso da ferramenta

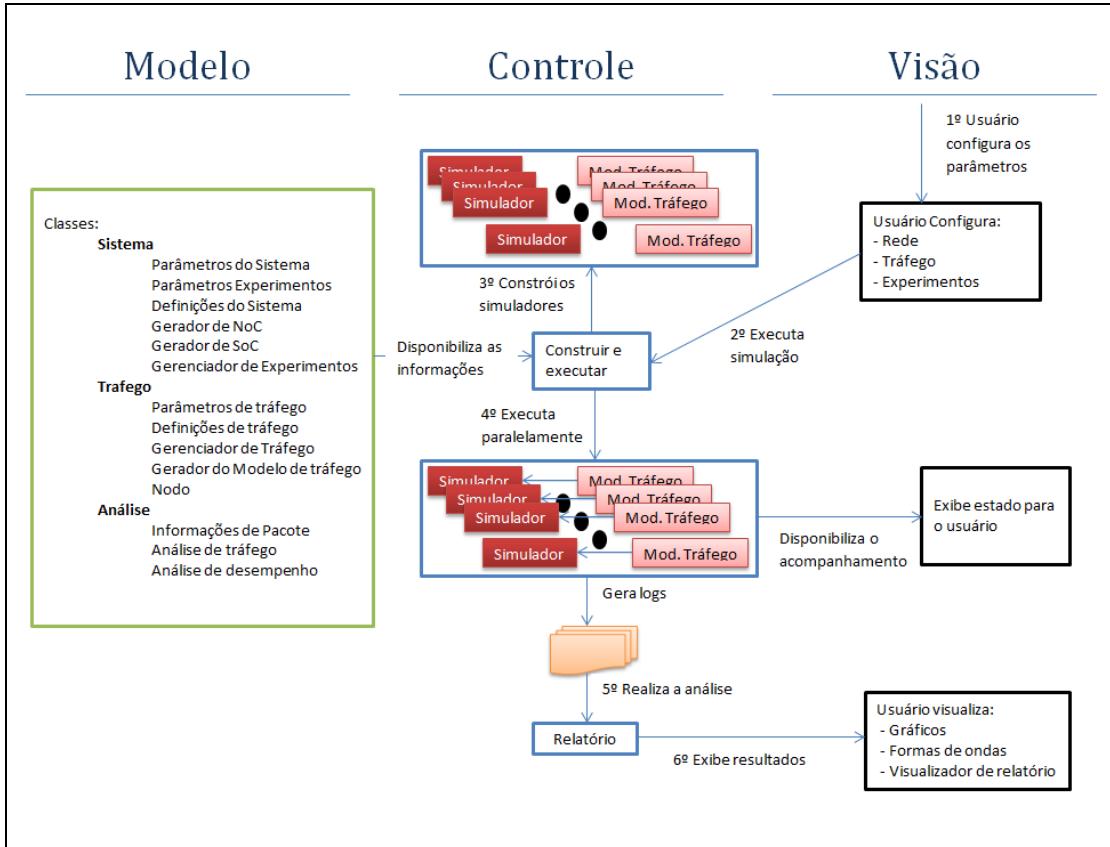


Figura 11. Fluxo de uso da ferramenta

A comunicação do *front-end* com o *back-end* é feita através de IPC (Inter Process Communication – Comunicação entre processos), no qual o canal de saída (*stdout*) do processo de *back-end* é capturado e tratado pelo processo de *front-end*.

3.3 PROJETO

O ambiente proposto neste trabalho é o RedScarf, com código-fonte reestruturado e orientado a objetos. Ele atende o modelo MVC de construção de software em camadas, aumentando a granularidade e, portanto a reusabilidade do projeto em relação à versão anterior (o BrownPepper). O framework Qt multiplataforma facilita as alterações na interface e na implementação da ferramenta.

O RedScarf conta com todas as características do BrownPepper e incorpora novas funcionalidades, tornando a nova versão mais robusta, estável e referência em simulação na SoCIN.

As necessidades inicialmente encontradas e que motivaram o processo de reengenharia do ambiente BrownPepper foram: a de adições de funcionalidades, correções e alterações nas

funcionalidades existentes. Como: (i) execução *multi-thread* (menor tempo para simular); (ii) acompanhamento do estado da rede em tempo de simulação (visualização gráfica); (iii) evitar fechamento do simulador para configurações inválidas (totalmente operacional); (iv) adicionar tratamento para configurações que não são suportadas (ex.: distribuição não uniforme e local utilizando a função de probabilidade Pareto) (restringir ações); e (v) geração de XML como arquivo de configuração do ambiente (arquivo XML contendo configuração de sistema e tráfego).

As correções feitas foram apontadas pelos utilizadores e, à medida que o processo de reengenharia da ferramenta foi evoluindo, estas e outras foram identificadas e realizadas. Por exemplo, na geração do modelo de tráfego da rede, algumas inconsistências faziam com que fluxos de comunicação inválidos fossem gerados, e em alguns casos a aplicação era encerrada.

Em relação às alterações, algumas mudanças visuais e comportamentais foram feitas, tais como: posicionamento e dimensionamento das representações dos nodos na rede, exibição de todos os parâmetros na configuração de tráfego, console de mensagens integrado à ferramenta, acompanhamento básico da simulação, criação da barra de menus e ferramentas, geração de arquivo de configuração em formato XML, entre outras, foram adotadas.

Este trabalho concluiu o processo de reengenharia de software iniciado a fim de atingir o estado da primeira versão da ferramenta e foi além do que já existia, tornando o *front-end* mais flexível para alterações, além de robusto e atrativo com as novas funcionalidades. As funcionalidades que se desejava que fizessem parte da nova versão para que a evolução da ferramenta tivesse impacto notável e significativo incluem:

- Armazenamento das configurações gerais do sistema como: tamanho do sistema, largura do canal de comunicação, padrões de tráfego, experimentos, frequência de operação e opções de execução;
- Definição do tempo a ser simulado na execução das simulações;
- Geração de arquivo de relatório no formato CSV (Comma Separated Value) formatado para a realização de inferência através de planilhas eletrônicas (ex. Microsoft Excel);

- Armazenamento dos resultados das simulações para criação de um portfólio de experimentos realizados, para análise posterior;
- Criação de uma interface de configuração dos ajustes das preferências do ambiente;
- Definição de fluxos maliciosos para ataque às propriedades de segurança da rede (ataques predefinidos e disponibilizados);
- Refinamento dos resultados da simulação nos pontos críticos da rede (pontos de saturação);
- Acompanhamento em tempo de simulação da rede, através da análise ciclo-a-ciclo dos roteadores e dos enlaces;
- Recurso visual para apresentar os resultados da análise em tempo de simulação;
- Eliminar as dependências de ferramentas de terceiros para a geração dos resultados visuais, que são o GNUPlot e GTKWave, afim de, facilitar a portabilidade para as plataformas Windows e Mac OS X;
- Estimar o custo em área e potência baseado em modelos analíticos para uma determinada tecnologia.

Somente o acompanhamento visual em tempo de simulação que era previsto para este trabalho não foi implementado para não comprometer a qualidade do resultado atingido. O refinamento dos resultados, definição de fluxos maliciosos, estimativa de custo e área, e eliminação da dependência de uma ferramenta de formas de ondas, eram necessidades desejáveis, mas não obrigatórias para este trabalho.

3.3.1 Requisitos funcionais

Com os aspectos apontados e as proposições estabelecidas, foi feito o levantamento dos requisitos de toda a ferramenta. A seguir são listados os requisitos funcionais da ferramenta:

- RF 01: O sistema deverá permitir o usuário definir o tamanho da rede a ser simulada;
- RF 02: O sistema deverá permitir o usuário definir a largura do canal de comunicação da rede;

- RF 03: O sistema deverá permitir o usuário definir até 4 padrões de tráfego para cada terminal da rede;
- RF 04: O sistema deverá permitir o usuário visualizar a configuração dos padrões de tráfegos aplicados aos nodos terminais da rede;
- RF 05: O sistema deverá permitir o usuário salvar a configuração do sistema, tráfegos configurados e experimentos;
- RF 06: O sistema deverá permitir o usuário carregar uma configuração do sistema e tráfegos armazenados;
- RF 07: O sistema deverá permitir o usuário configurar de 1 (um) à 5 (cinco) experimentos para a simulação;
- RF 08: O sistema deverá permitir o usuário definir um intervalo de frequências de operação para as simulações;
- RF 09: O sistema deverá permitir o usuário definir se irá gerar o relatório para visualização das formas de onda na simulação, e oferecer as opções de visualização;
- RF 10: O sistema deverá permitir o usuário definir os fluxos que devem ser analisados na apresentação dos resultados;
- RF 11: O sistema deverá permitir o usuário definir qual gráfico deseja gerar, oferecendo as possibilidades de geração;
- RF 12: O sistema deverá permitir o usuário gerar a distribuição de latência das simulações;
- RF 13: O sistema deverá permitir o usuário gerar as formas de ondas da simulação, porém através de uma ferramenta de terceiros;
- RF 14: O sistema deverá permitir o usuário acompanhar o estado da rede durante as simulações;
- RF 15: O sistema deverá permitir o usuário armazenar os resultados das simulações;
- RF 16: O sistema deverá permitir o usuário carregar os resultados armazenados de simulações já realizadas;

- RF 17: O sistema deverá permitir o usuário definir o tempo a ser simulado;
- RF 18: O sistema deverá permitir o usuário ajustar as preferências das configurações;
- RF 19: O sistema deverá gerar um arquivo csv que contenha os resultados formatados e possibilite a importação por ferramentas de planilhas eletrônicas (ex. Excel);
- RF 20: O sistema deverá apresentar a estimativa de custo em área e potência. Utilizando os modelos analíticos desenvolvidos no laboratório para uma determinada tecnologia de circuitos;
- RF 21: O sistema deverá permitir o usuário definir fluxos maliciosos para ataque às propriedades de segurança da rede, disponibilizando ataques predefinidos;
- RF 22: O sistema deverá realizar uma análise sobre os dados da simulação e refinar os resultados nos pontos críticos da rede (pontos de saturação);
- RF 23: O sistema deverá conter uma opção que permita a visualização do manual do usuário.

Dos requisitos funcionais levantados, é importante ressaltar que, do RF 01 até o RF 13, as funcionalidades são referentes às existentes na primeira versão do ambiente, e também, do RF 20 ao RF22 são requisitos desejáveis para a aplicação, mas que não eram obrigatórios para este trabalho afim de não comprometer o resultado a ser atingido. Os requisitos não-funcionais e as regras de negócio são apresentados no Apêndice A, no qual é feito todo o detalhamento do projeto de software.

3.4 IMPLEMENTAÇÃO

Como pretendido, a implementação do ambiente atendeu todas as funcionalidades presentes em sua primeira versão e ainda as novas características que foram projetadas. Das novas características têm-se:

1. Internacionalização da interface (possibilidade de incorporar vários idiomas à ferramenta, atualmente idiomas inglês e português);
2. Menu de configuração das opções do ambiente com seleção dos diretórios de trabalho, local da biblioteca SystemC, escolha de idioma e definição do número de

threads para a simulação, além de a configuração do diretório do compilador MinGW no ambiente Windows;

3. Configuração de simulação baseada em tempo, tanto em nanossegundos quanto em ciclos de relógio, embora limitada à simular um número predefinido de pacotes no gerador de tráfego;
4. Visualização da configuração de tráfego em forma de árvore baseada em modelo XML de configuração;
5. Opção de salvar e carregar resultados das simulações, evitando ter de dispensar tempo de simulação para experimentos já executados. Este salvamento e carregamento não são automáticos e dependem da intervenção do usuário;
6. Geração de relatório em formato CSV;
7. Eliminação da dependência do GNUPlot para a geração de gráficos. Agora o ambiente possui um plotador próprio;
8. Execução multi-thread das simulações, acelerando o tempo de experimentação;
9. Correção de usabilidade para inferência na análise considerando *warm-up* e *drain*;
10. Disponibilidade multiplataforma; e
11. Documentação disponível, em nível de projeto de software, manual do usuário e documentação de código-fonte.

Os requisitos considerados desejáveis não foram implementados para não comprometer a qualidade deste trabalho no tempo em que foi realizado. O acompanhamento visual em tempo de simulação ficou limitado à apresentação de uma barra de progresso durante as simulações, isto devido à complexidade de representação visual da rede e disponibilização das informações necessárias por parte do simulador (*back-end*) para tal finalidade.

A internacionalização foi feita apenas para o português, tendo inglês como o idioma em que a aplicação possui as *strings* codificadas, mas a implementação foi feita de forma extensível. Isso permite ampliar o número de idiomas da ferramenta de forma fácil, apenas criando o arquivo de tradução do Qt para o idioma alvo e adicionando o novo idioma ao arquivo de configuração, sem a necessidade de recompilar o código-fonte. A internacionalização traduz inclusive o relatório gerado em CSV.

O menu de configuração também é escalável, podendo ser expandido sem muito esforço, e além disso, novas categorias podem ser desenhadas usando a ferramenta de desenho de interfaces do Qt.

A geração de arquivo XML para armazenar as configurações do ambiente facilita a interoperabilidade com outros sistemas pois o formato XML já é bem difundido e utilizado nas mais diversas aplicações. Com isso também é possível visualizar a configuração do ambiente em formato de árvore, apresentando os dados dos padrões de tráfego, experimentos, frequências de operação, opções de formas de onda e opções de parada da simulação.

A possibilidade de armazenar os resultados das simulações trouxe grande vantagem, permitindo carregar uma simulação já executada. No armazenamento foi utilizada a compressão binária de dados do Qt, que provedu uma taxa de aproximadamente 85% de compressão, o que significa dizer que uma simulação que gera 6,6 GB pode ser armazenada e tem como resultado um arquivo de 1 GB. Essa funcionalidade é especialmente útil quando se deseja arquivar os resultados de um experimento para uma análise posterior.

A funcionalidade de gerar relatório em formato CSV com os dados da análise de desempenho da simulação permite que esses dados sejam reutilizados em outras aplicações como planilhas eletrônicas. Isso possibilita realizar análise desses dados da forma que for conveniente ao utilizador, como na própria geração de gráficos diferentes do que a ferramenta exibe.

A eliminação da dependência do GNUPlot traz mais robustez e portabilidade da aplicação para diferentes plataformas, sendo que a apresentação dos gráficos é de suma importância uma vez que facilita a inferência dos resultados pelos analistas com a representação visual da análise de desempenho.

3.5 RESULTADOS

Da execução *multi-thread*, obtém-se o ganho de tempo para a execução das simulações, trazendo, portanto, o benefício aos utilizadores de diminuir o tempo despendido para investigar diferentes configurações. Esse é um fator importante quando se explora o espaço de projeto das Redes-em-Chip, resultando em menor tempo para experimentação. Com a finalidade de demonstrar o ganho de tempo, foram feitos experimentos variando o número de *threads* de 1 a 22 para a simulação em um servidor com dois processadores Xeon E5410

com um total de 8 núcleos operando a 2.33GHz. Obteve-se que, executando um experimento da SoCIN com arranjo em malha 4x4 (no total de 16 roteadores), tráfego uniforme (no qual um nodo envia pacotes para todos os demais nodos na rede), com a quantidade de 1000 pacotes para cada destino, variando a frequência de operação 1 a 100 MHz, em passos de 10 MHz (1, 10, 20, 30,..., 90, 100 MHz) – total de 22 simulações. O experimento na primeira versão da ferramenta que executa sobre uma única *thread* demorou cerca de 55 minutos, enquanto na melhor configuração do RedScarf demorou cerca de 9,5 minutos. A Figura 12 apresenta a execução com a variação do número de *threads* simultâneas (1 a 22) no servidor utilizado e a aceleração com a execução *multi-thread*.

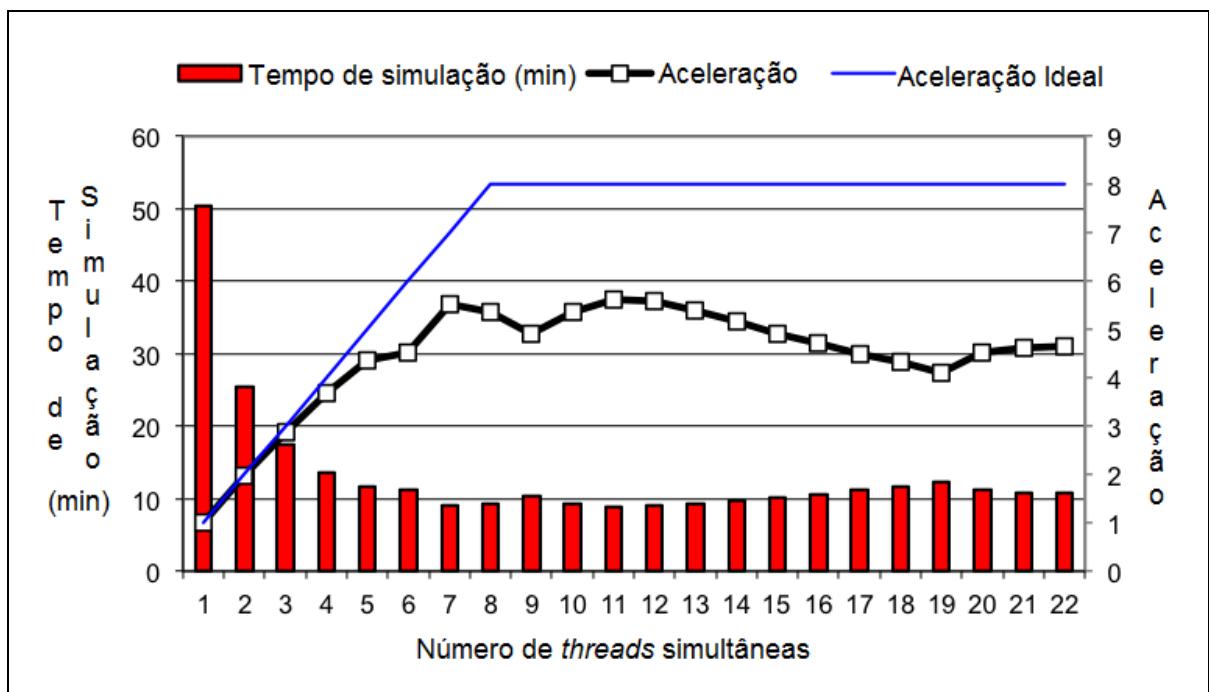


Figura 12. Aceleração da execução *multi-thread*

É possível notar que a aceleração ocorre até que o número de *threads* seja igual ao número de *cores* do servidor (nesse caso, 8). Após este número, é apresentada uma flutuação que está diretamente relacionada com o escalonamento de processos do processador, acesso à memória e operações de acesso ao disco. A aceleração ideal indica que à medida que dobra o número de *threads* o tempo deve cair pela metade, até que atinja o tempo mínimo para a execução de uma simulação, mas esta é uma definição teórica, pois não é somente processamento que a simulação consome, também efetua acesso a disco o que dispõe muito mais tempo se comparado ao tempo de processamento.

A inferência sobre os resultados da simulação, considerando a análise sobre os pacotes contando com a configuração de *warm-up* e *drain* era realizada na ferramenta BrownPepper, porém o momento no qual era feita a análise estava incoerente com o sequenciamento das atividades de utilização da ferramenta. Isso foi observado durante a implementação desta funcionalidade e corrigido.

Foram executados testes nas plataformas Windows (Windows XP e Windows 7), Linux e Mac OS X. Em ambas, os testes foram bem sucedidos e a ferramenta está totalmente funcional para essas plataformas. O pré-requisito para o funcionamento é um compilador compatível com o SystemC (que deve estar devidamente instalado e apontado) e a disponibilidade de um visualizador de diagrama de formas de onda (ex. GTKWave no Linux).

A documentação estará disponível em diferentes níveis/usos, manual de usuário (ANEXO A.), documentação do código-fonte, documento de especificação de software e este trabalho¹¹.

Os testes das funcionalidades foram feitos de forma unitária e integrada, não foram feitos testes exaustivos para todas as funcionalidades e nem foi seguido algum plano de teste específico. Mas como exemplo, para a geração do modelo de tráfego, que é a entrada do simulador, foram feitos 1375 testes, para comparar se os resultados gerados na primeira versão da ferramenta e na versão proposta seriam idênticos. Muitos erros, problemas e inconsistências foram encontradas durante todo o processo, o que retardou a implementação e aprimoramento de algumas funcionalidades, mas foi obtida uma versão estável da ferramenta com a correção dos erros encontrados.

O código do *front-end* possui 22.463 linhas de código C++, incluindo:

- Modelo: 6.625 linhas;
- Controle: 4.448 linhas;
- Interface: 3.848 linhas; e
- Código gerado automaticamente pelo Qt: 7.542 linhas.

¹¹ O Manual do Usuário e a documentação do código fonte (usando Doxygen) estava em elaboração no momento da entrega deste volume para análise da banca. O manual do usuário foi contemplado e adicionado ao ANEXO A.

Para o funcionamento da aplicação, é necessário um compilador C++ para a construção dos simuladores em SystemC, e para a visualização das formas de ondas é necessária a configuração de uma aplicação externa.

O executável no Linux possui 1,2 MB de tamanho e na execução apenas do *front-end* aproximadamente 28 MB de memória são utilizados para manter a aplicação. No Windows, o executável possui 1,43 MB e consome aproximadamente 41,2 MB de memória. No Mac OS X, o executável possui 1,4 MB e consome aproximadamente 37 MB de memória. Em ambas as plataformas, o uso de processamento é mínimo para as operações realizadas na interface gráfica pelo usuário. No *back-end*, o consumo de memória varia de acordo com o experimento configurado e o uso da CPU é máximo na quantidade de *threads* configuradas para a simulação.

Para demonstrar o funcionamento do ambiente, a seguir, são apresentados resultados para um experimento de avaliação do desempenho de uma rede 4x4 com canal de comunicação de 32 bits, tráfego uniforme para todos os nodos da rede e 3 configurações de roteador, em que a única variação nas configurações foi a profundidade dos *buffers* de entrada (2, 4 e 8 flits). A Figura 13(a) apresenta um gráfico gerado pela ferramenta. A Figura 13(b) apresenta o relatório CSV aberto em uma planilha do Excel.

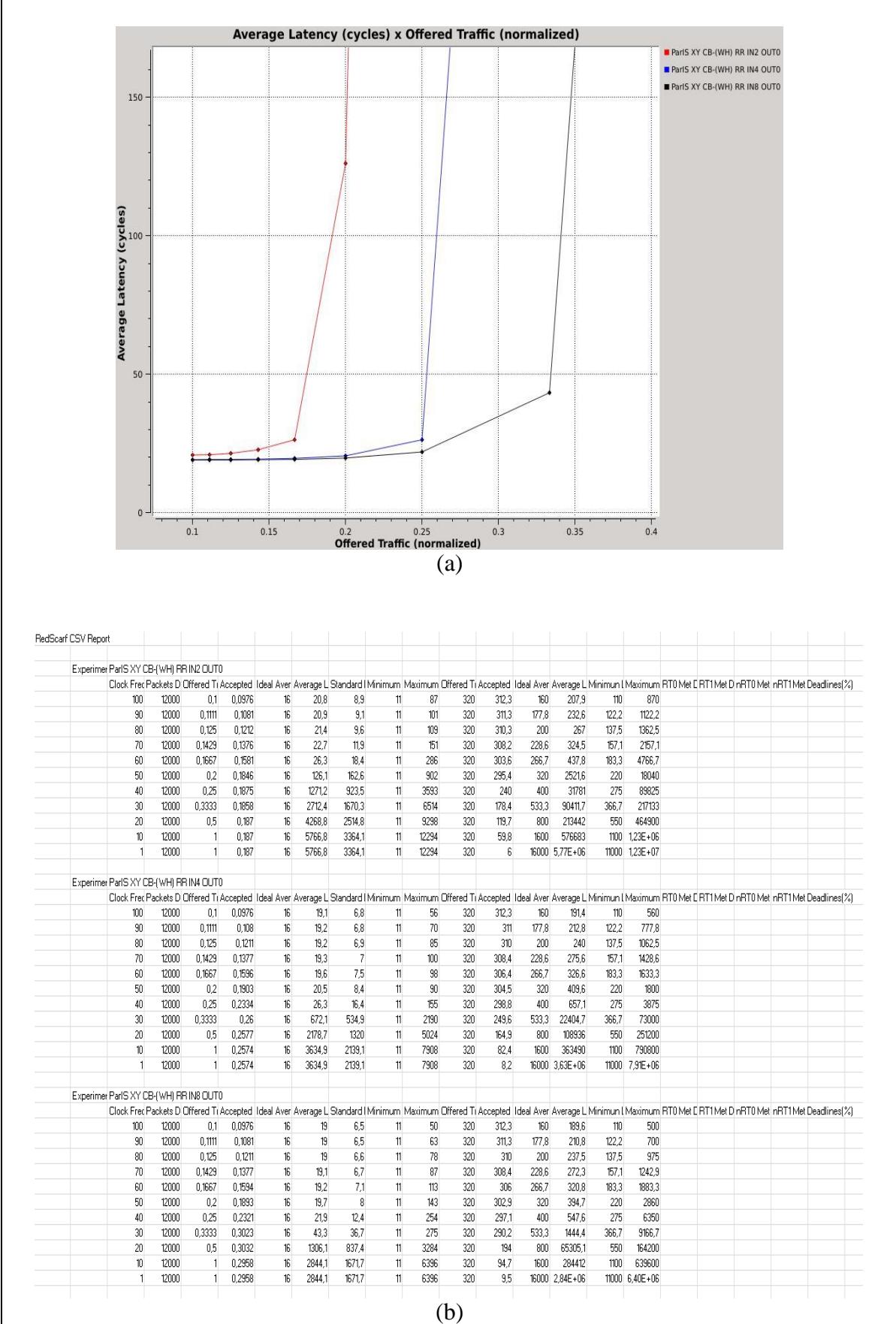


Figura 13. Resultados RedScarf: (a) exemplo de gráfico; (b) resultados exportados para o MS Excel

As características da ferramenta são apresentadas no Quadro 4, em que é realizado um comparativo entre as funcionalidades das duas versões da ferramenta. Observa-se que a nova versão agregou diversas funcionalidades novas.

Quadro 4.Comparativo das versões da ferramenta

Funcionalidade	BrownPepper	RedScarf
Configuração do dimensionamento da rede e do canal de comunicação	Sim	Sim
Definição de padrões de tráfego para cada nodo na rede	Sim	Sim
Visualização dos padrões de tráfego em forma de lista	Sim	Sim
Visualização dos padrões de tráfego em forma de árvore		Sim
Arquivamento da configuração de tráfego	Sim/txt	Sim/XML
Arquivamento da configuração dos experimentos e definições da simulação		Sim
Execução paralela das simulações (<i>multi-threading</i>)		Sim
Simulação baseada em tempo		Sim
Console integrado de mensagens		Sim
Análise de desempenho da simulação	Sim	Sim
Consideração dos <i>warm-up</i> e <i>drain</i>	Não usável	Usável
Possibilidade de definir a configuração padrão do ambiente		Sim
Verificação salvamento ao realizar alterações na configuração		Sim
Menu de configuração da ferramenta		Sim
Internacionalização		Sim
Carregamento e arquivamento dos resultados das simulações		Sim
Geração de relatório CSV		Sim
Documentação inclusa na ferramenta		Sim
Plotador de gráficos integrado		Sim
Visualizar distribuição de latência de todas as simulações de um experimento		Sim
Disponibilizar resultado das formas de ondas para todas as simulações		Sim
Documentação de software		Sim
Manual de usuário		Sim

3.6 FUNCIONALIDADES PARA A PRÓXIMA VERSÃO

Das funcionalidades já consideradas para as próximas versões tem-se:

- Definição de fluxos maliciosos para ataque as propriedades de segurança da rede: para verificar o impacto que fluxos maliciosos podem causar a rede e qual o comportamento da rede em situações anormais;
- Refinamento das simulações próximo aos pontos de saturação: para aprimorar e refinar a disponibilização de resultados mais precisos ao usuário nos principais pontos de verificação;

- Acompanhamento gráfico em tempo de simulação do estado da rede: para disponibilizar ao usuário uma forma visual de verificar a operação da rede ciclo-a-ciclo;
- Geração infinita de pacotes: para remover a limitação de definir um tempo de simulação maior do que a quantidade de pacotes disponíveis para serem entregues e a simulação continuar em execução mesmo que ociosa;
- Integração com uma linguagem de especificação e geração de modelos de NoCs: para facilitar a geração de diferentes modelos de NoCs;
- Estimativa de custo em área e potência em determinada tecnologia: ampliar a capacidade de análise da ferramenta e disponibilizar os custos da configuração;
- Análise do consumo de energia: ampliar a capacidade de análise e verificar o consumo para cada experimentação; e
- Mapeamento de tarefas: facilitar a configuração de tarefas e alocação destas em diferentes locais de rede para agilizar a buscar pelo melhor posicionamento das tarefas a serem transportadas pela rede.

4 CONCLUSÕES

Neste trabalho, foram realizados estudos sobre conceitos bases para execução da pesquisa proposta, incluindo: Redes-em-Chip, com foco em avaliação de desempenho, simuladores para Redes-em-Chip, a SoCIN, o *framework* Qt, a biblioteca SystemC e a ferramenta BrownPepper. Após os estudos foi realizado o projeto da nova versão do ambiente BrownPepper que foi rebatizado para RedScarf.

Todos os requisitos obrigatórios do projeto do RedScarf foram implementados e várias funcionalidades adicionais foram integradas a ferramenta em relação a versão anterior. Dentre elas destaca-se a aceleração da execução *multi-thread* que traz ganho de tempo nas experimentações e melhora a exploração do espaço de projeto permitindo o projetista verificar diferentes configurações em menor espaço de tempo.

A documentação criada também é destaque desta nova versão, pois permite que a ferramenta seja disponibilizada dando suporte aos usuários, com o manual da ferramenta, projetistas, com o documento de especificação de software, e até mesmo desenvolvedores, com documentação do código-fonte, que queiram dar continuidade no processo de desenvolvimento do ambiente de simulação da rede SoCIN.

Outra característica que marca positivamente a nova versão é a possibilidade de armazenar os resultados das simulações, a qual elimina a necessidade de executar e despender mais de uma vez o tempo de simulação para experimentos já realizados, ajudando também na exploração do espaço de projeto, para projetistas economizarem o tempo na busca pela melhor parametrização do sistema sem ter de executar várias vezes os mesmos experimentos. Outra possibilidade coberta por esta característica é a de diferentes grupos de pesquisas trocarem os resultados de suas experimentações para um objetivo em comum, além de permitir a criação de um portfólio de simulações.

Também é fruto deste trabalho a disponibilização da ferramenta para as principais plataformas desktop: Windows, Mac OS X e Linux, sendo que o único requisito é a biblioteca SystemC e um compilador compatível com tal. Essa característica que facilita a adesão da ferramenta sem a necessidade de montar um ambiente exclusivo para seu uso.

Um ponto importante na execução deste projeto foi à correção de alguns erros e inconsistências existentes na primeira versão da ferramenta. Como exemplo, pode ser citado o cálculo de latência para algumas configurações, que era feito erroneamente.

Vale lembrar que uma motivação para este trabalho era a necessidade de reestruturar a ferramenta para facilitar a manutenção e reusabilidade. Todo o código-fonte do *front-end* foi refeito com cerca de 16 mil linhas de código-fonte codificadas. Isso foi realizado no período de 2 anos, gerou retrabalho, mas a ferramenta possui um novo formato, paradigma e potencial de produtividade para pesquisas futuras.

Durante o andamento deste trabalho, foi publicado um artigo nos *proceedings* do 4th Workshop on Circuits and Systems Design – WCAS 2014 (SILVA; METZGER; ZEFERINO, 2014) que reporta o desenvolvimento do *front-end* em Qt e o ganho de desempenho com a execução *multi-thread*.

Espera-se que o produto deste esforço sirva de base para trabalhos futuros e que a evolução do ambiente atingido seja constante, para que o aperfeiçoamento deste, o torne referência em simulação na SoCIN e em Redes-em-Chip.

REFERÊNCIAS

ACCELLERA. **SystemC**. Disponível em:
<http://www.accelera.org/downloads/standards/systemc>. Acesso em: 02 Maio 2014.

ACHBALLAH, Ahmed Ben; SAoud, Slim Ben. A Survey of Network-On-Chip Tools. International Journal of Advanced Computer Science & Applications 4.9, 2013, p. 61-67.

ANDRIAHANTENAINA, A.; CHARLERY, H.; GREINER, A.; MORTIEZ, L.; ZEFERINO, C. A. SPIN: a scalable, packet switched on-chip micro-network. In: DESIGN, AUTOMATION AND TEST ON EUROPE (DATE), 2003, Munich. **Proceedings...** Los Alamitos: IEEE Computer Society, 2003. p. 70-73.

Arteris. **Arteris FlexNoC Interconnect IP**. Disponível em:
<http://www.arteris.com/flexnoc>. Acesso em: 07 Maio 2014.

Atlas. **Atlas**: An Environment for NoC Generation and Evaluation. Disponível em:
<https://corfu.pucrs.br/redmine/projects/atlas>. Acesso em: 07 Maio 2014.

AYALA, Jose L. **Communication architectures for systems-on-chip**. Hoboken: Taylor & Francis Group, 2011.

BENINI, Luca; DE MICHELI, Giovanni. **Networks on chips**: technology and tools. San Francisco: Elsevier, 2006.

BLANCHETTE, Jasmin; SUMMERFIELD, Mark. **C++ GUI programming with Qt4**. Stoughton: Prentice Hall, 2006.

BookSim. **BookSim 2.0 User's Guide**. 2013 Disponível em: <<http://nocs.stanford.edu/cgi-bin/trac.cgi/wiki/Resources/BookSim>>. Acesso em: 07 Maio 2014.

CHAN, Jeremy; PARAMESWARAN, Sri. NoCGEN: A Template Based Reuse Methodology for Networks on Chip Architecture. In: VERY LARGE SCALE INTEGRATION DESIGN (VLSID'04), 2004, [S.I.]. **Proceedings...** IEEE, 2004, p. 717-720.

DALLY, William; TOWLES, Brian. **Principles and Practices of Interconnection Networks**. San Francisco: Morgan Kaufmann, 2004.

DUATO, José; YALAMANCHILI, Sudhakar; NI, Lionel. **Interconnection Networks**: an engineering approach. San Francisco: Morgan Kaufmann Pub, 2003

GEBALI, Fayez; ELMILIGI, Haytham; EL-KHARASHI, Mohamed W. **Networks-on-Chip**: theory and practice. New York: CRC Press, 2009.

GOTTSCHLING, Philip; YING, Haoyuan; HOFMANN, Klaus. GSNOc UI – A Comfortable Graphical User Interface for Advanced Design and Evaluation of 3-Dimensional Scalable Networks-on-Chip. In: HIGH PERFORMANCE COMPUTING AND SIMULATION (HPCS), 2012, Madrid. **Proceedings...** IEEE, 2012, p. 261-267.

GUERRIER, Pierre; GREINER, Alain. A Scalable Architecture for System-On-Chip Interconnections. In: Sophia-Antipolis MICRO-ELETROONICS CONFERENCE (SAME), 1999, France. **Proceedings...** Sophia Antipolis: [s.n.], 1999. p. 90-93.

HEMANI, Ahmed; JANTSCH, Axel; KUMAR, Shashi; POSTULA, Adam; OBERG, Johnny; MILLBERG, Mikael; LINDQVIST, Dan. Network on chip: An architecture for billion transistor era. In: IEEE NORCHIP CONFERENCE, 2000, Turku. **Proceedings...** IEEE NorChip, vol. 31. 2000, p. 1-8.

IEEE Computer Society. **SystemC® Language Reference Manual**. 2012. 614p. Disponível em: <<https://standards.ieee.org/findstds/standard/1666-2011.html>>. Acesso em: 02 Maio 2014.

JANTSCH, Axel; TENHUNEN, Hannu. **Networks on Chip**. Dordrecht: Kluwer Academic Pub., 2003.

JERGER, N. E.; PEH, L. S. **On-Chip Networks**. Madison: Morgan e Claypool, 2009.

NIRGAM. **NIRGAM**: A Simulator for NoC Interconnect Routing and Application Modeling. 2007. Disponível em: <<http://nirgam.ecs.soton.ac.uk/>>. Acesso em: 07 Maio 2014.

NNSE. **NNSE**: The Nostrum NoC Simulation Environment. Disponível em: <<http://www.ict.kth.se/nostrum/NNSE/>>. Acesso em: 07 Maio 2014.

NoC Simulator. **NoC Simulator**. 2007. Disponível em: <<http://nocsim.blogspot.com.br/>>. Acesso em: 07 Maio 2014.

Noxim. **Noxim**: the NoC Simulator. Disponível em: <<http://noxim.sourceforge.net/>>. Acesso em: 07 Maio 2014.

OCCN. **OCCN**: On-Chip Communication Architecture. 2011. Disponível em: <<http://occn.sourceforge.net/>>. Acesso em: 07 Maio 2014.

OGRAS, Umit Y.; MARCULESCU, Radu. **Modeling, Analysis and Optimization of Network-on-Chip Communication Architectures**. Pittsburgh: Springer, 2013.

OpenGL. **OpenGL**: The Industry's Foundation for High Performance Graphics. Disponível em: <www.opengl.org>. Acesso em: 08 Maio 2014.

PASRICHA, Sudeep; DUTT, Nikil. **On-chip communication architectures**: system on chip interconnect. Burlington: Morgan Kaufmann Pub, 2008.

PATTERSON, David A.; HENNESSY, John L. **Organização e projeto de computadores**: interface hardware/software. Tradução de Daniel Vieira. 4 ed. Rio de Janeiro: Elsevier, 2014.

Qt Digia. **Qt Digia**. Disponível em: <<http://qt.digia.com/>>. Acesso em: 30 Abr. 2014.

Qt Project. **Documentation**. Disponível em: <<http://qt-project.org/doc/>> . Acesso em: 30 Abr. 2014.

SILVA, Eduardo Alves da; METZGER, Luiz Gustavo; ZEFERINO, Cesar Albenes. On the development of a Qt-based multithread NoC simulator. In: WORKSHOP ON CIRCUITS

AND SYSTEMS DESIGN (WCAS 2014), 4., 2014, Aracajú. **Proceedings....** Aracajú: UFS, 2014. p. 1-4.

TEWKSBURY, S. K., UPPULURI, M., HORNAK, L. A. Interconnections/Micro-Networks for Integrated Microelectronics. In: IEEE Global Telecommunications Conference, Orlando, 1992. **Proceedings...** Los Alamitos, IEEE Computer Society Press (1992) 180–186.

TONG, Ian G.; ANDERSON, Ian D. L.; KHALID, Mohammed A. S. Soft-Core Processors for Embedded Systems. In: INTERNATIONAL CONFERENCE MICROELETRONICS (ICM'06), 18., 2006, Dharan. **Proceedings...** ICM, 2006, p. 170-173.

VAHID, Frank. **Sistemas Digitais:** projeto, otimização e HDLs. Tradução de Anatólio Laschuk. Porto Alegre: Artmed, 2008.

ZEFERINO, C. A. **Redes-em-Chip:** Arquiteturas e modelos para avaliação de área e desempenho. 2003. 242p. Tese (Doutorado) – Programa de Pós Graduação em Computação, Universidade Federal do Rio Grande do Sul, Porto Alegre. 2003.

ZEFERINO, C. A.; SANTO, Frederico G. M. E.; SUSIN, A. ParIS: A Parameterizable Interconnect Switch for Network-on-Chip. In: SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEM DESIGN, 2004, Pernambuco. **Proceedings...** New York: ACM, 2004, p. 204-209.

ZEFERINO, C. A.; SUSIN, Altamiro. A. SoCIN: a parametric and scalable network-on-chip. In: SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN, 16th., 2003, São Paulo. **Proceedings...** Los Alamitos: IEEE Computer Society, 2003. p. 169-174.

ZEFERINO, Cesar A.; BRUCH, Jaison. V.; PIZZONI, Magnos. R. BrownPepper: a SystemC-based Simulator for Performance Evaluation of Networks-on-Chip. In: International Conference On Very Large Scale Integration (VLSI-SOC), 17., 2009, Florianópolis. **Proceedings...** Piscataway: IEEE Computer Society, 2009, p. 223-226.

ZEFERINO, Cesar Albenes; BRUCH, Jaison Valmor; PEREIRA, Thiago Felski KREUTZ, M. E.; SUSIN, A. A. Avaliação de desempenho de Rede-em-Chip modelada em SystemC. In: WORKSHOP DE DESEMPENHO DE SISTEMAS COMPUTACIONAIS E DE COMUNICAÇÃO (WPERFORMANCE 2007), 2007, Rio de Janeiro. **Anais do Congresso da Sociedade Brasileira de Computação.** Porto Alegre: Sociedade Brasileira de Computação, 2007. p. 559-578.

APÊNDICE A. DETALHAMENTO DAS PROPRIEDADES DAS REDES-EM-CHIP

Nas seções deste apêndice são apresentados os detalhamentos sobre as propriedades das Redes-em-Chip, que são: topologia, roteamento, controle de fluxo e microarquitetura dos roteadores.

A.1 TOPOLOGIA

As topologias das Redes-em-Chip podem ser classificadas em diretas e indiretas. Nas topologias diretas cada nodo terminal é associado com um roteador, assim todos os roteadores são fontes e destinos de tráfego. Nas topologias indiretas, roteadores são distintos de nodos terminais. Somente nodos terminais são fonte e destino de tráfego, enquanto os nodos intermediários simplesmente realizam o chaveamento do tráfego entre os nodos terminais (AYALA, 2011). Para exemplificar, algumas topologias diretas são ilustradas na Figura 14, e topologias indiretas na Figura 15(a seguir).

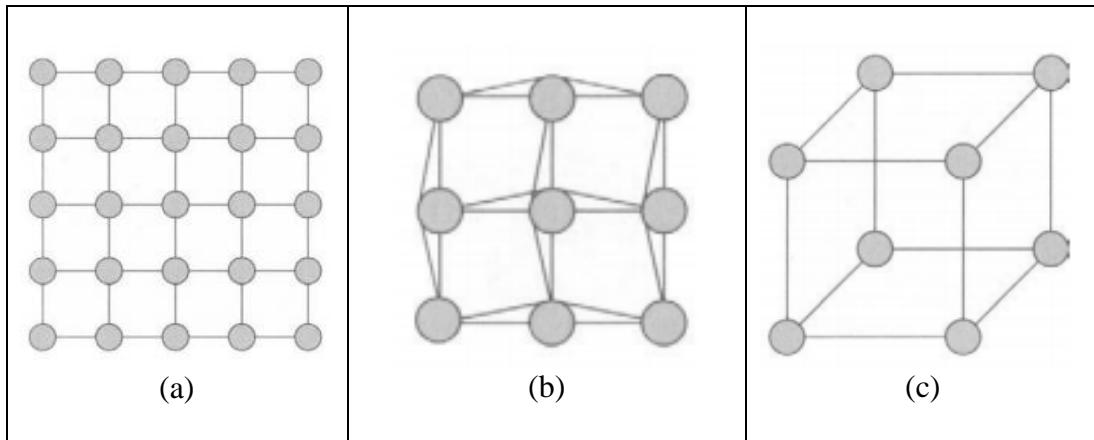


Figura 14. Topologias diretas: (a) malha, (b) toróide e (c) hipercubo

Fonte: Benini e De Micheli (2006).

Na Figura 14, cada círculo representa um nodo na rede, composto pelo núcleo e pelo roteador, já que são fontes e destinos de tráfego. As linhas representam as ligações entre os roteadores, ou seja, os enlaces. Nas topologias toróide e hipercubo a uniformidade está na quantidade de porta dos roteadores que é a mesma para todos os nodo da rede e na malha apenas há variação na quantidade de portas dos roteadores de borda da rede.

Na Figura 15, cada quadrado representa um nodo, sendo que os quadrados preenchidos (tom escuro) são os comutadores que somente transferem mensagens entre os terminais de

processamento, e os quadrados não preenchidos são os nodos terminais os quais estão conectados à um núcleo emissor ou receptor de tráfego.

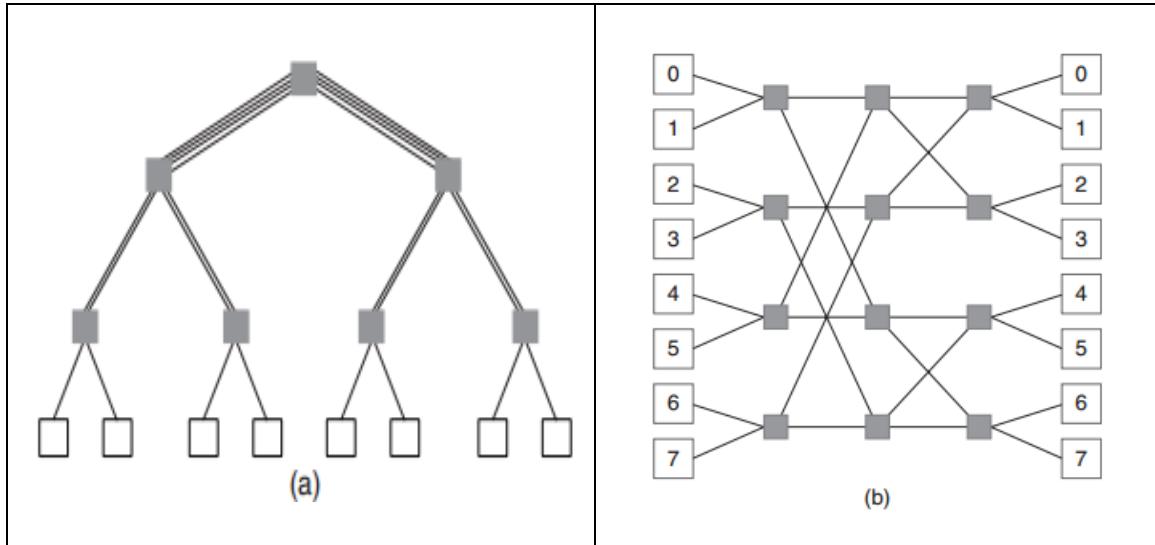


Figura 15. Topologias Indiretas: (a) árvore gorda e (b) *butterfly*

Fonte: Pasricha e Dutt (2008).

Uma propriedade importante das topologias diretas é que à medida que o número de nodos no sistema aumenta a largura de banda da comunicação também aumenta. Os principais agravantes destas topologias estão na conectividade e no custo que aumentam à medida que o tamanho da rede aumenta. Alta conectividade resulta em alto desempenho, mas aumenta o consumo de energia e o custo na implementação dos canais dos roteadores (PASRICHA; DUTT, 2008).

Também existem as topologias irregulares, que utilizam uma abordagem híbrida das topologias direta e indireta, mas são tipicamente utilizadas em aplicações específicas, diminuindo, portanto, a escalabilidade (PASRICHA; DUTT, 2008).

As topologias mais comuns nos projetos de NoCs são a malha e o toróide, que constituem acima de 60% dos casos (AYALA, 2011).

A.2 ROTEAMENTO

As arquiteturas das Redes-em-Chip são baseadas nas redes que realizam chaveamento por pacote. Os roteadores podem implementar várias funcionalidades, de um simples chaveamento à um roteamento inteligente. O roteamento nas NoCs é muito semelhante ao

roteamento em qualquer rede. Um algoritmo de roteamento determina como os dados serão transmitidos do emissor ao receptor (AYALA, 2011).

Os algoritmos de roteamento são responsáveis por rotear os pacotes ou circuitos corretamente e de forma eficiente da fonte ao destino. Podem ser classificados de diferentes formas. Em relação ao momento da realização do roteamento, pode ser classificado como dinâmico (em tempo de execução) ou estático (em tempo de compilação). Quanto às decisões de roteamento, fonte (definido no nodo emissor) ou distribuído (definido nos roteadores). E no que tange a adaptatividade, pode ser determinístico (caminho fixo e mínimo) e adaptativo (caminho variável podendo ser não mínimo) (PASRICHA; DUTT, 2008; JERGER; PEH, 2009; OGRAS; MARCULESCU, 2013; ZEFERINO, 2003).

Sobre os algoritmos de roteamento pode-se afirmar que são os responsáveis por garantir que uma mensagem seja transmitida da fonte ao destino, evitar congestionamentos de acordo com o tipo de algoritmo utilizado, evitar *deadlocks*, manter o consumo de energia uniforme sobre os roteadores, prover tolerância à faltas, entre outros (OGRAS; MARCULESCU, 2013).

A complexidade de implementação e os requisitos de desempenho são os dois maiores conceitos envolvidos na escolha da estratégia de roteamento. Enquanto o roteamento adaptativo provê melhor vazão e baixa latência por permitir caminhos alternativos baseados na congestão da rede, algoritmos determinísticos garantem que não haverá pontos de *deadlock* e *livelock* na rede (OGRAS; MARCULESCU, 2013).

Um exemplo de algoritmo de roteamento determinístico que é muito comum, é o roteamento ordenado por dimensão para redes com topologia em malha 2D (DUATO; YALAMANCHILI; NI, 2003), conhecido como XY. Esse roteamento consiste em realizar os saltos na rede primeiro na dimensão X e depois na dimensão Y. Considerando o exemplo da Figura 16, em uma NoC em malha 2D de tamanho “3,3” para o roteamento XY, supondo que deseja-se enviar uma informação do roteador “0,0” para o “2,2”, o caminho a ser percorrido pela mensagem será o ilustrado e com os passos enumerados na Figura 16(b).

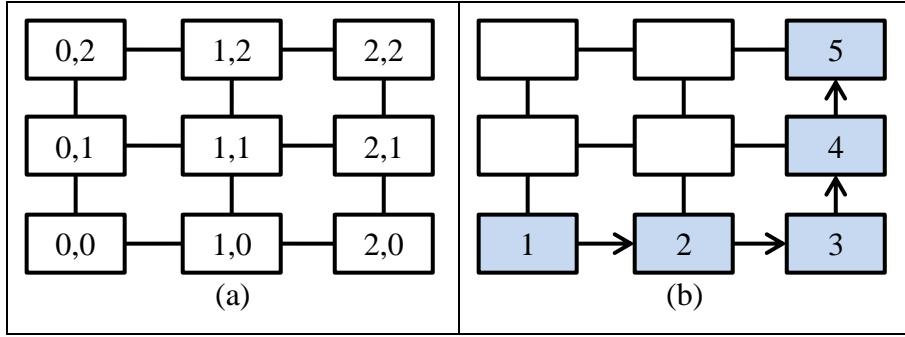


Figura 16. Roteamento numa malha 2D 3x3: (a) malha com o posicionamento dos nodos e (b) caminho percorrido no algoritmo XY com os passos enumerados do nodo “0,0” para o “2,2”

Os quadrados na Figura 16 representam os roteadores, enquanto as linhas representam os enlaces. O caminho do roteamento XY é o caminho mais curto, com o menor número de saltos do nodo de origem até o nodo de destino, passando por cinco roteadores (da fonte em 1 até o destino em 5), e 4 enlaces ($1 \rightarrow 2$; $2 \rightarrow 3$; $3 \rightarrow 4$; $4 \rightarrow 5$). O pacote percorre a rede primeiramente no eixo X e depois no eixo Y.

Com relação à caracterização, o algoritmo de roteamento XY pode ser classificado como roteamento determinístico (estático, de uma fonte para um destino sempre será feito o mesmo caminho) e de caminho mínimo (sempre a menor distância). Conforme a implementação, a definição da rota pode ser feita no nodo origem (fonte) ou nos roteadores (distribuído). Uma desvantagem do roteamento XY é quando há congestão em um caminho, pois nenhum pacote que depende deste caminho pode ser transferido.

A.3 CONTROLE DE FLUXO

O objetivo do controle de fluxo é alocar (e desalocar) os recursos da rede para atravessar uma mensagem pela NoC. É o controle de fluxo que determina como os recursos são alocados, tais como largura de banda do canal, capacidade dos *buffers*, estado de controle. Ele determina quando *buffers* e enlaces são atribuídos para as mensagens, a granularidade em que são alocados, e como estes recursos são compartilhados por várias mensagens na rede. Um protocolo de controle de fluxo bem projetado diminui a latência na entrega das mensagens, diminuindo a carga sem aumentar a sobrecarga na alocação dos recursos, e aumenta a vazão por permitir um efetivo compartilhamento dos *buffers* e enlaces sobre as mensagens (JERGER; PEH, 2009; AYALA, 2011).

Quando uma mensagem é injetada na rede, ela é segmentada em pacotes, que podem ser divididos em *flits* de tamanho fixo. Os *flits* ainda podem ser divididos em *phits* (physical units – unidades físicas), que correspondem à largura física do canal de comunicação. A Figura 17 demonstra a composição de uma mensagem e sua segmentação (JERGER; PEH, 2009).

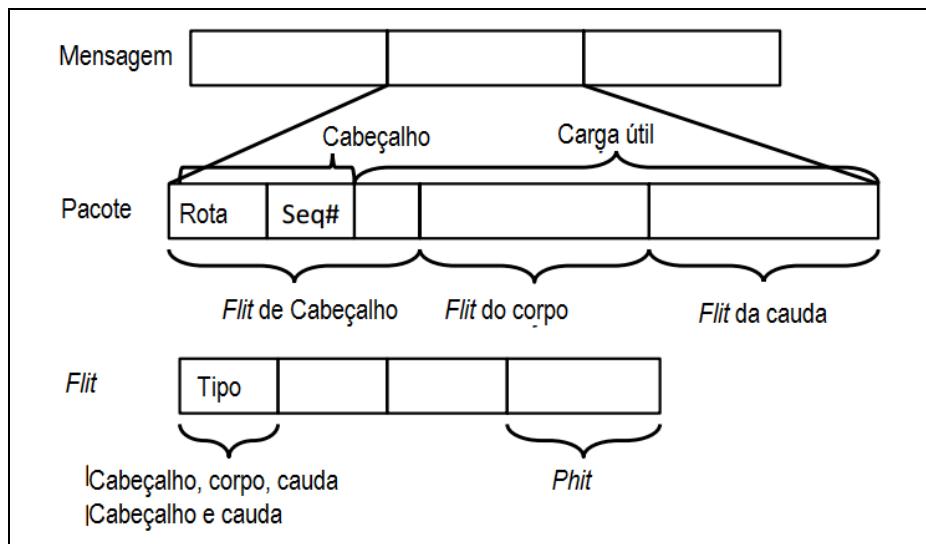


Figura 17. Segmentação de uma mensagem em pacote, divisão do pacote em *flits* e divisão do *flit* em *phits*

Fonte: Adaptado de Jerger e Peh (2009).

Na Figura 17, é apresentada uma mensagem dividida em pacotes, um pacote em *flits* e *flit* em *phits* (menor unidade de transferência). No pacote informações como rota e número de sequência ficam no *flit* de cabeçalho. E para cada *flit* é identificado seu tipo, que pode ser cabeçalho, corpo, cauda.

O controle de fluxo é projetado considerando a granularidade das informações, em nível de mensagem, pacote ou *flit*, pois os *buffers* e unidades internas do roteador serão projetados e controlados com base nessas informações (PASRICHA; DUTT, 2008).

Os esquemas de controle de fluxo oferecem diferentes características de implementação que impactam diretamente no consumo de energia, área de silício e desempenho (PASRICHA; DUTT, 2008).

Para reduzir os requisitos de memorização e diminuir o custo, em NoCs, são utilizados controles de fluxo baseados em *flits*. Sendo que, quanto maior a granularidade (maior

subdivisão da mensagem e unidades de transferências menores), menores são os *buffers*, facilitando o projeto dos roteadores e reduzindo a área de silício.

Um exemplo de controle de fluxo utilizado em Redes-em-Chip é o baseado em créditos, que possui um contador do espaço disponível nos *buffers*, sendo que o emissor pode enviar o número de *flits* equivalente ao espaço disponível no *buffer* do receptor (neste caso quando o chaveamento for do tipo *wormhole*) (GEBALI; ELMILIGI; AL-KHARASHI, 2009).

A.4 MICROARQUITETURA DO ROTEADOR

O projeto do roteador envolve a determinação das técnicas de controle de fluxo, a definição dos *buffers* de entrada-e-saída, a matriz de conexão (referenciado como *crossbar* na literatura) e a lógica de controle. Os roteadores devem ser projetados para atender os requisitos de latência e vazão na estreita área de silício e as restrições quanto ao consumo de energia (OGRAS; MARCULESCU, 2013; AYALA, 2011; JERGER; PEH, 2009).

O roteador é o principal componente na arquitetura de comunicação, sendo que seu trabalho é transferir uma informação de uma porta de entrada para uma ou mais portas de saída. A lacuna de tempo entre o momento em que a informação chega à porta de entrada do roteador e é encaminhado por uma das portas de saída é denominado de atraso do roteador (JANTSCH; TENHUNEN, 2003).

Segundo Gebali, Elmiligi e Al-Kharashi (2009), o encaminhamento dos pacotes no roteador, consiste de quatros estágios, são eles: (*i*) escolha da rota: obter as informações do pacote para selecionar a porta de saída; (*ii*) arbitragem: devido à possibilidade de múltiplas requisições pelo mesmo recurso no roteador (ex. múltiplas requisições na mesma porta de saída), é necessário um árbitro para definir qual requisição ganha o recurso primeiro, sendo que o árbitro mantém o recurso alocado até que a transferência de toda a informação seja feita no roteador; (*iii*) chaveamento: com a rota definida e a decisão de arbitragem tomada, o roteador encaminha cada palavra do pacote de sua porta de entrada para a porta de saída; e (*iv*) liberação do árbitro: após a última palavra ser encaminhada pelo *crossbar*, o árbitro libera os recursos da comunicação deixando disponível para outros pacotes que estejam esperando na porta de entrada.

O roteador é caracterizado então pelas portas de entrada, *buffers* de entrada, *crossbar*, controle de chaveamento, *buffers* de saída (opcional) e portas de saída. A Figura 18 apresenta de forma generalizada a estrutura de um roteador, indicando localmente os *buffers* de entrada e saída, um bloco de controle e o *crossbar* representado pelo bloco com “X”. A quantidade de canais de comunicação varia de implementação e em alguns casos de acordo com a topologia e localização do roteador na rede.

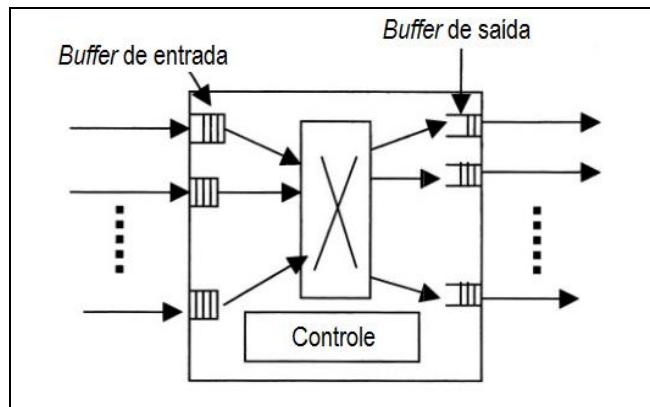


Figura 18. Estrutura geral de um roteador

Fonte: Adaptado de Jantsch e Tenhunen (2003).

As formas de *buffers* comumente encontradas são do tipo FIFO (First-In-First-Out). O atraso do roteador ocorre devido aos ciclos gastos para a realização das operações desempenhadas por tal, que de forma geral são: (i) escrita no *buffer* de entrada; (ii) roteamento; (iii) arbitragem; (iv) encaminhamento; e/ou (v) escrita no *buffer* de saída.

Um exemplo de arbitragem utilizada é a circular ou Round-Robin, em que a última requisição que foi atendida possuirá a menor prioridade na próxima arbitragem, tentando balancear de forma justa a utilização dos recursos (JERGER; PEH, 2009).

APÊNDICE B.PROJETO DE SOFTWARE

Nesta sessão será apresentado o projeto completo de software realizado até o momento. Nele estão contidos os requisitos funcionais, não funcionais, regras de negócio, casos de uso, diagrama de classes, diagrama de atividades e protótipos de tela.

Serão apresentadas apenas as imagens obtidas da ferramenta CASE de modelagem, sendo que o detalhamento textual será realizado na conclusão do projeto.

B.1 REQUISITOS FUNCIONAIS

A lista dos requisitos funcionais da aplicação é exibida na Figura 19, que contempla todas as funcionalidades presentes na versão proposta.



Figura 19. Diagrama dos Requisitos Funcionais

B.2 REQUISITOS NÃO FUNCIONAIS

O diagrama dos requisitos não funcionais da aplicação é apresentado na Figura 20.

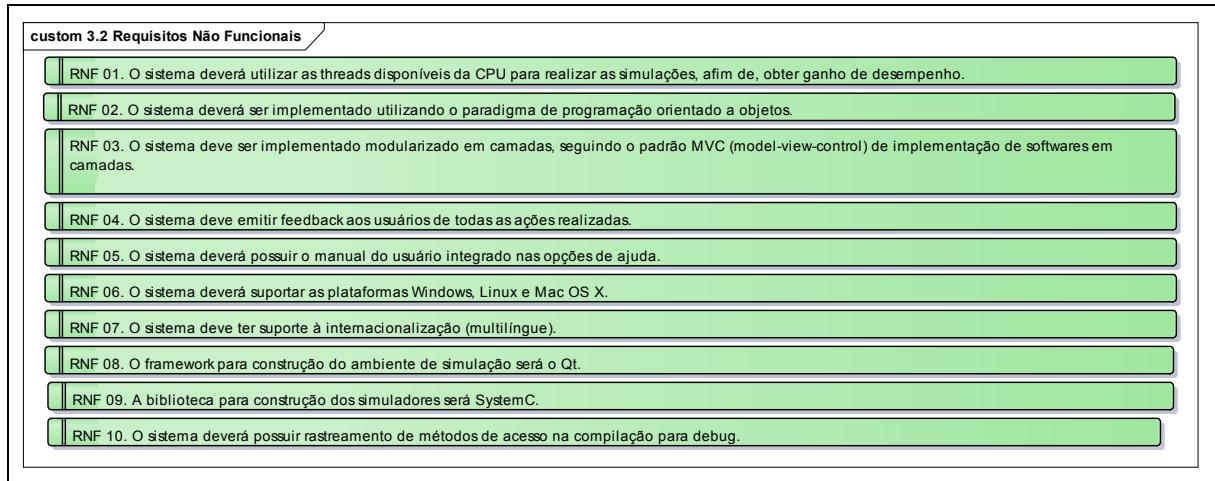


Figura 20. Diagrama dos Requisitos Não Funcionais

B.3 REGRAS DE NEGÓCIO

O diagrama das regras de negócio da aplicação, que estão diretamente relacionadas com os requisitos funcionais, é apresentado na Figura 21.



Figura 21. Diagrama das Regras de Negócio

B.4 DIAGRAMA DE CLASSES

O diagrama de classes apresentado é apenas uma visão global das classes, pois devido ao nível de detalhamento, não fica visível o diagrama completo (Figura 22).

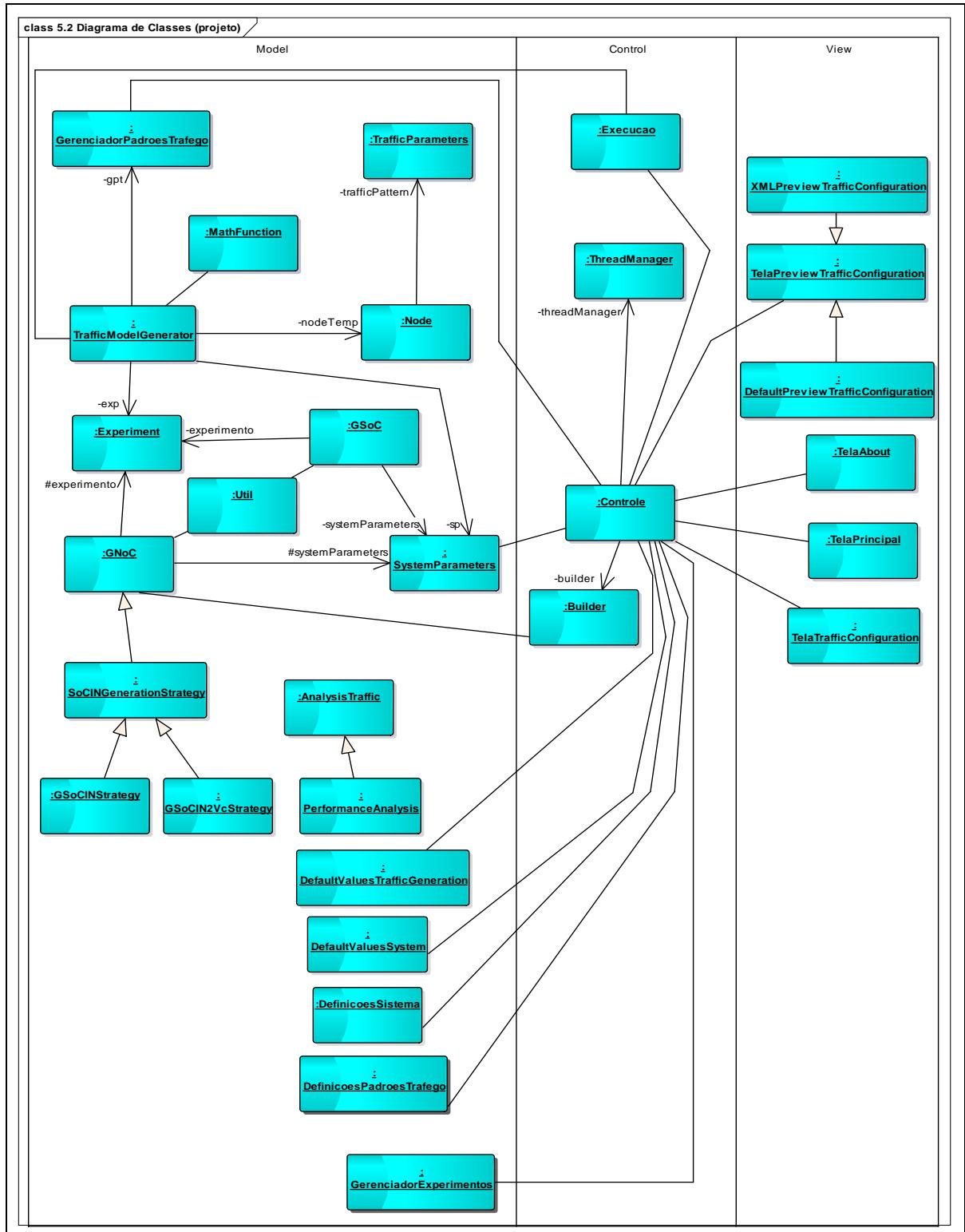


Figura 22. Diagrama de Classes (visão macro)

B.5 DIAGRAMA DE CASOS DE USO

O diagrama de casos de uso (Figura 23) da aplicação apresenta os pontos de interação com as funcionalidades do sistema, bem como os atores que realizam as interações.

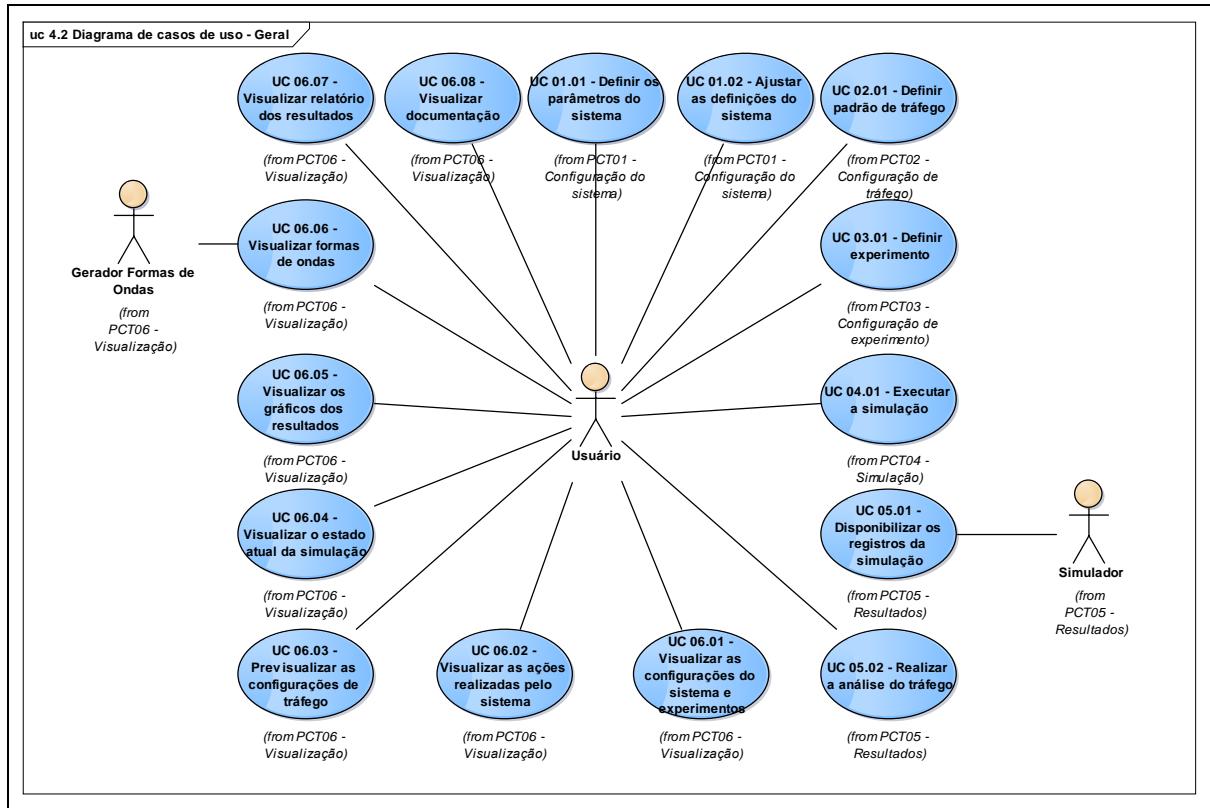


Figura 23. Diagrama de Casos de Usos

B.6 DIAGRAMA DE ATIVIDADES

O diagrama de atividades (Figura 24) mostra o fluxo de utilização da aplicação proposta.

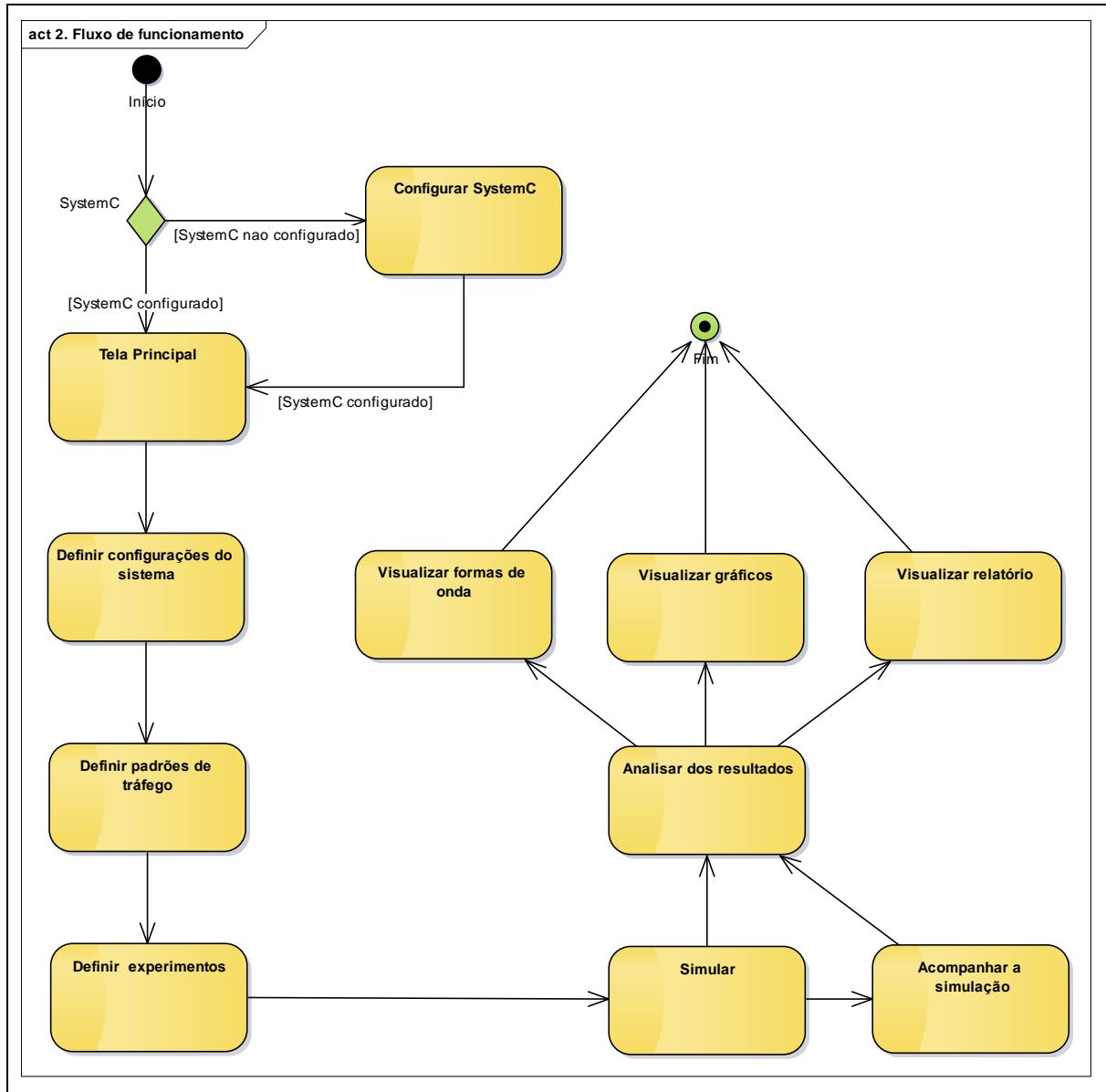


Figura 24. Diagrama de Atividades

B.7 TELAS

B.7.1 Telas de configuração do sistema

Na Figura 25, são apresentadas as telas de configuração da rede, e configuração das preferências do ambiente.

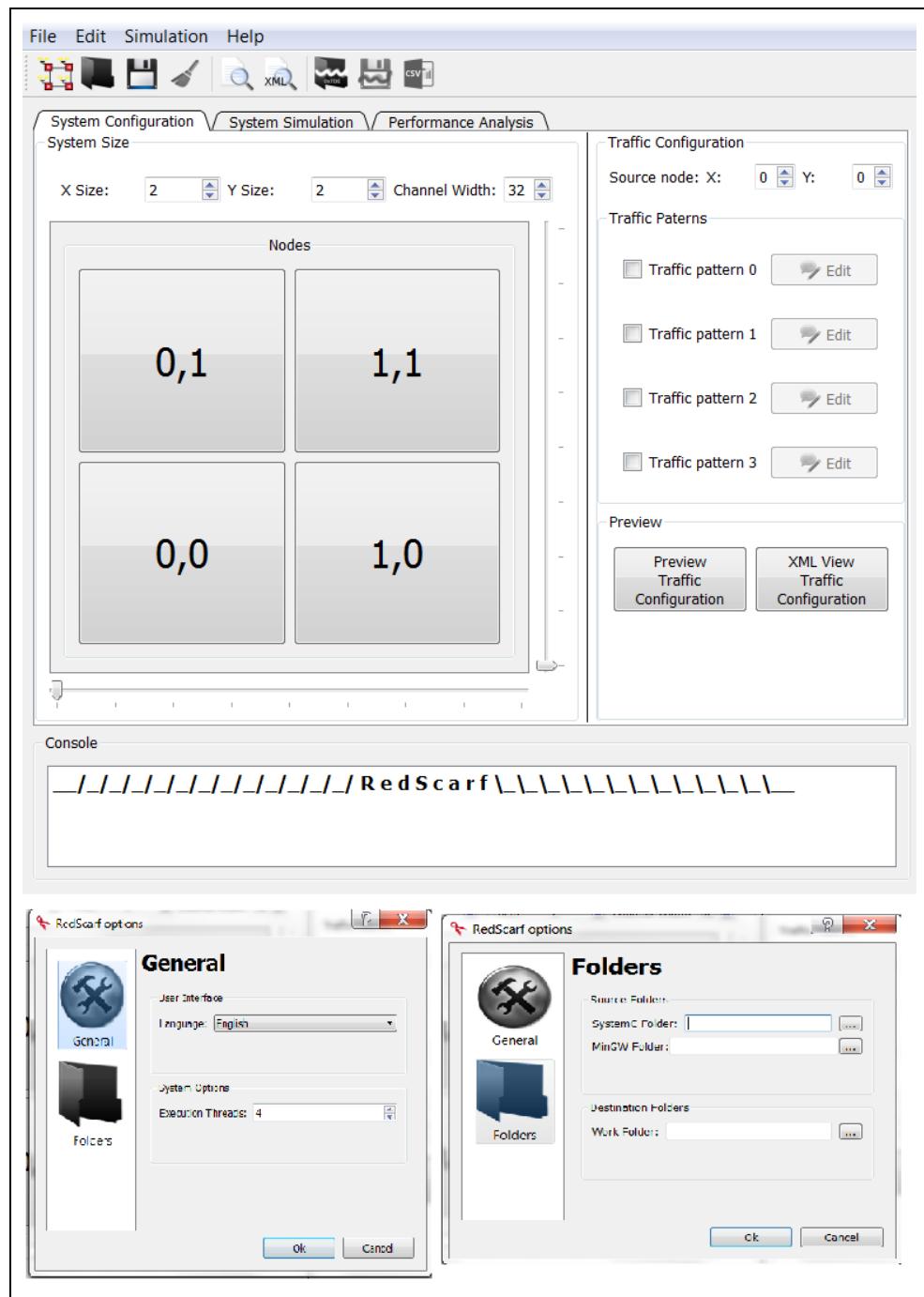


Figura 25. Telas de configurações do sistema

B.7.2 Tela de simulação do sistema

A tela de simulação do sistema (Figura 26) ilustra as configurações dos experimentos a serem realizados. De acordo com a configuração dos experimentos é definida a quantidade de simuladores a serem executados.

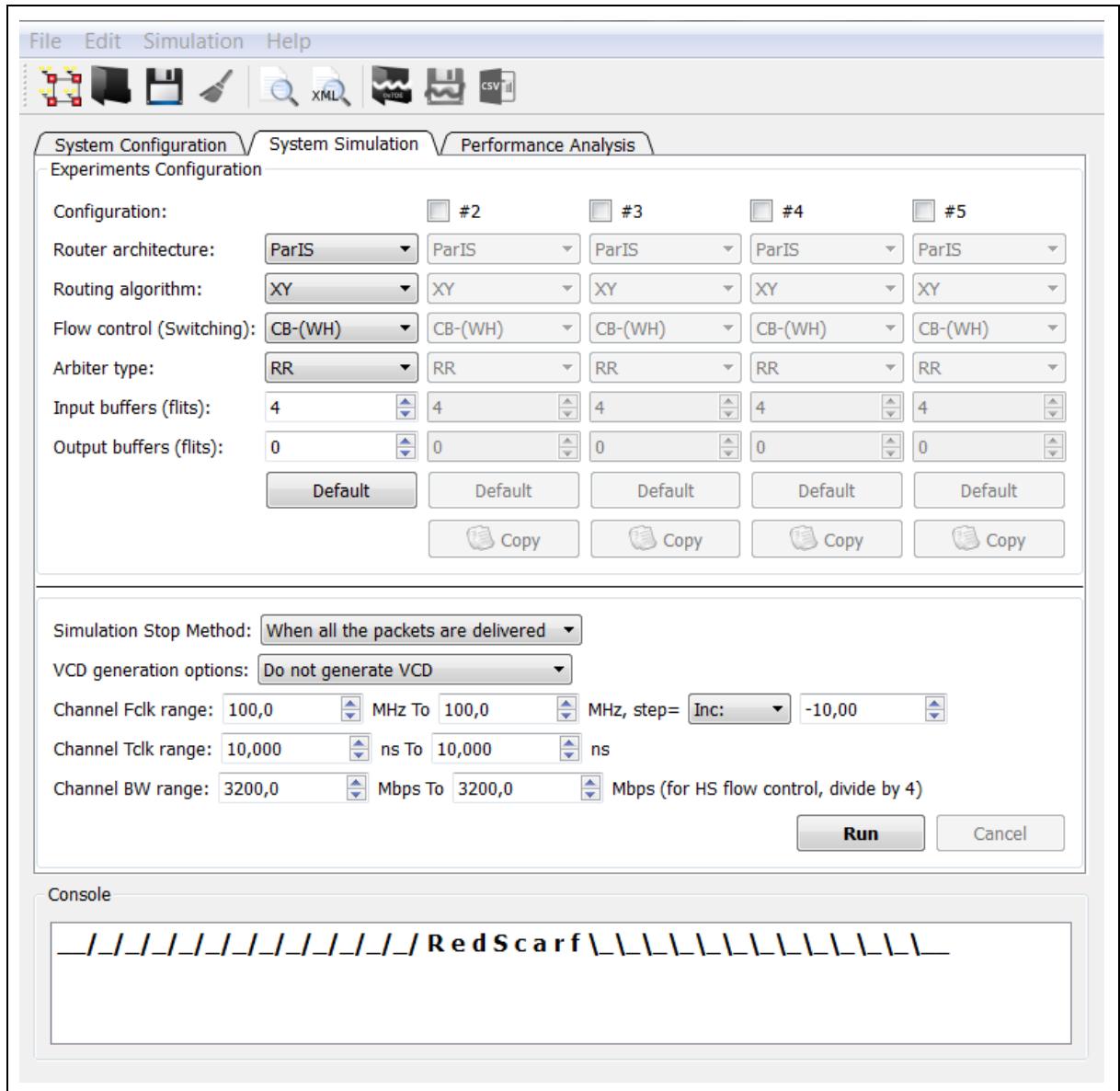


Figura 26. Tela de simulação do sistema

B.7.3 Tela de análise de desempenho

Na tela de análise de desempenho (Figura 27) é possível determinar o intervalo de análise de desempenho, gerar o relatório, formas de onda e gráficos a serem visualizados, bem como a definição dos parâmetros dos gráficos.

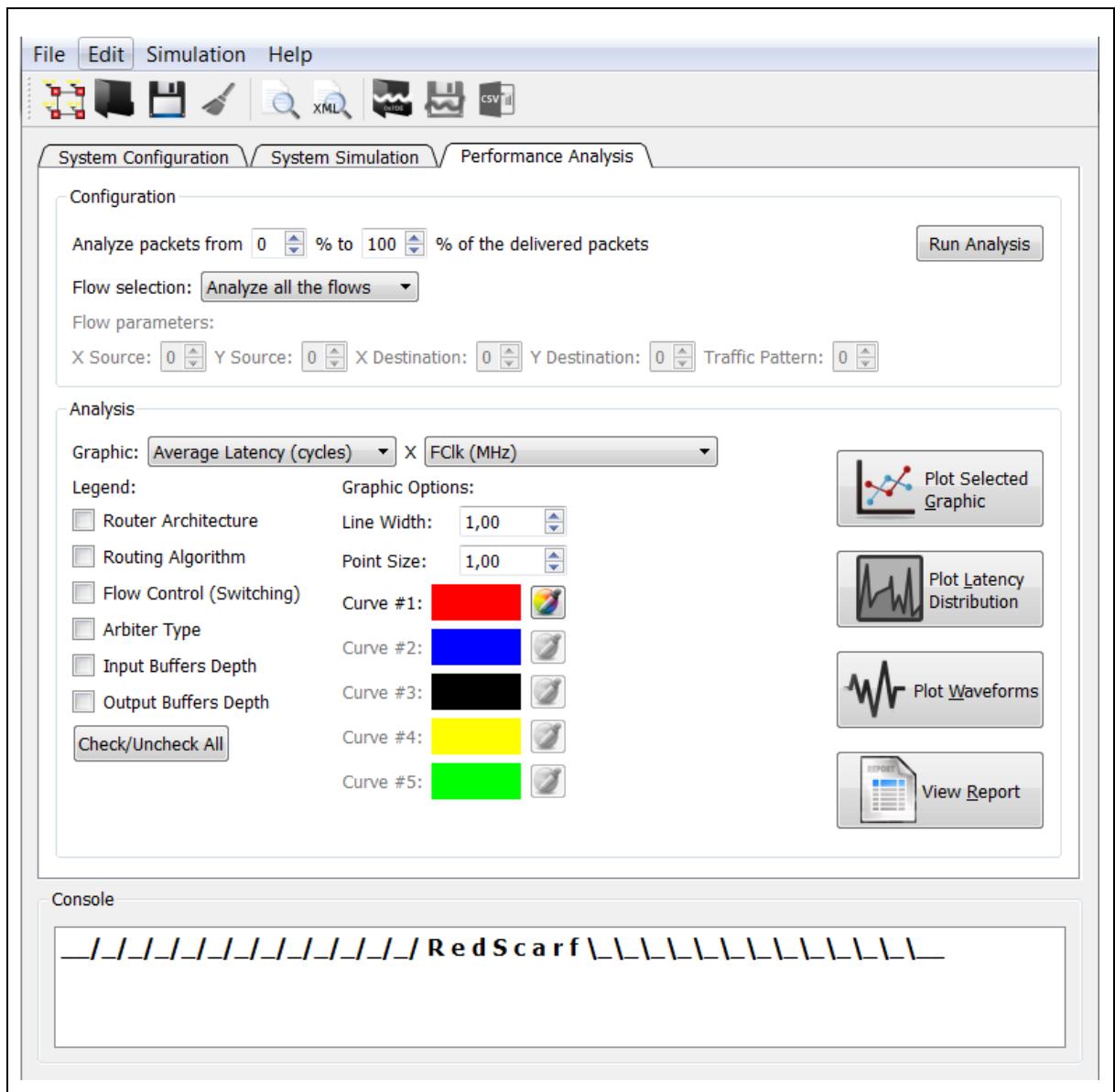


Figura 27. Tela de análise de desempenho

B.7.4 Telas de previsualização de tráfego

Nas telas de pré-visualização de tráfego (Figura 28) é possível visualizar as configurações de tráfego configuradas. Estão disponíveis dois formatos de pré-visualização, formato de relatório e visão em árvore.

The screenshot shows the 'ui TEL004 - Visualização de tráfego' application interface. It features two main windows:

- TEL004.01:** A table view showing traffic configurations. The columns include: Source Node, Destination Node, Traffic Class, Injection Type, Switching Type, Packets to send, Deadline (ns), Required Bandwidth (Mbps), Message Size (bits), Idle Time (ns), Message Interval (ns), Probability Function, and Standard Deviation (% Req. BW). The data rows are:

(0,0); 0	Uniform	RT0 - Signalling	Constant	Wormho...	100	0	100	128				
(0,1); 0	Uniform	RT0 - Signalling	Constant	Wormho...	100	0	100	128				
(1,0); 0	Uniform	RT0 - Signalling	Constant	Wormho...	100	0	100	128				
(1,1); 0	Uniform	RT0 - Signalling	Constant	Wormho...	100	0	100	128				
- TEL004.02:** A property editor view showing traffic configuration details. The properties listed are:

Property	Value
Source Node	X: 0 Y: 0
Traffic Pattern	0
Spatial Distribution	Uniform
Traffic Class	RT0 - Signalling
Type Injection	Constant
Switching Technique	Wormhole Switching
Number Packets Per Flow	100
Deadline	0
Required Bandwidth	100
Message size	128
Source Node	X: 0 Y: 1
Traffic Pattern	0
Spatial Distribution	Uniform
Traffic Class	RT0 - Signalling
Type Injection	Constant
Switching Technique	Wormhole Switching
Number Packets Per Flow	100
Deadline	0
Required Bandwidth	100
Message size	128
Source Node	X: 1 Y: 0
Source Node	X: 1 Y: 1

Figura 28. Telas de previsualização de tráfego

B.7.5 Tela de configuração de tráfego

A tela de configuração de tráfego (Figura 29) permite o usuário definir e editar todos os parâmetros necessários para a geração de fluxo(s) de tráfego(s). Na configuração do tráfego do nodo “0,0” é possível aplicar a replicar a configuração a todos os demais nodos na rede.

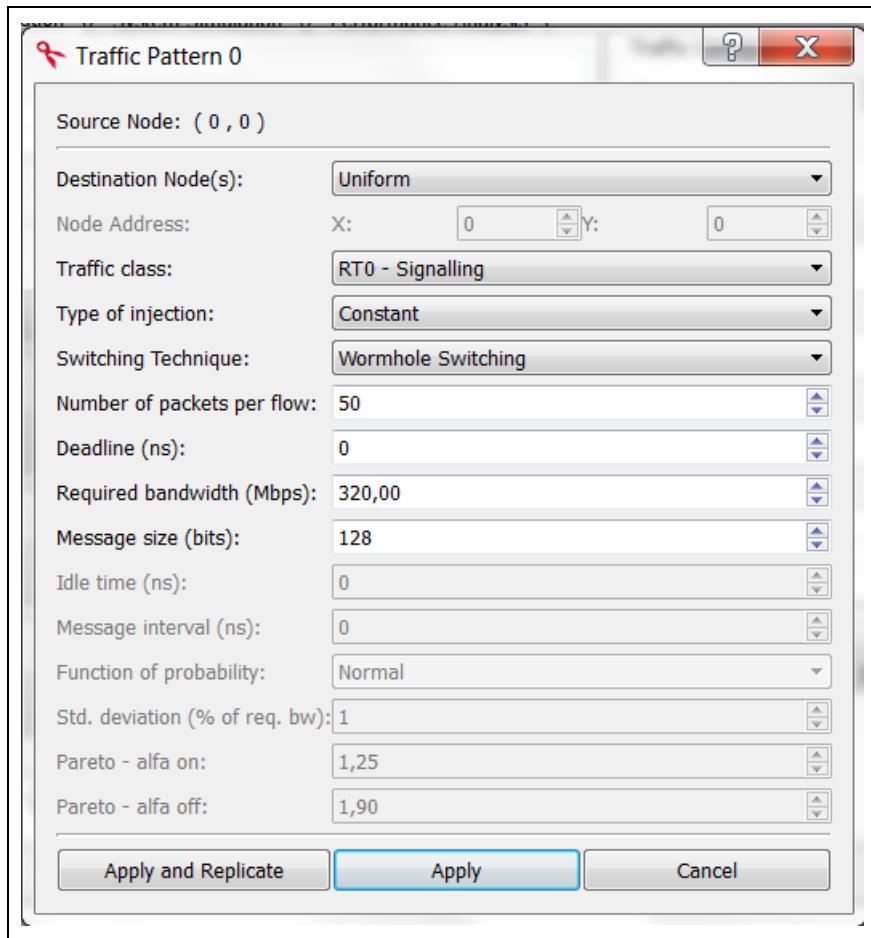


Figura 29. Tela de configuração de tráfego

B.7.6 Tela de exibição das formas de ondas

A tela de exibição das formas de ondas (Figura 30) exibe todos os sinais e ocorrências no tempo de simulação do sistema, permitindo a verificação funcional das operações da rede.

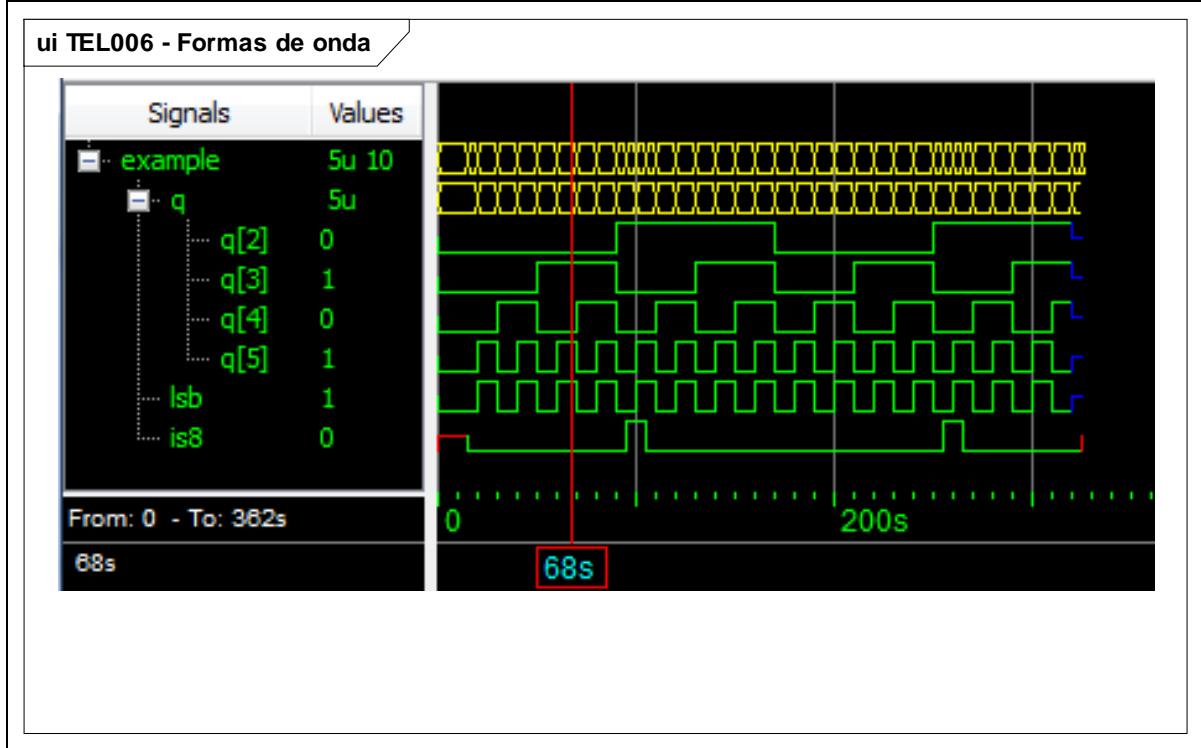


Figura 30. Protótipo de tela de visualização das formas de ondas

B.7.7 Tela de relatório

A tela de relatório (Figura 31) exibe os resultados de latência e vazão obtidos na simulação dos experimentos realizados.

FClk (MHz)	Packets Delivered	Offered Traffic (norm)	Offered Traffic (Mbps/n)	Accepted Traffic (f/c/n)	Accepted Traffic (Mbps/n)	Ideal Avg Latency (cycles)	Ideal Avg Latency (ns)	Avg Latency (cycles)	Avg Latency (ns)	Min Latency (cycles)	Min Latency (ns)	Max Latency (cycles)	Max Latency (ns)	Std Dev Latency
100	12000	0.1	320	0.0976	312.3	16	160	19.1	191.4	11	110	56	560	6.8
90	12000	0.1111	320	0.108	311	16	177.8	19.2	212.8	11	122.2	70	777.8	6.8
80	12000	0.125	320	0.1211	310	16	200	19.2	240	11	137.5	85	1062.5	6.9
70	12000	0.1429	320	0.1377	308.4	16	228.6	19.3	275.6	11	157.1	100	1428.6	7
60	12000	0.1667	320	0.1596	306.4	16	266.7	19.6	326.6	11	183.3	98	1633.3	7.5
50	12000	0.2	320	0.1903	304.5	16	320	20.5	409.6	11	220	90	1800	8.4
40	12000	0.25	320	0.2334	298.8	16	400	26.3	657.1	11	275	155	3875	16.4
30	12000	0.3333	320	0.26	249.6	16	533.3	672.1	22404.7	11	366.7	2190	73000	534.9
20	12000	0.5	320	0.2577	164.9	16	800	2178.7	108936	11	550	5024	251200	1320
10	12000	1	320	0.2574	82.4	16	1600	3634.9	363490	11	1100	7908	790800	2139.1
1	12000	1	320	0.2574	8.2	16	16000	3634.9	3.6349...	11	11000	7908	7.908e...	2139.1

Figura 31. Tela de relatório

B.7.8 Tela de visualização dos gráficos

As telas de visualização dos gráficos (Figura 32) facilitam a compreensão dos usuários na inferência dos resultados.

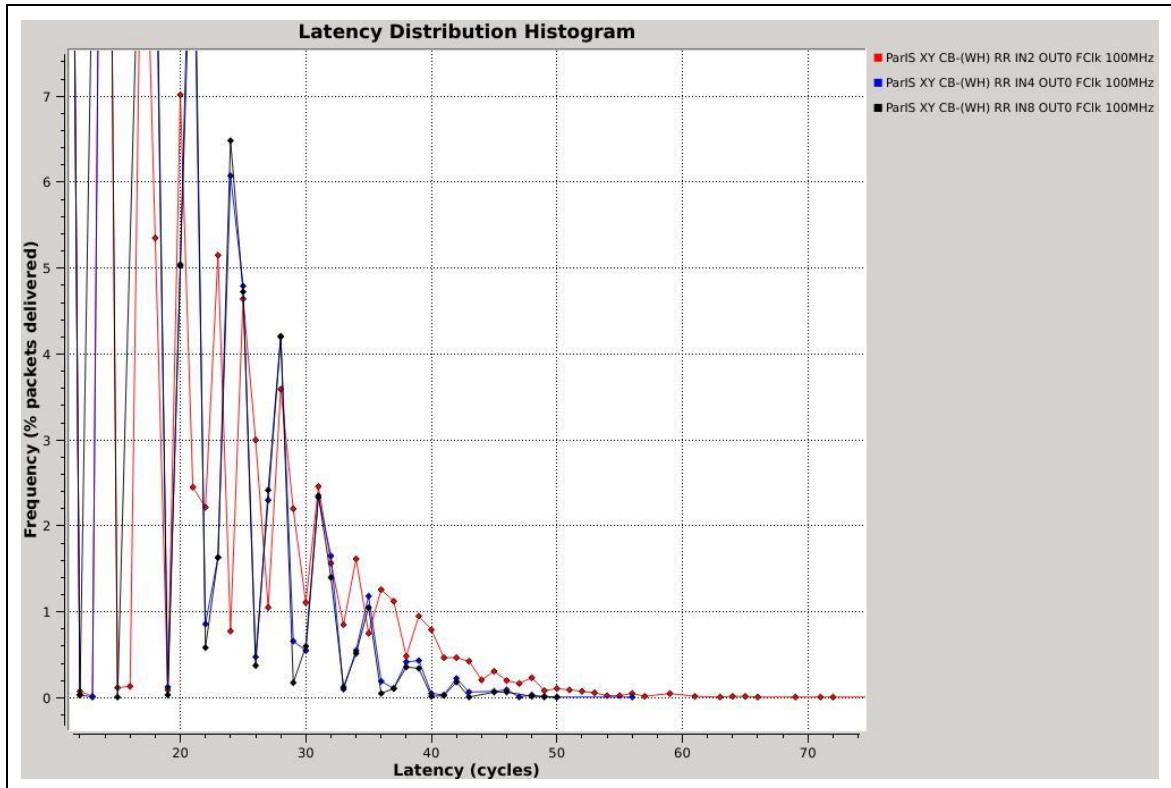


Figura 32. Tela de visualização dos gráficos

ANEXO A. Manual do Usuário

Este anexo contém o manual do usuário que foi elaborado pelo autor, porém por possuir formatação diferente ao texto, não foi colocado como apêndice. E tem por objetivo orientar o utilizador da ferramenta, sobre as funcionalidades presentes no ambiente.



RedScarf

Manual do Usuário

Versão 1.0



RedScarf – Manual do Usuário

Versão 1.0

Dezembro 2014

LEDS-UM-2014.01

Sobre

Este software foi desenvolvido no Laboratório de Sistemas Embarcados e Distribuídos (LEDS) da Universidade do Vale do Itajaí (Univali). Para mais informações, sugestões, reportar erros, por favor, entre em contato com:

- Prof. Dr. Cesar Albenes Zeferino (zeferino@univali.br)
- Eduardo Alves da Silva (eduardoalves@edu.univali.br)

Histórico de versões do documento

Descrição	Responsável	Data
Criação do documento	Eduardo Alves da Silva	12/2014
Revisão geral	Cesar Albenes Zeferino	12/2014

Informações sobre licenciamento

A definir

CONTEÚDO

<i>1</i>	<i>Introdução ao sistema RedScarf.....</i>	<i>1</i>
<i>1.1</i>	<i>Funcionalidades.....</i>	<i>2</i>
<i>1.2</i>	<i>Interface</i>	<i>3</i>
<i>2</i>	<i>Arquitetura.....</i>	<i>6</i>
<i>3</i>	<i>Instalação do sistema.....</i>	<i>8</i>
<i>3.1</i>	<i>Pré-requisitos.....</i>	<i>8</i>
<i>3.2</i>	<i>Instalação em Windows</i>	<i>9</i>
<i>3.3</i>	<i>Instalação em Linux.....</i>	<i>10</i>
<i>3.4</i>	<i>Instalação em OS X.....</i>	<i>10</i>
<i>4</i>	<i>Funcionalidades.....</i>	<i>11</i>
<i>4.1</i>	<i>Visão Geral</i>	<i>11</i>
<i>4.2</i>	<i>Interface</i>	<i>12</i>
<i>4.2.1</i>	<i>Barra de menu</i>	<i>13</i>
<i>4.2.2</i>	<i>Barra de estado</i>	<i>14</i>
<i>4.2.3</i>	<i>Console de mensagens</i>	<i>14</i>
<i>4.2.4</i>	<i>Barra de ferramentas</i>	<i>15</i>
<i>4.2.5</i>	<i>Abas de configuração, experimentação e análise.....</i>	<i>16</i>
<i>4.3</i>	<i>Ferramentas</i>	<i>17</i>
<i>5</i>	<i>Como usar.....</i>	<i>19</i>
<i>5.1</i>	<i>Configuração da ferramenta.....</i>	<i>19</i>
<i>5.2</i>	<i>Definir as dimensões do sistema e a largura do canal de comunicação.....</i>	<i>21</i>
<i>5.3</i>	<i>Definir padrões de tráfego para os nodos na rede</i>	<i>23</i>
<i>5.4</i>	<i>Configurar os experimentos, definir o intervalo de frequência de operação e executar a simulação.....</i>	<i>25</i>
<i>5.5</i>	<i>Realizar a análise.....</i>	<i>26</i>
<i>5.6</i>	<i>Visualizar os resultados.....</i>	<i>27</i>
<i>5.7</i>	<i>Exercício: Cenário de exemplo.....</i>	<i>31</i>
<i>6</i>	<i>Referências Bibliográficas</i>	<i>45</i>

1 Introdução ao sistema RedScarf

O RedScarf é um ambiente integrado que permite avaliar diferentes configurações de uma NoC (Network-on-Chip ou, em português, Rede-em-Chip) por meio de simulação baseada em modelos SystemC em nível RTL da rede e RTL/TL dos geradores de tráfego. O RedScarf utiliza como NoC de referência a rede SoCINfp (System-on-Chip Interconnection Network *fully parametrizable*), a qual trata-se de uma rede parametrizável de baixo custo (ZEFERINO; SANTO; SUSIN, 2004).

O ambiente interface gráfica para facilitar a configuração dos parâmetros dos experimentos, ferramentas para a geração do modelo da NoC, dos modelos de tráfego e para análise dos resultados, disponibilizando métricas de desempenho como vazão e latência, entre outras. Conta com ferramentas para geração de gráficos e relatórios para análise dos resultados da simulação. Também é capaz de executar ferramenta externa para visualização da simulação por meio de um diagrama de formas de onda a partir de arquivo em formato VCD (Value Change Dump) gerado pelo simulador, o que permite ao usuário realizar uma análise mais detalhada da simulação.

Este software foi desenvolvido e é mantido pelo **Laboratório de Sistemas Embarcados e Distribuídos** (LEDS – Laboratory of Embedded and Distributed Systems) da **Universidade do Vale do Itajaí** (Univali – SC/Brasil) e é uma atualização do BrownPepper (BRUCH; PIZZONI; ZEFERINO, 2009), que foi a primeira ferramenta desenvolvida para avaliação de desempenho da SoCINfp. O RedScarf é fruto de um trabalho de conclusão de curso de graduação na área de Ciência da Computação.

Os autores do BrownPepper, foram os responsáveis pela construção do simulador da SoCINfp e o RedScarf é a atualização da interface gráfica e da parte de configuração e automação dos experimentos. Os contribuintes para o resultado deste trabalho são:

- Cesar Albenes Zeferino (Mantedor e Desenvolvedor)
- Jaison Valmor Bruch (Desenvolvedor – [BrownPepper](#))
- Eduardo Alves da Silva (Desenvolvedor e Documentador – [RedScarf](#))

1.1 Funcionalidades

As funcionalidades do RedScarf estão categorizadas e apresentadas a seguir.

- Recursos para configuração de experimento
 - Configuração do dimensionamento da rede e do canal de comunicação;
 - Definição de padrões de tráfego para cada nodo na rede;
 - Configuração experimentos com as propriedades do roteador;
 - Simulação baseada em tempo ou na entrega de todos os pacotes.
- Recursos para automação do experimento
 - Construção automatizada do modelo de simulação da SoCINfp;
 - Geração do modelo de tráfego;
 - Execução automática.
- Recursos para análise dos resultados
 - Análise de desempenho considerando aquecimento e resfriamento da rede;
 - Visualizador de gráficos;
 - Visualizador de relatórios;
 - Executor de ferramenta externa de formas de ondas;
 - Geração de relatório da simulação.
- Recursos para acelerar os experimentos
 - Cache de objetos do modelo da SoCINfp
 - Execução paralela das simulações.
- Recursos para facilitar o uso da ferramenta
 - Visualização dos padrões de tráfegos configurados em forma de lista e árvore;
 - Arquivamento e carregamento da configuração da rede, tráfego e experimentos;
 - Console integrado de mensagens;
 - Possibilidade de definir a configuração padrão do ambiente;
 - Verificar salvamento quando feitas alterações nas configurações;
 - Menu de configuração do ambiente;

- Internacionalização (Inglês e Português);
- Arquivamento e carregamento dos resultados das simulações;
- Documentação inclusa na ferramenta;
- Multiplataforma.

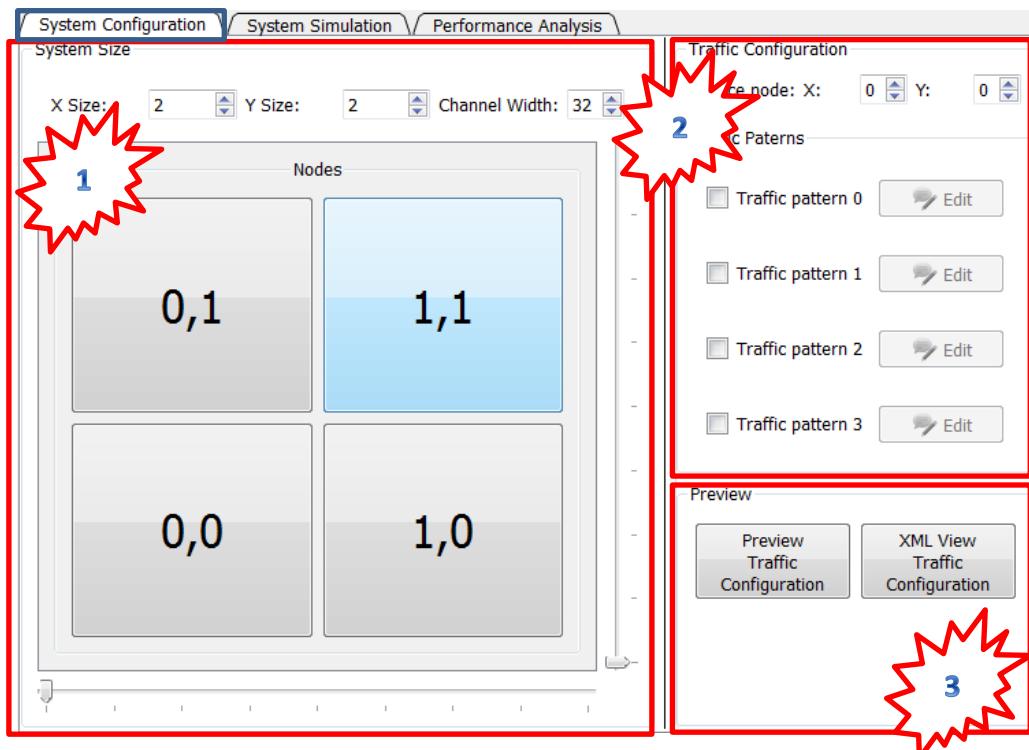
A descrição e detalhamento destas características e funcionalidades são apresentadas no capítulo Capítulo 4.

1.2 Interface

A interface do sistema é composta de uma tela principal (com três abas) e de caixas de diálogo para facilitar as operações do usuário. Na versão atual, a interface pode ser exibida em inglês ou em português, e neste manual, todas as capturas de tela apresentadas são da configuração em língua inglesa.

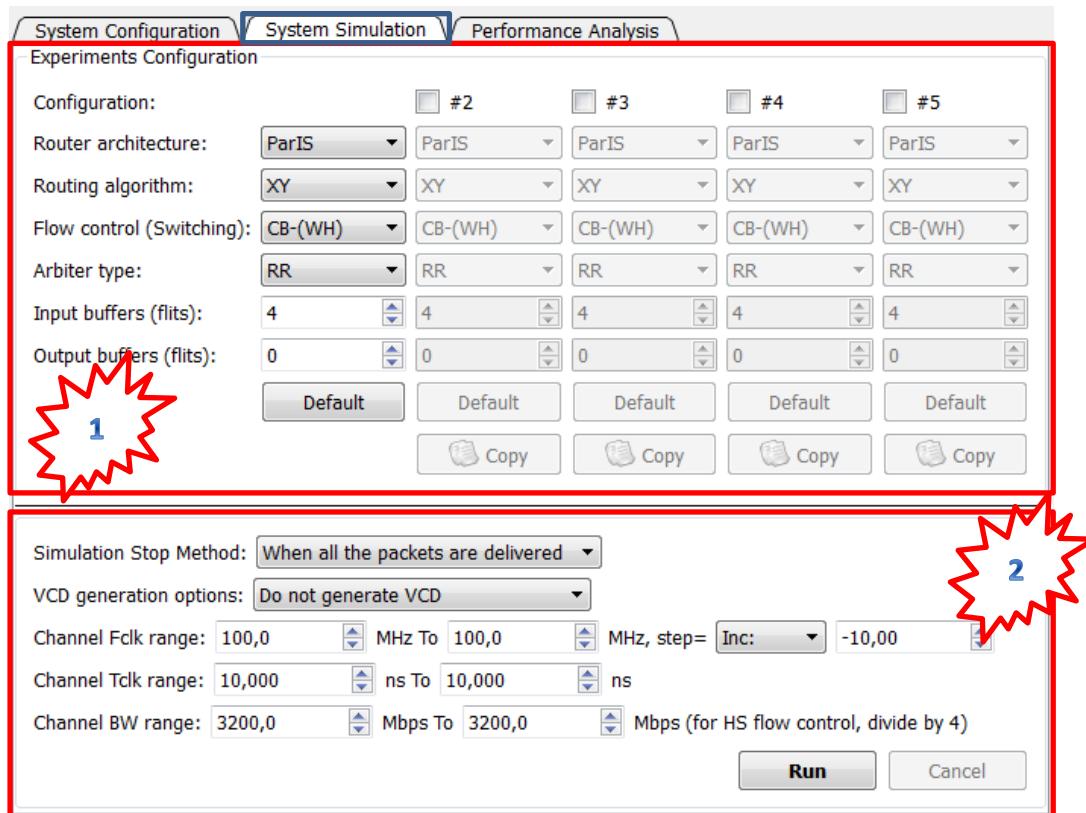
As abas da interface são dispostas de acordo com o fluxo de uso da ferramenta, embora o usuário possa chavear livremente entre uma e outra aba. As três abas da tela principal são intituladas: **System Configuration** (Configuração do Sistema), **System Simulation** (Simulação do Sistema); e **Performance Analysis** (Análise de Desempenho).

A aba **System Configuration** é dividida em três partes, apresentadas a seguir.



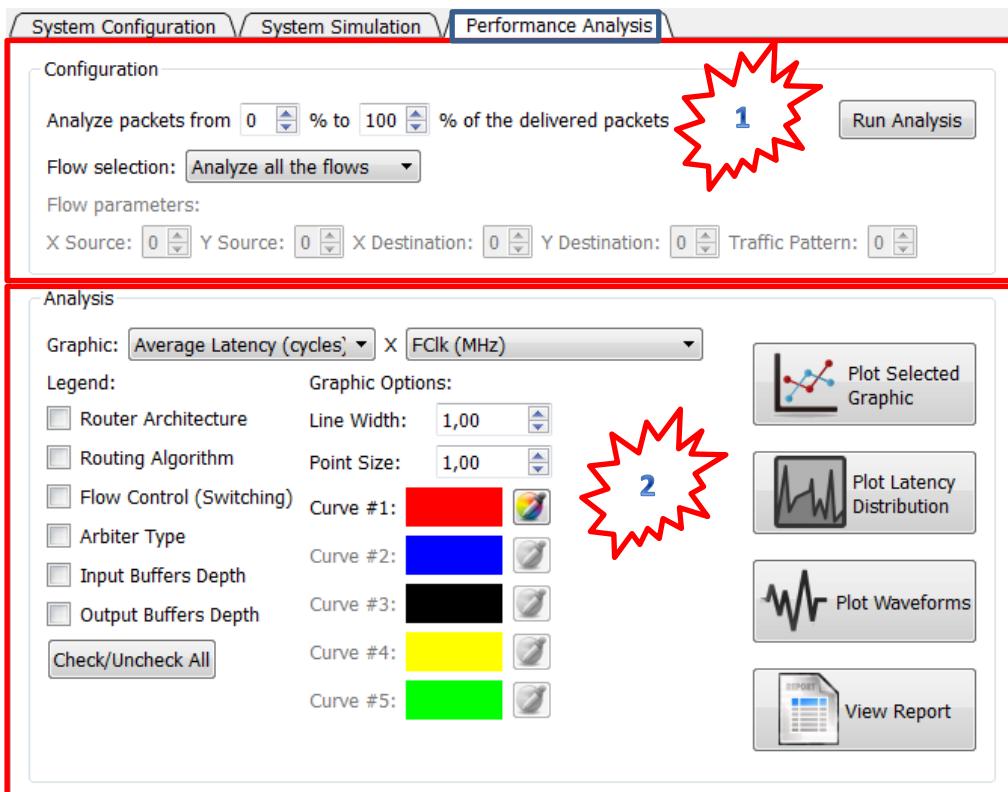
- 1) **System Size (Tamanho do Sistema):** nesta parte, são definidos os tamanhos em X e Y do sistema (ou seja, o tamanho da rede) e a largura dos canais de comunicação da rede. O tamanho do sistema define a quantidade de roteadores da rede nos eixos horizontal e vertical sendo que o limite mínimo desses eixos é de 2 roteadores e o máximo é de 10 roteadores.
- 2) **Traffic Configuration (Configuração do Tráfego):** nesta parte, o usuário tem a possibilidade de definir até quatro padrões de tráfego para cada nodo do sistema. Esses padrões são reproduzidos por geradores de tráfego que injetam pacotes na rede conforme os parâmetros de configuração definidos pelo usuário.
- 3) **Preview (Visualização):** esta parte da aba permite ao usuário verificar os padrões de tráfego configurados.

A aba de **System Simulation** é dividida em duas partes, apresentadas a seguir.



- 1) **Experiments Configuration (Configuração dos Experimentos):** nesta parte, o usuário define os parâmetros do roteador utilizado pela rede (ex. profundidade dos buffers). Podem ser definidas até cinco configurações diferentes.
- 2) **Configurações adicionais e execução:** nesta parte, o usuário define o método de parada para a simulação dos experimentos, a opção da geração de arquivo VCD de formas de ondas e o intervalo de frequências de operação para realização dos experimentos. O usuário deve definir a frequência de operação da rede para o primeiro e para o último experimento, além do passo de incremento/decremento de frequência do primeiro ao último experimento. Esse passo e o intervalo irão definir a quantidade de simulações a ser realizada para cada configuração de roteador (ex: para um intervalo de 10 a 100 MHz, com passo de 10 MHz, são realizadas 10 simulações). Nesta parte da interface encontra-se o botão **Run** pelo qual o usuário aciona o início das simulações após ter completado a configuração dos experimentos.

A aba **Performance Analysis** é dividida em 2 partes, apresentadas a seguir.



- 1) **Configuration (Configuração):** esta parte permite definir o intervalo de análise dos pacotes e os fluxos a serem analisados. Isso possibilita descartar os pacotes referente às fases de aquecimento (primeiros pacotes) e drenagem (últimos pacotes) da simulação, bem como selecionar fluxos individuais ou classes de fluxo para análise. Esta parte contém o botão **Run Analysis**, o qual deve ser acionado após a configuração da análise para que sejam processados apenas os pacotes dos intervalos e dos fluxos selecionados. A partir desses, são calculadas as métricas a serem exibidas pela ferramenta;
- 2) **Analysis (Análise):** esta parte refere-se à exibição dos resultados dos experimentos de forma gráfica ou textual. É possível definir os eixos do gráfico a ser gerado (ex. Latência média x Frequência de operação), escolhendo a opção pelo eixo X e Y, marcar a legenda para diferenciação dos dados, configurar as opções do gráfico e visualizar o gráfico selecionado. Além disto, é possível visualizar a distribuição de latência de todas as simulações, os relatórios em forma de tabela e os diagramas de formas de ondas dos experimentos realizados.

2 Arquitetura

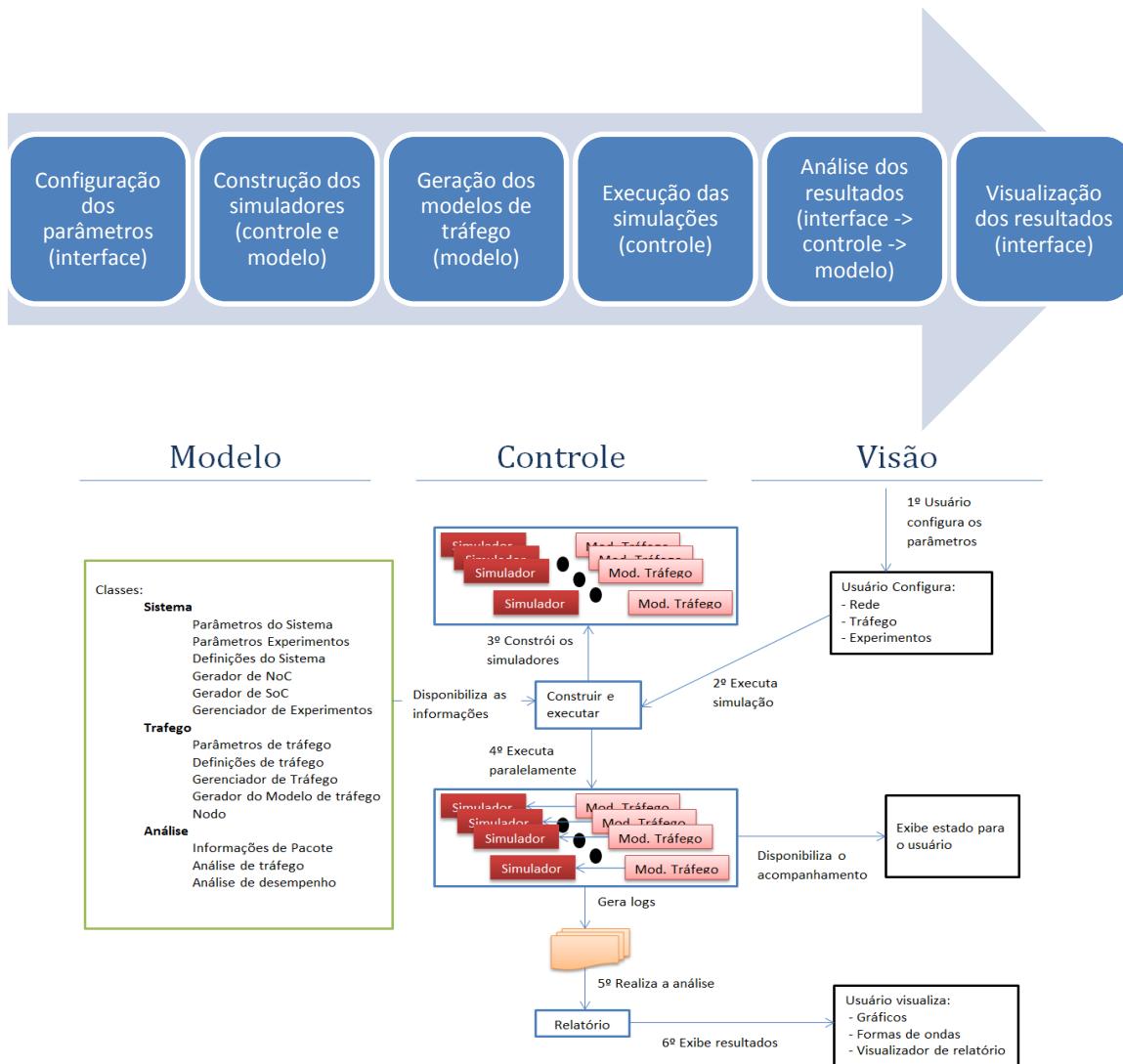
O ambiente é um software Desktop de janela única que está disponível nas plataformas Windows (.exe), Linux (executável via script shell .sh) e Mac OS X (.app contido em imagem .dmg). Conta com interface gráfica para todas as configurações. Realiza simulação e análise de desempenho automaticamente, disponibilizando formas alternativas de visualizar os resultados.

A arquitetura do sistema pode ser vista na figura a seguir.



Para melhor organizar a estrutura da ferramenta, a parte de configuração (*front-end*) é separada da parte de simulação (*back-end*). No módulo da interface do *front-end*, são feitas todas as interações do usuário que são encaminhadas ao controlador da aplicação para o correto tratamento da ação disparada. Quando necessário o controlador utiliza as ferramentas do modelo para realizar o processamento completo da ação do usuário.

O fluxo de funcionamento da ferramenta é apresentado nas figuras a seguir. O processo começa com o usuário definindo os parâmetros na Interface (Visão) e dando início ao processo de simulação. O fluxo é repassado ao controle para a construção, execução e análise das simulações, utilizando as ferramentas e estruturas disponíveis no próprio controle e no modelo da aplicação. Após o processamento realizado pelo controle, os resultados ficam disponíveis para o usuário visualizar por meio de gráficos, diagramas de formas de ondas e relatórios.



3 Instalação do sistema

A instalação está disponível nas plataformas por meio de pacotes. No Windows um executável instalador que copia os arquivos e diretórios na estrutura correta junto as dlls necessárias. No Linux através de pacotes do Debian (.deb) e para sistemas derivados do RedHat (.rpm), solicitando as dependências necessárias (os repositórios podem ser obtidos através de contato com os pesquisadores da lista disponível no capítulo 7). No OS X, o pacote está disponível como aplicação com seu próprio Bundle em uma imagem Apple (.dmg).

3.1 Pré-requisitos

As dependências do software são as bibliotecas do Qt que são incorporadas aos pacotes ou solicitadas durante a instalação.

É preciso ter a biblioteca SystemC compilada e um compilador compatível com tal. No Windows, sugere-se o uso do MinGW (<http://www.mingw.org/>). No Linux, recomenda-se o pacote “build-essential” em distribuições Debian e o grupo de pacotes de desenvolvimento “Development Tools” em distribuições baseadas no RedHat. No Mac OS X, recomenda-se o XCode com a extensão “Command Line Tools”.

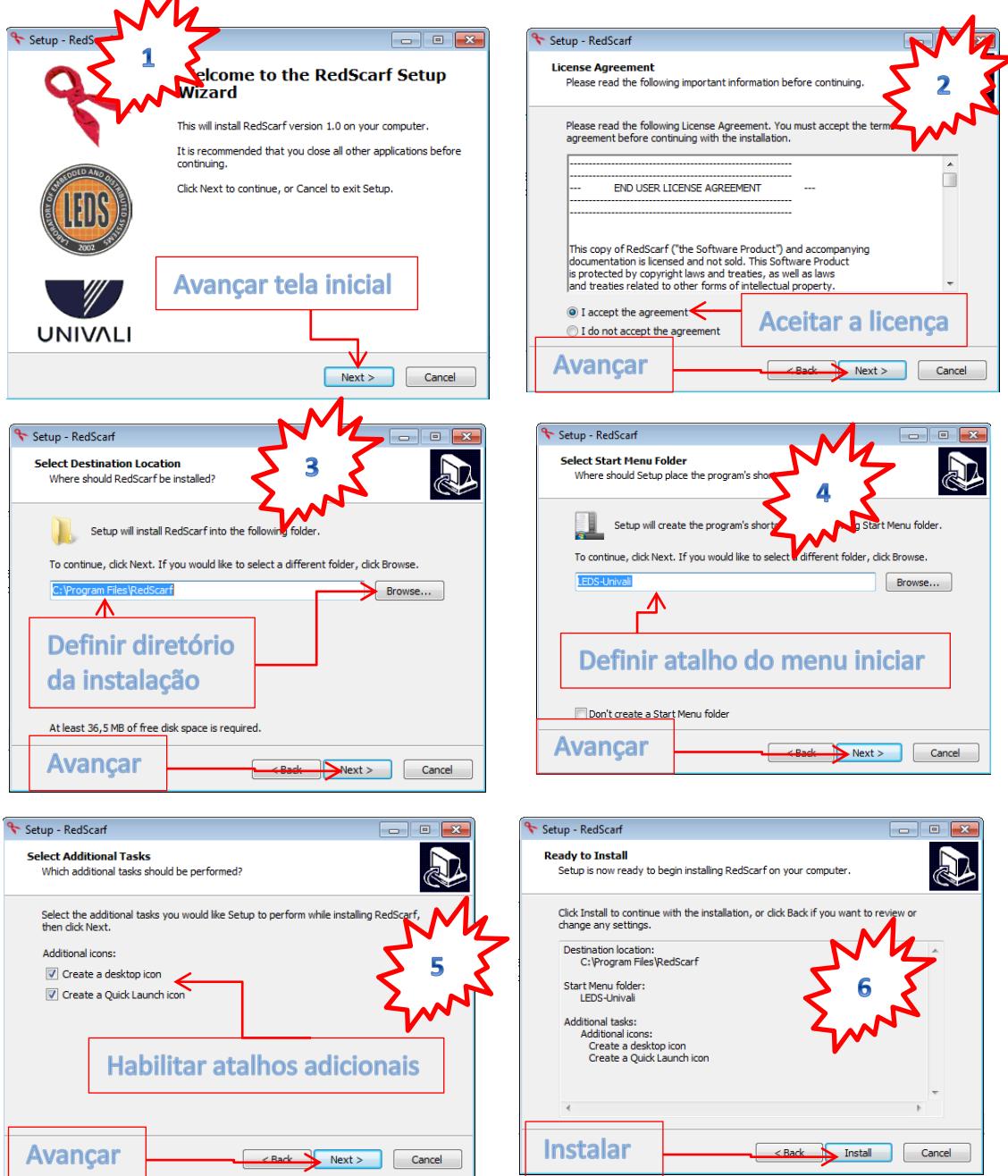
A biblioteca SystemC e o compilador são necessários para a construção dos simuladores, sendo que a NoC é modelada utilizando SystemC (C++).

Somente com privilégios de administrador a instalação pode ser feita. Isto se aplica a todas as plataformas.

IMPORTANTE: Nas plataformas baseadas em Unix o caminho do compilador deve estar definido nas variáveis de ambiente.

3.2 Instalação em Windows

Para realizar a instalação no Windows basta seguir os passos ilustrados.



Após seguir os passos ilustrados basta finalizar o instalador e o RedScarf está instalado.

3.3 Instalação em Linux

A instalação em sistemas Linux é feita por meio da adição do repositório da ferramenta e da instalação via gerenciador de pacotes.

Para sistemas baseados em Debian a instalação é feita via “apt”, com o comando:

```
apt-get install RedScarf
```

Em sistemas baseados em RedHat a instalação é feita com o comando:

```
yum install RedScarf
```

IMPORTANTES: Para ambos os tipos de ambientes será necessária à resolução das dependências. Para a correta ligação das bibliotecas da aplicação, é necessário que o comando de instalação seja feito com privilégios de administrador.

3.4 Instalação em OS X

Para uso no Mac OS X, basta extrair a aplicação para o diretório desejado do sistema. Ou simplesmente executar após montar a imagem (.dmg) no sistema. Pois a aplicação é autocontida possuindo todas as configurações necessárias já embarcadas.

IMPORTANTES: Vale lembrar que para todas as plataformas é necessário um visualizador de formas de ondas externo compatível com arquivos VCD (Value Change Dump), que deve ser configurado no ambiente. O SystemC e o compilador compatível devidamente instalados e configurados.

4 Funcionalidades

4.1 Visão Geral

O RedScarf permite a configuração dos parâmetros da rede SoCINfp com topologia baseada em malha. Inicialmente, é possível definir o tamanho do sistema e da rede (número de nodos em X e em Y) e a largura do canal de comunicação em bits.

Após o dimensionamento do sistema, é possível definir até quatro padrões de tráfegos para cada nodo do sistema. A configuração dos padrões de tráfego define as informações que serão utilizadas pelos geradores de tráfego para injetar fluxos de comunicação na rede durante a simulação. Uma vez definidos os padrões de tráfego, a ferramenta possui visualizadores para o usuário confirmar se a configuração está correta, caso seja necessário.

Após a configuração inicial, o ambiente disponibiliza a possibilidade de definir até 5 configurações de roteador para comparação. Também permite definir a opção de parada das simulações e um intervalo de frequências de operação da rede.

A partir das definições supracitadas, o ambiente está pronto para realizar a simulação. Para cada configuração de roteador, é gerado um simulador do sistema e seus modelos de tráfego (um para cada frequência de operação a ser simulada).

Com os simuladores gerados e modelos de tráfego prontos, a simulação é iniciada automaticamente. Conforme a configuração da ferramenta e o computador utilizado, o RedScarf pode despachar múltiplas simulações simultâneas, explorando o seu suporte a processamento *multithread*, o que reduz o tempo total para execução dos experimentos.

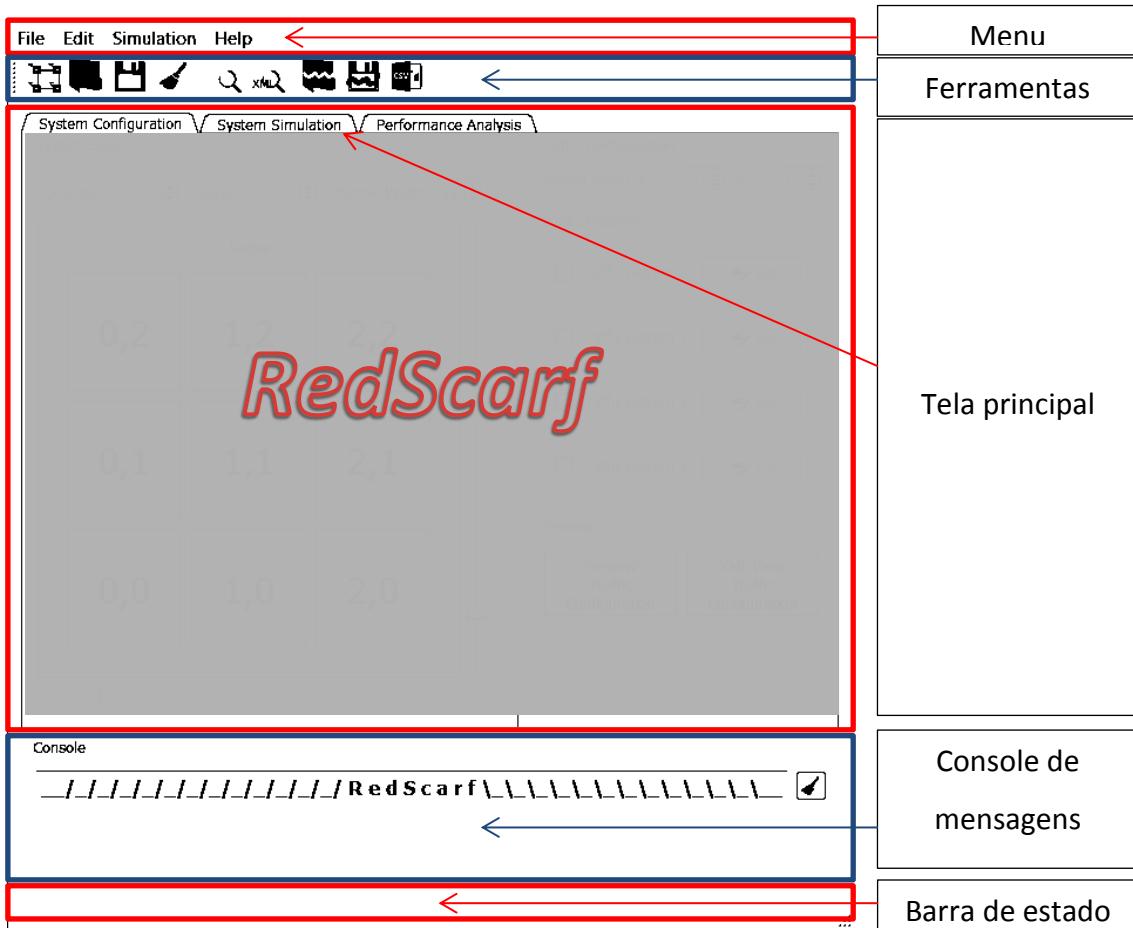
Concluídas todas as experimentações da simulação, é possível salvar os resultados da simulação para restaurar posteriormente (o que evita a necessidade de simular novamente o sistema). Mas, para dar continuidade ao processo de verificação é necessária definição do intervalo de pacotes a ser analisado. Esse intervalo é definido em percentuais, sendo possível eliminar um percentual inicial (fase de aquecimento) e final (fase de drenagem) dos pacotes entregues na rede. Com o intervalo definido, é iniciado o processo de análise dos *logs* gerados pela simulação. Este processo pode demorar, pois depende da leitura e escrita de arquivos no disco.

Com o processo de análise concluído, a ferramenta disponibiliza a visualização do relatório da simulação, de gráficos de distribuição de latência para cada experimento em cada frequência de operação simulada, de gráficos para diferentes métricas de desempenho (latência, tráfego aceito, prazos cumpridos) e de diagramas de formas de onda da simulação. Também é possível exportar o relatório em formato CSV (Comma-Separated Values) para análise por outras ferramentas se for da escolha do utilizador.

Além de todo o processo de configuração, simulação e análise, a ferramenta dispõe de recursos para salvar e carregar as configurações realizadas em formato XML, definir uma ferramenta para a visualização das formas de ondas e configurar o diretório de trabalho, o qual é utilizado para salvar os *logs* da simulação. Com isso, permite a execução de diversas experimentações em diferentes diretórios de trabalho, ao mesmo tempo.

4.2 Interface

A interface gráfica possui cinco componentes principais para uso da ferramenta: (1) barra de menu; (2) barra de ferramentas; (3) tela principal (com suas três abas); (4) console de mensagens; e (5) barra de estado, os quais são apresentados a seguir.



4.2.1 Barra de menu

Na barra de menu, há quatro opções: File (Arquivo), Edit (Editar), Simulation (Simulação) e Help (Ajuda).

As opções disponíveis nos menus são:

- **File (Arquivo)**
 - New (Novo): carrega a configuração padrão e limpa as definições de tráfego;
 - Open (Abrir): abre um arquivo de configuração;
 - Load Default (Carrega padrão): carrega a configuração padrão;
 - Save (Salvar): salva a configuração atual;
 - Save As (Salvar como): salva a configuração atual em um novo arquivo;
 - Save As Default (Salvar como padrão): salva a configuração atual como definição padrão;
 - Clear All Traffic Patterns (Limpar todos os padrões de tráfego): limpa todos os padrões de tráfego definidos e mantém as demais configurações;
 - Exit (Sair): encerra a aplicação;

- **Edit (Editar)**
 - Language (Língua): disponibiliza os idiomas para a visualização da aplicação;
 - Options (Opções): abre o menu de configuração do ambiente;
- **Simulation (Simulação)**
 - Load Simulation (Carregar simulação): permite carregar uma simulação já realizada;
 - Save Simulation (Salvar simulação): permite salvar a simulação realizada;
 - Generate CSV (Gerar relatório CSV): gera um arquivo no formato CSV contendo os resultados da simulação;
- **Help (Ajuda)**
 - Documentation (Documentação): disponibiliza a documentação da ferramenta;
 - About Qt (Sobre o Qt): exibe informações sobre o Qt;
 - About (Sobre): exibe informações sobre o RedScarf.

Algumas destas funções possuem atalhos de teclado para facilitar o acesso, como por exemplo, Ctrl+S para salvar a configuração da simulação.

As funções mais comuns também são disponibilizadas na barra de ferramentas que é apresentada na próxima seção.

4.2.2 Barra de estado

A barra de estado exibe algumas informações adicionais das funcionalidades enquanto o usuário passa o ponteiro do mouse sobre os elementos da ferramenta.

4.2.3 Console de mensagens

Neste console, são apresentadas todas as mensagens textuais de realimentação ao usuário. As saídas dos simuladores são apresentadas neste console, sendo que as informações disponíveis são apenas de início e final de simulação. As ações realizadas com sucesso ou erro são informadas através deste console.

4.2.4 Barra de ferramentas

A barra de ferramentas dá um acesso rápido a algumas funções disponíveis nos menus e nas abas da aplicação. São elas:

Ícone	Aplicação
	New (Novo): Opção para carregar a configuração padrão do ambiente e limpar as definições de tráfego da configuração padrão.
	Open (Abrir): Opção para carregar um arquivo previamente salvo de configuração do ambiente. Carrega a dimensão, tráfego e experimentos.
	Save (Salvar): Salvar a configuração atual do ambiente no arquivo atual ou novo, caso não tenha sido definido ainda. Salva dimensão, tráfego e experimentos. Arquivo salvo em formato XML.
	Clear All Traffic Patterns (Limpar todos os padrões de tráfego): Limpa todas as configurações de tráfego do sistema.
	Preview Traffic Configuration (Visualizar configuração de tráfego): Exibe a configuração de tráfego atual em forma de lista .
	XML View Traffic Configuration (Visualizar configuração de tráfego (XML)): Exibe a configuração de tráfego atual em forma de árvore (níveis XML).
	Load Simulation (Carregar simulação): Opção para carregar o arquivo de uma simulação previamente realizada. Carrega os resultados de uma simulação, analisados ou não, dependendo de como o arquivo foi salvo no diretório de trabalho atualmente definido.
	Save Simulation (Salvar simulação): Opção para salvar os resultados da simulação disponíveis no diretório de trabalho atualmente definido em um arquivo binário com uma taxa de compressão de aproximadamente 85%.
	Generate CSV (Gerar relatório CSV da simulação): Gera um relatório da análise da simulação em arquivo no formato CSV.

IMPORTANTE: A saída de um experimento de simulação é formada por arquivos de *log* contendo informações sobre cada pacote transferido na rede. Para minimizar a ocupação de disco, ao salvar uma simulação, é feita uma compressão desse dados. No entanto, conforme a quantidade de pacotes transferidos, esse arquivo pode ocupar várias centenas de MB.

4.2.5 Abas de configuração, experimentação e análise

As abas de uso do sistema são onde o usuário define todos os parâmetros de entrada da simulação, executa a simulação e a análise, e é capaz de visualizar os resultados. Conforme já apresentado, a tela principal é composta de três abas dispostas conforme o fluxo de uso da ferramenta:

1^a Aba – Configuração do sistema: Nesta aba, o usuário:

- define as dimensões do sistema no eixos X e Y;
- define a largura do canal de comunicação, em bits;
- pode definir até 4 padrões de tráfego para cada nodo da rede;
- pode visualizar a configuração de tráfego em forma de lista ou árvore.

2^a Aba – Experimentação (simulação do sistema): Nesta aba, o usuário:

- define até 5 experimentos com os seguintes parâmetros: (1) arquitetura do roteador; (2) algoritmo de roteamento; (3) controle de fluxo; (4) tipo de árbitro; (5) profundidade dos *buffers* de entrada; e (6) profundidade dos *buffers* de saída;
- define o método de parada da simulação em que as opções são: (1) quando todos os pacotes forem entregues; (2) após um determinado tempo simulado (em nano segundos); e (3) após um determinado número de ciclos simulados;
- define as opções de geração de formas de ondas. A primeira opção não gera o arquivo VCD, enquanto as demais servem para selecionar quais sinais serão exibidos, gerando um *script* para a ferramenta de visualização. A versão atual do RedScarf gera *script* apenas para o GTK Wave. Portanto, essa seleção ainda não está disponível para outras ferramentas, nas quais a opção consiste em exibir ou não todos os sinais da simulação;
- define um intervalo de frequências de operação a ser simulado. Este intervalo é definido por um valor inicial, um valor final e o passo que pode ser incremental (positivo ou negativo) ou exponencial (positivo ou negativo).
- executa e pode cancelar o processo de simulação.

3^a Aba – Análise de desempenho: Nesta aba, o usuário:

- define um intervalo em que é possível desconsiderar um percentual inicial e final dos pacotes na análise;
- executa a análise;
- seleciona qual o fluxo que deseja visualizar os resultados;
- escolhe as variáveis a serem utilizadas no gráfico que deseja visualizar;
- define a opção de legenda para a diferenciação na visualização;
- escolhe a largura da linha e tamanho do ponto dos gráficos;
- escolhe a cor de cada linha em relação aos experimentos realizados para visualização dos gráficos;
- visualiza o gráfico selecionado;
- visualiza a distribuição de latência selecionando quais experimentos deseja visualizar;
- visualiza os relatórios da simulação; e
- pode iniciar um visualizador de formas de ondas devidamente configurado.

4.3 Ferramentas

As principais ferramentas do RedScarf são:

- Gerador de modelo SystemC de NoC;
- Gerador de modelo SystemC de SoC;
- Gerador do modelo de tráfego;
- Ferramenta de construção dos simuladores;
- Ferramenta de execução;
- Gerenciador de *threads*;
- Analisador de pacotes;
- Plotadores de gráficos; e
- Visualizadores de diagramas de formas de onda.

As ferramentas de geração da NoC, SoC e modelo de tráfego são utilizadas quando o usuário dá início ao processo de simulação. A ferramenta de geração da NoC

é responsável por criar os arquivos SystemC do modelo da rede SoCINfp na versão de acordo com a arquitetura do roteador definida para o experimento. A geração do SoC é semelhante à da NoC, porém os arquivos gerados são os de ligação do sistema e os respectivos *testbenches*. Já o modelo de tráfego é gerado após a construção (compilação do modelo SystemC) do sistema. A partir das configurações de tráfego, são gerados os arquivos de texto que servem de entrada para os simuladores. Após essas gerações, a é iniciada a simulação *multithread*, de acordo com o número de *threads* definidas pelo usuário na configuração do ambiente. Não é feita simulação paralela. Cada *thread* corresponde à execução de um simulador submetido a um padrão de tráfego. Logo, em um computador com processador *quadcore*, por exemplo, podem ser executados quatro simuladores simultaneamente.

A construção do sistema é feita por uma ferramenta específica que gera o arquivo de montagem do simulador. Após, são executadas as ferramentas de geração citadas anteriormente e o processo de construção é iniciado.

Para a simulação, existem duas ferramentas que cuidam do processo. Uma responsável por executar as simulações e outra responsável por gerenciar o número de *threads* em execução.

Após o término das simulações, a ferramenta de análise deve ser executada para gerar os dados para visualização. Com os dados analisados, o usuário é capaz de visualizar os gráficos e relatórios, e pode executar uma ferramenta de visualização de formas de ondas previamente configurada no ambiente, desde que tenha sido habilitada a geração das formas de ondas na simulação.

5 Como usar

O fluxo de uso do RedScarf para realizar a simulação, analisar e visualizar os resultados é:

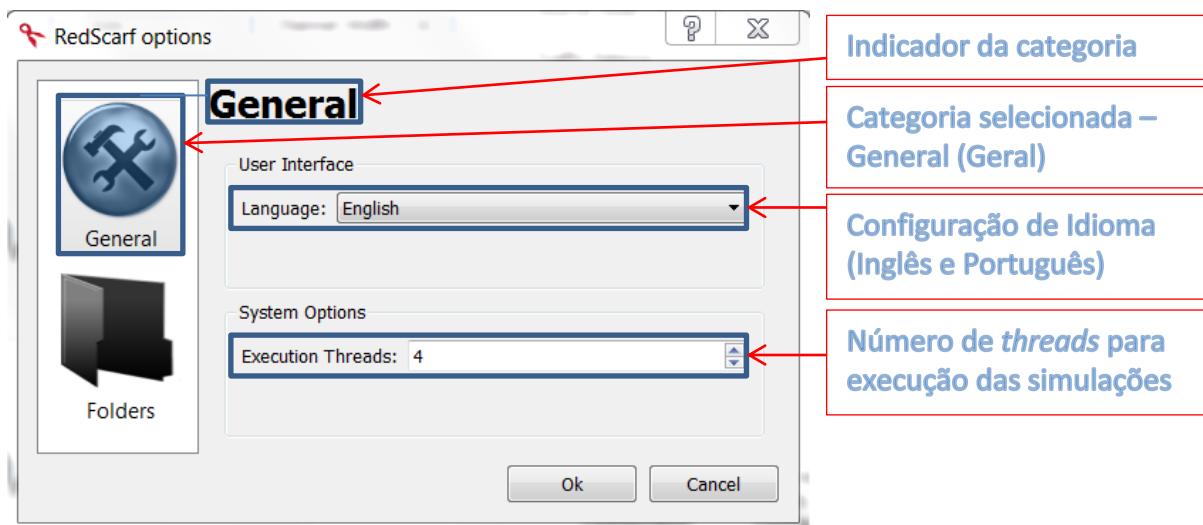
1. Definir as dimensões do sistema e a largura dos canais de comunicação;
2. Definir padrões de tráfegos para os nodos na rede;
3. Configurar os experimentos, definir o intervalo de frequência de operação e executar a simulação;
4. Realizar a análise; e
5. Visualizar os resultados.

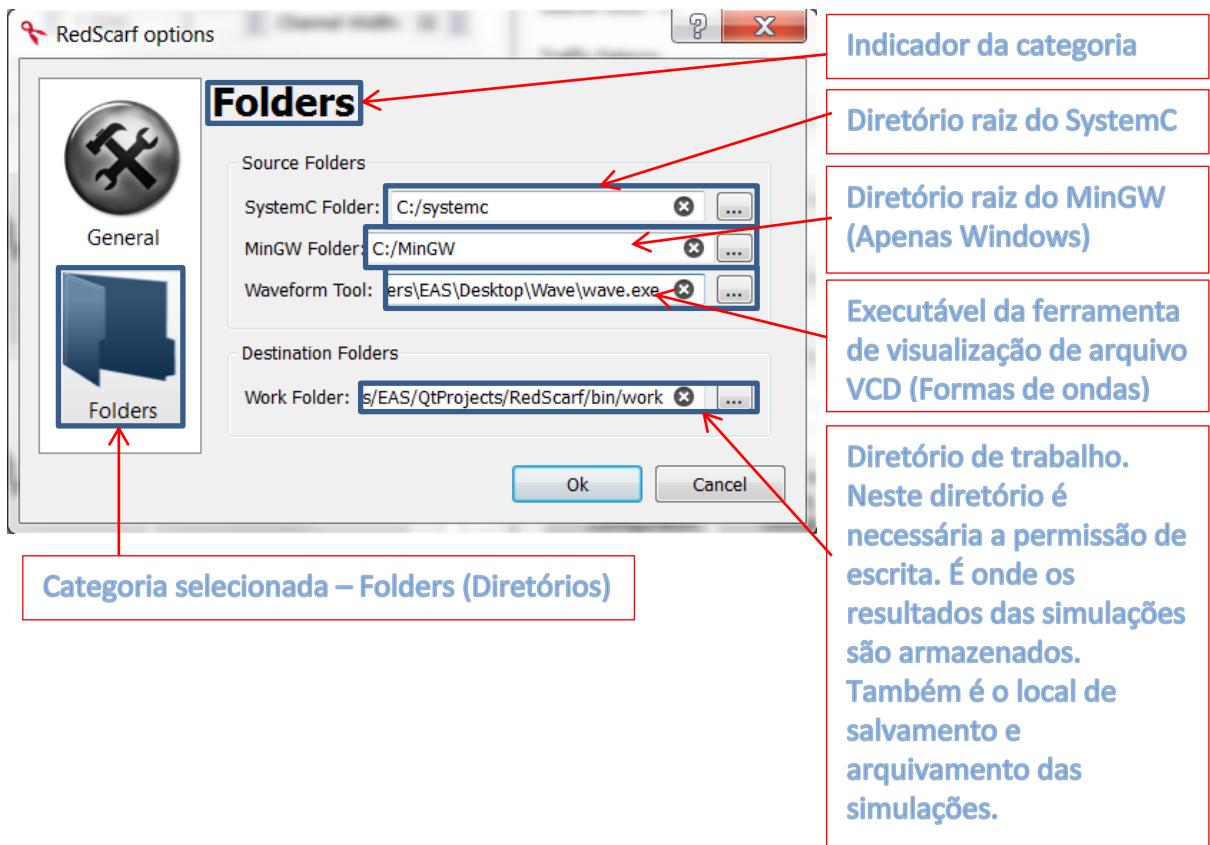
Na primeira execução, será solicitada a configuração do diretório raiz do SystemC. No Windows, também é solicitada a configuração do diretório raiz do MinGW. Essas opções podem ser definidas posteriormente no menu de configuração da ferramenta, que será apresentado na seção 7 deste capítulo.

5.1 Configuração da ferramenta

Além do fluxo de utilização da ferramenta para realizar experimentos, há também a configuração do ambiente, em que algumas definições são necessárias. Esta parte de configuração é acessível através do menu: Edit (editar) -> Options (opções).

A configuração do ambiente é dividida em 2 partes, uma de definições gerais e outra de definições de diretórios. A seguir são apresentadas as figuras dessas duas categorias.





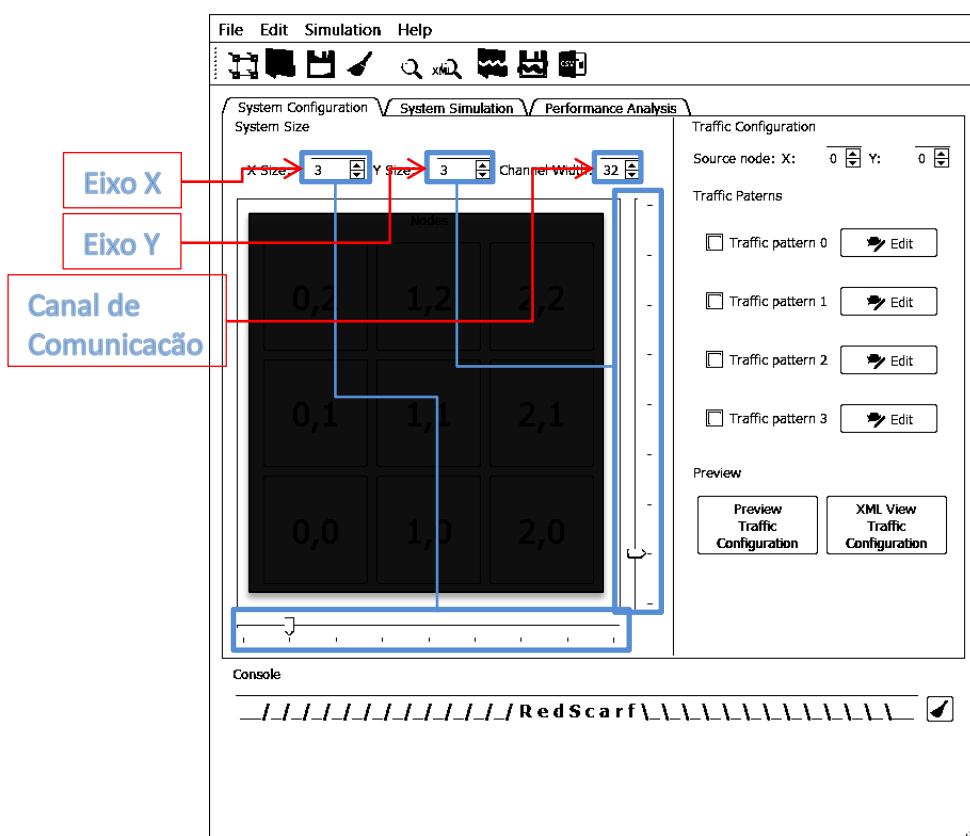
A seguir será apresentado o fluxo de uso da ferramenta para realizar a simulação e a análise dos resultados.

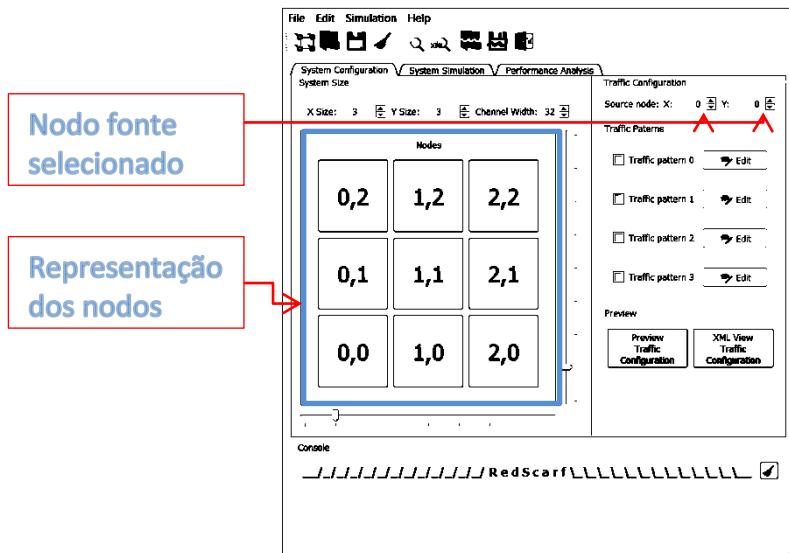
5.2 Definir as dimensões do sistema e a largura do canal de comunicação

Para configurar as dimensões do sistema, basta definir o tamanho do eixo X e Y, sendo que este valor é referente ao número de nodos em cada dimensão. Este valor é definido na primeira aba na parte superior a esquerda, conforme destacado na figura a seguir. Ao lado da definição das dimensões do sistema, está disponível o campo para definir a largura do canal de comunicação da rede em bits, também destacado.

Alternativamente, o usuário pode deslizar os slides que se encontram na horizontal e vertical para configurar os eixos X e Y, respectivamente.

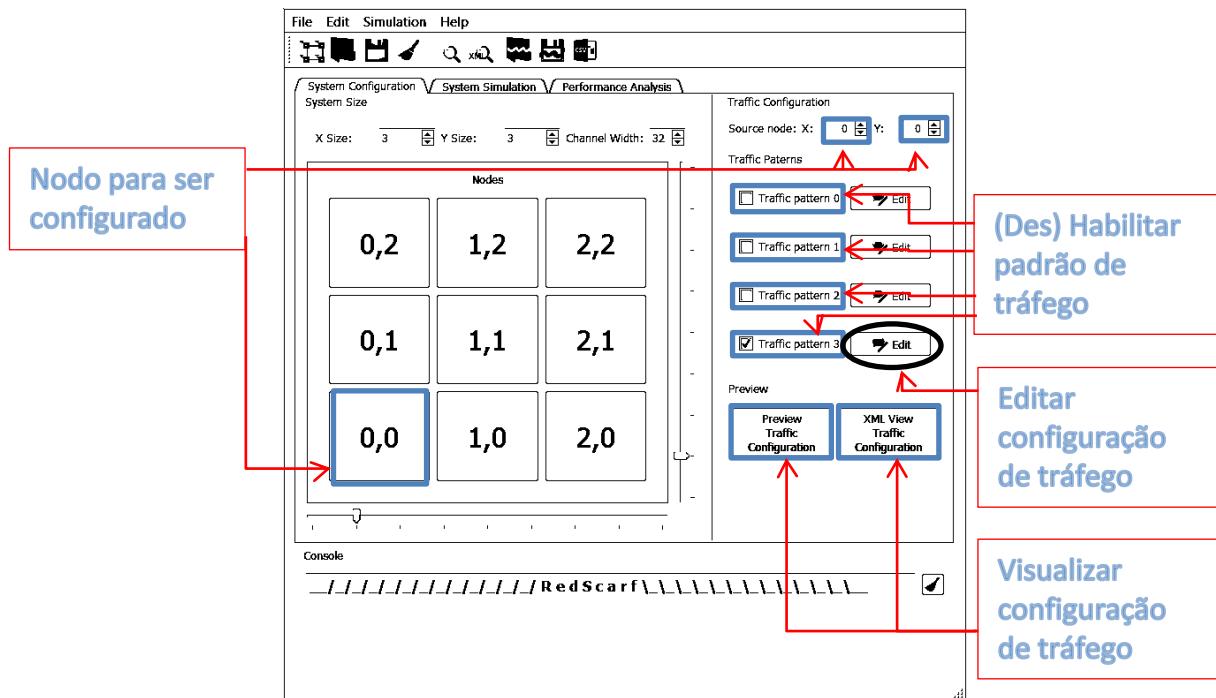
A quantidade de nodos em relação ao dimensionamento realizado é apresentada por meio de botões na interface, os quais servem para selecionar o nodo fonte na configuração de tráfego.





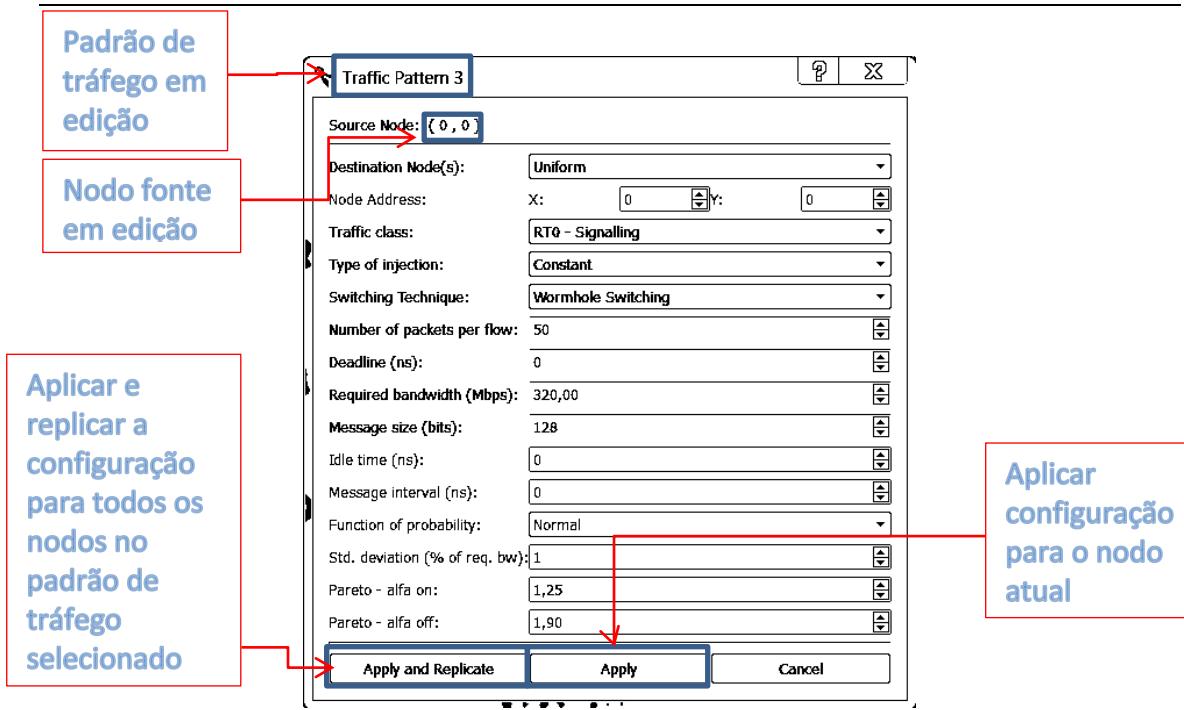
5.3 Definir padrões de tráfego para os nodos na rede

Para definir os padrões de tráfego, o usuário deve selecionar o nodo fonte para configura-lo. A seleção pode ser feita de duas maneiras: clicando sobre o botão que representa o nodo; ou pelas coordenadas X e Y do nodo que se deseja configurar. A figura a seguir apresenta o local onde deve ser feita a seleção do nodo fonte que se deseja configurar.



Após selecionar o nodo fonte, é preciso habilitar o padrão de tráfego a ser configurado. Com o padrão de tráfego habilitado, o usuário deve acessar a opção de edição dos parâmetros de tráfego. Esta opção está disponível no botão “Editar” ao lado direito do padrão de tráfego ativado, conforme mostrado no círculo na figura anterior (que representa a edição do padrão de tráfego 3).

Ao acessar a opção de edição de um padrão de tráfego, é apresentada a tela de configuração dos parâmetros, exibida na figura a seguir. Na janela de configuração de tráfego, os parâmetros a serem definidos são habilitados conforme a distribuição espacial, tipo de injeção e distribuição de probabilidade para distribuições variáveis.



IMPORTANTE: A opção “aplicar e replicar” somente está disponível para o nodo 0,0.

Após configurar os parâmetros do tráfego e concluir a edição escolhendo a opção de aplicar ou aplicar e replicar a configuração, uma mensagem é apresentada no console e é possível visualizar a configuração realizada. A imagem a seguir apresenta a visualização em forma de lista da configuração de tráfego realizada na imagem anterior que foi aplicada e replicada para uma rede 3x3.

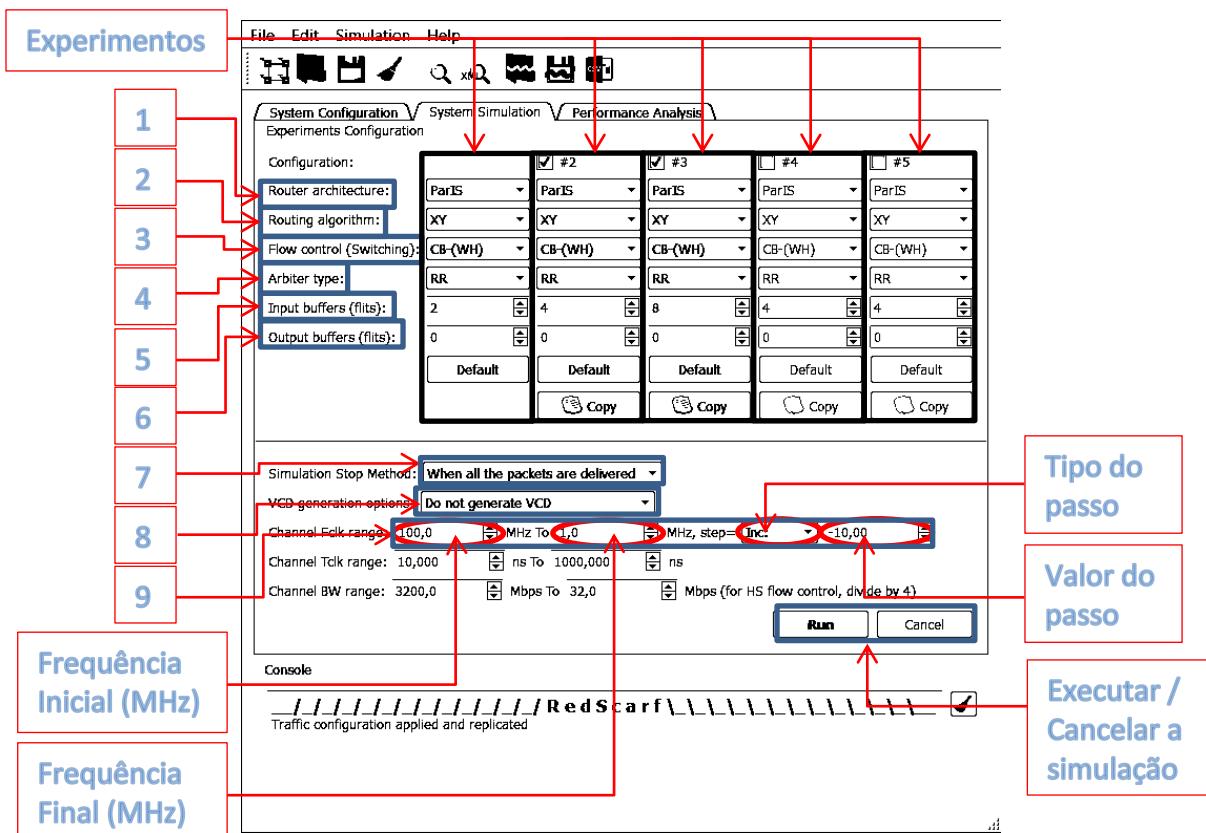
Source Node	Destination Node	Traffic Class	Injection Type	Switching Type	Packet to send (ns)	Deadline (ns)	Required Bandwidth (Mbps)	Message Size (bits)	
0,0	3	Uniform	RTO - Signal...	Constant	Wormhole...	50	0	320	128
(0,1)	3	Uniform	RTO - Signal...	Constant	Wormhole...	50	0	320	128
(0,2)	3	Uniform	RTO - Signal...	Constant	Wormhole...	50	0	320	128
(1,0)	3	Uniform	RTO - Signal...	Constant	Wormhole...	50	0	320	128
(1,1)	3	Uniform	RTO - Signal...	Constant	Wormhole...	50	0	320	128
(1,2)	3	Uniform	RTO - Signal...	Constant	Wormhole...	50	0	320	128
(2,0)	3	Uniform	RTO - Signal...	Constant	Wormhole...	50	0	320	128
(2,1)	3	Uniform	RTO - Signal...	Constant	Wormhole...	50	0	320	128
(2,2)	3	Uniform	RTO - Signal...	Constant	Wormhole...	50	0	320	128

Para editar um padrão de tráfego, basta selecionar o nodo desejado e acessar a opção de edição do padrão de tráfego. Após definir todos os padrões de tráfego, o usuário passa para a segunda aba da aplicação para configurar os experimentos.

5.4 Configurar os experimentos, definir o intervalo de frequência de operação e executar a simulação

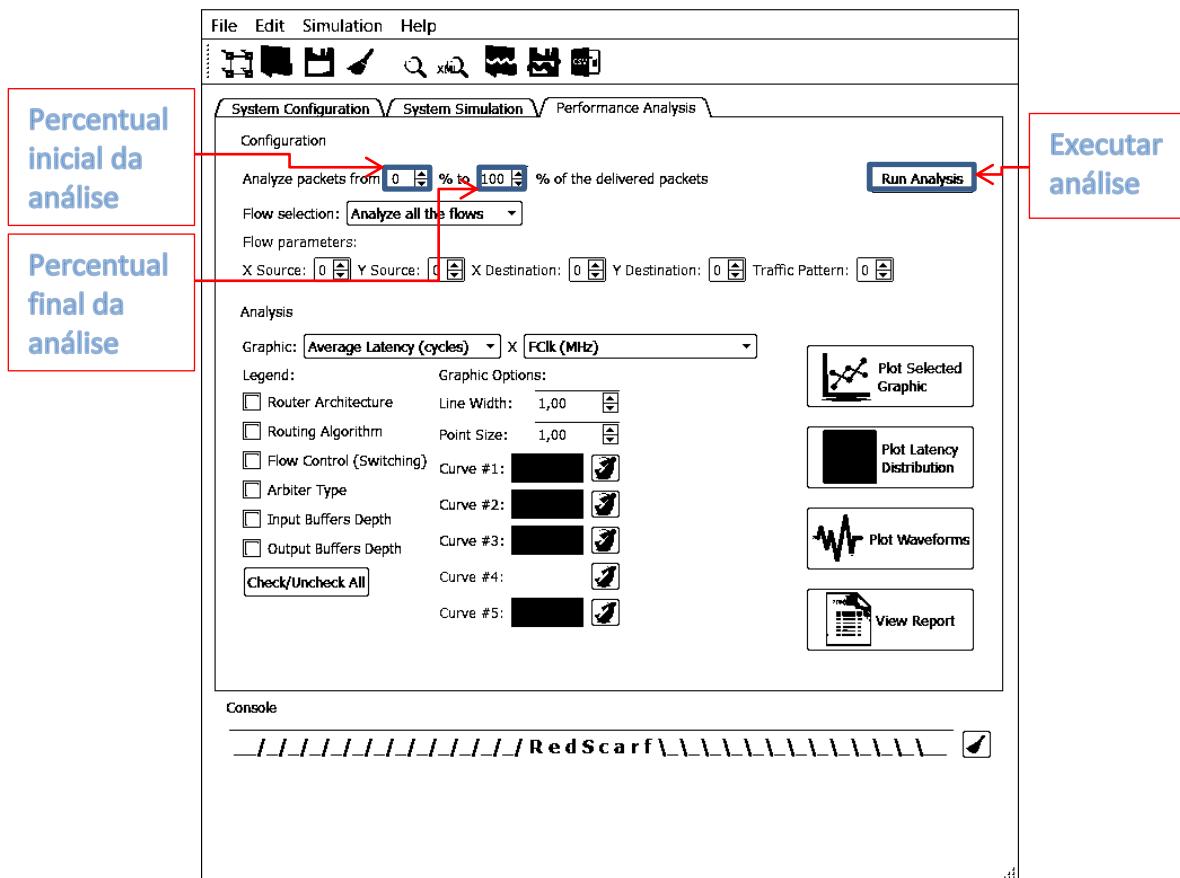
A configuração dos experimentos é realizada na segunda aba da aplicação e permite ao usuário definir até cinco experimentos, sendo que um experimento é obrigatório e os demais são opcionais, basta habilitá-los. Na configuração dos experimentos seis parâmetros são considerados, são eles: (1) Arquitetura do roteador; (2) Algoritmo de roteamento; (3) Controle de fluxo; (4) Tipo de árbitro; (5) Profundidade dos *buffers* de entrada; e (6) Profundidade dos *buffers* de saída. Além dos atributos relacionados ao roteador da rede, também é necessário: (7) definir o método de parada da simulação; (8) habilitar a geração de formas de ondas (se desejado); e (9) definir o intervalo de frequências de operação a serem simulados.

Feita a configuração, deve-se executar a simulação.



5.5 Realizar a análise

Após a simulação, o usuário deve executar a análise, o que é feito na terceira aba da tela principal. O usuário define apenas o intervalo de pacotes para a análise, sendo que os parâmetros são o percentual inicial e final de análise dos pacotes. Assim é possível desconsiderar um percentual inicial (aquecimento) e final (drenagem), se for da preferência do utilizador. A imagem a seguir destaca o local destes parâmetros e da execução da simulação.



A execução da análise pode demorar um pouco, pois realiza a leitura de todos os *logs* da simulação e gera os relatórios da análise. Durante a análise, o ponteiro do mouse fica ocupado e alguns componentes gráficos são desabilitados. Após a execução da análise os resultados estão prontos para serem visualizados.

5.6 Visualizar os resultados

A visualização dos resultados é feita após a análise. Estão disponíveis: a visualização de gráficos, exibição de diagrama de formas de ondas (desde que uma ferramenta externa esteja devidamente configurada) e relatórios da simulação.

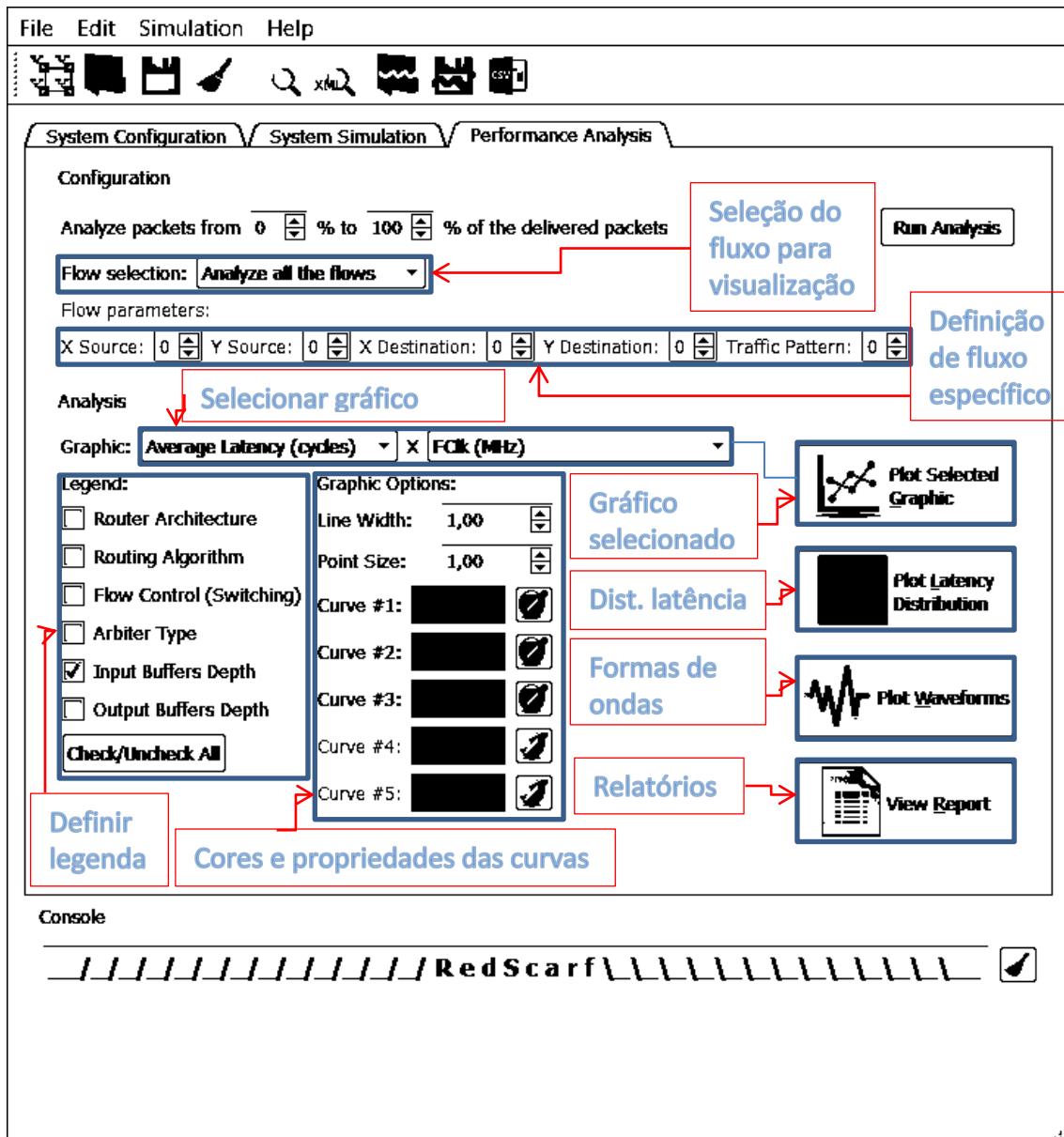
O usuário pode visualizar todos os fluxos (opção padrão), escolher a visualização de acordo com uma classe de tráfego e ainda pode estipular um fluxo específico. Todas as visualizações são feitas de acordo com esta configuração, exceto as formas de ondas que exibem os sinais da rede e devem ser visualizadas por ferramenta externa (devidamente configurada), inclusive sem a realização da análise, pois basta estar definida a geração do arquivo VCD no momento da simulação. O usuário só deve escolher qual experimento deseja visualizar em uma lista dos experimentos disponíveis em que a descrição indica o diretório de trabalho, os parâmetros do roteador e a frequência de operação.

Para diferenciar os experimentos na visualização, estão disponíveis seis opções de legenda, conforme os parâmetros da configuração dos experimentos. Estas opções da legenda podem ser combinadas.

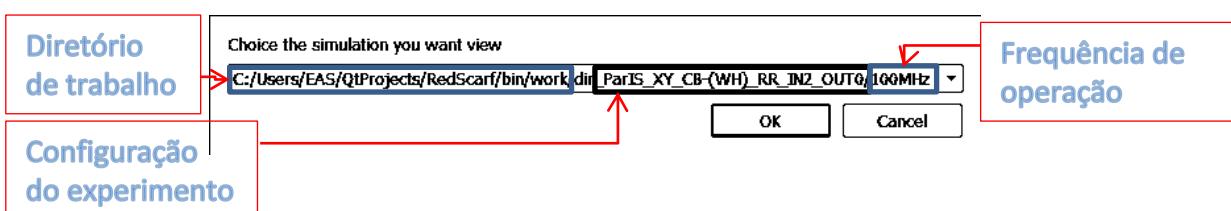
O usuário pode selecionar um gráfico para visualizar, dentre as opções disponíveis, visualizar a distribuição de latência dos experimentos desejados e visualizar os relatórios da simulação.

Para diferenciar as curvas dos experimentos nos gráficos gerados, há a opção de definir a cor de cada curva, que está indexada de acordo com a ordem dos experimentos ativos da simulação. Além disso, é possível definir o tamanho dos pontos da curva e a largura dos traços.

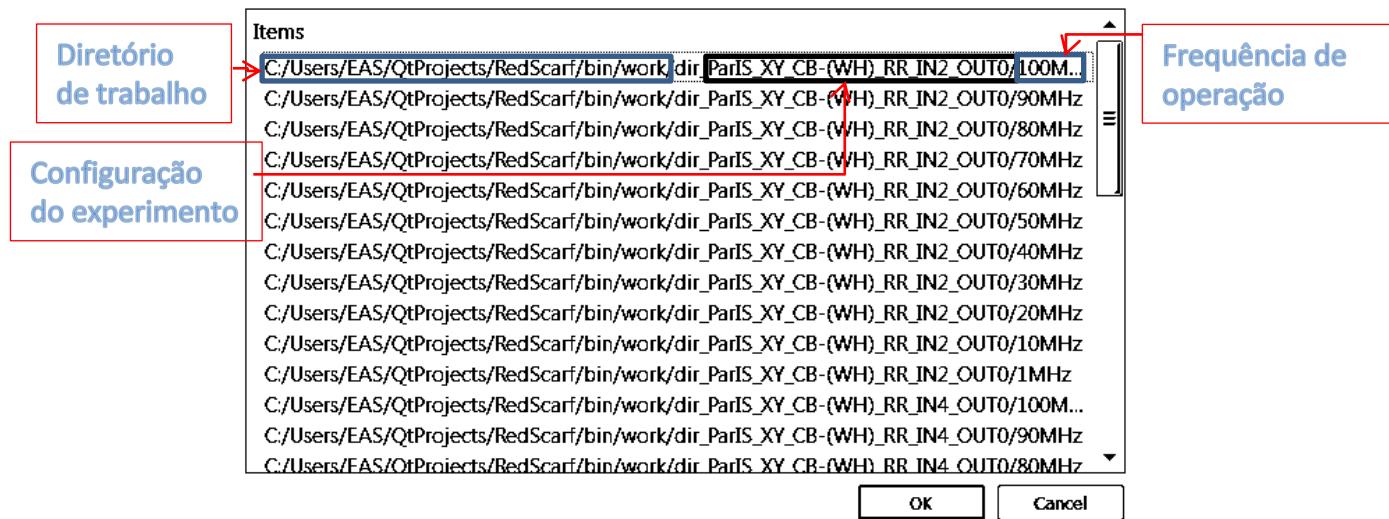
Na visualização da distribuição de latências, será solicitada ao usuário a seleção de quais distribuições deseja visualizar. Os itens possuem a descrição do diretório do de trabalho, a descrição da configuração relativa aos parâmetros do roteador e a frequência de operação.



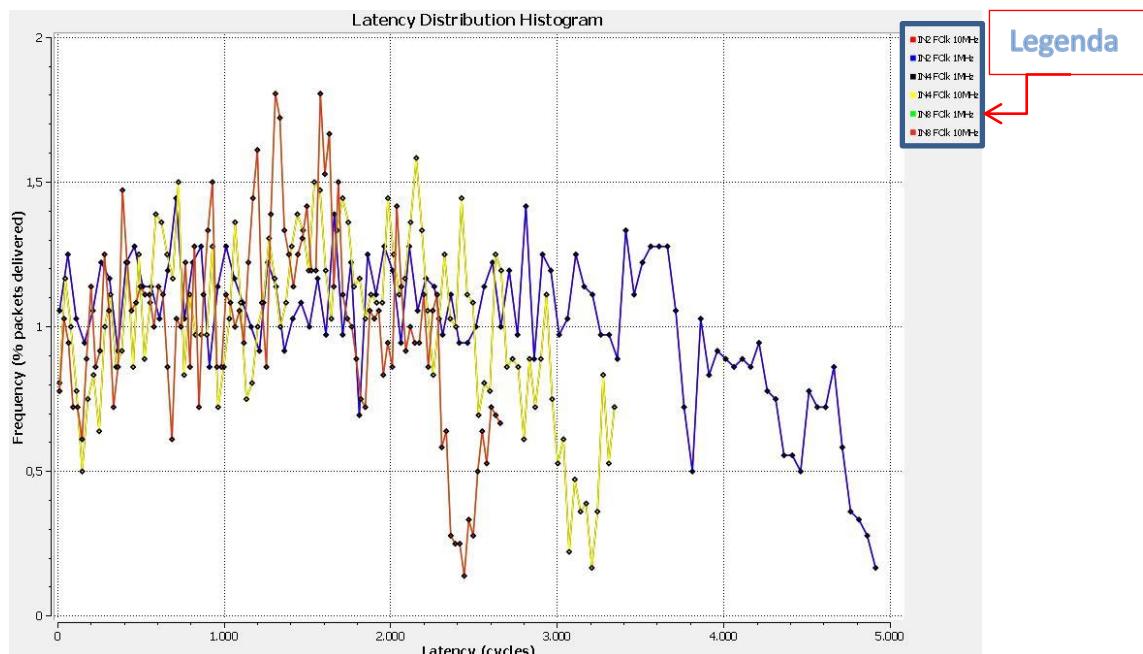
A lista para selecionar as formas de ondas é no formato apresentado na figura a seguir.



O formato da lista para selecionar as distribuições de latências é apresentado na figura a seguir. É uma lista de seleção de múltiplos itens caso necessário visualizar mais de uma distribuição no mesmo gráfico.



Um exemplo de gráfico de distribuição de latência é apresentado na figura a seguir.



Na visualização do relatório, o usuário pode alterar qual experimento atualmente está sendo visualizado na opção na parte inferior da exibição do relatório, conforme apresentado na figura a seguir.

FCLK (MHz)	Packets Delivered	Offered Traffic (norm)	Offered Traffic (Mbps/n)	Accepted Traffic (f/c/n)	Accepted Traffic (Mbps/n)	Ideal Avg Latency (cycles)	Ideal Avg Latency (ns)	Avg Latency (cycles)	Avg Latency (ns)	Min Latency (cycles)	Min Latency (ns)	Max Latency (cycles)	Max Latency (ns)	Std Dev Latency
100	3600	0.1	320	0.0975	312	14	140	28.8	288.3	11	110	117	1170	15.4
90	3600	0.1111	320	0.1081	311.3	14	155.6	29.3	325.8	11	122.2	121	1344.4	15.8
80	3600	0.125	320	0.1211	310	14	175	30.5	381.8	11	137.5	136	1700	17.4
70	3600	0.1429	320	0.1375	308	14	200	32.3	461.7	11	157.1	168	2400	19.6
60	3600	0.1667	320	0.1593	305.9	14	233.3	36	600.5	11	183.3	193	3216.7	23.6
50	3600	0.2	320	0.189	302.4	14	280	42.7	854.4	11	220	226	4520	30
40	3600	0.25	320	0.2315	296.3	14	350	64.4	1610.3	11	275	312	7800	51.6
30	3600	0.3333	320	0.2837	272.4	14	466.7	193.1	6438.1	11	366.7	734	24466.7	184.6
20	3600	0.5	320	0.2927	187.3	14	700	790	39499.5	11	550	1972	98600	491.5
10	3600	1	320	0.2957	94.6	14	1400	1614.3	161429	11	1100	3415	341500	889.8
1	3600	1	320	0.2957	9.5	14	14000	1614.3	1.6142...	11	11000	3415	3.415e...	889.8

III

Experiment: IM4

Seleção do experimento do relatório

Também pode ser gerado o relatório em um arquivo no formato CSV que apresenta todos os experimentos no mesmo plano e pode ser visualizado por outra ferramenta (ex.: MS Excel).

5.7 Exercício: Cenário de exemplo

Para ilustrar a utilização da ferramenta será apresentado um exercício e o passo-a-passo da resolução para a configuração, simulação, análise e visualização dos resultados.

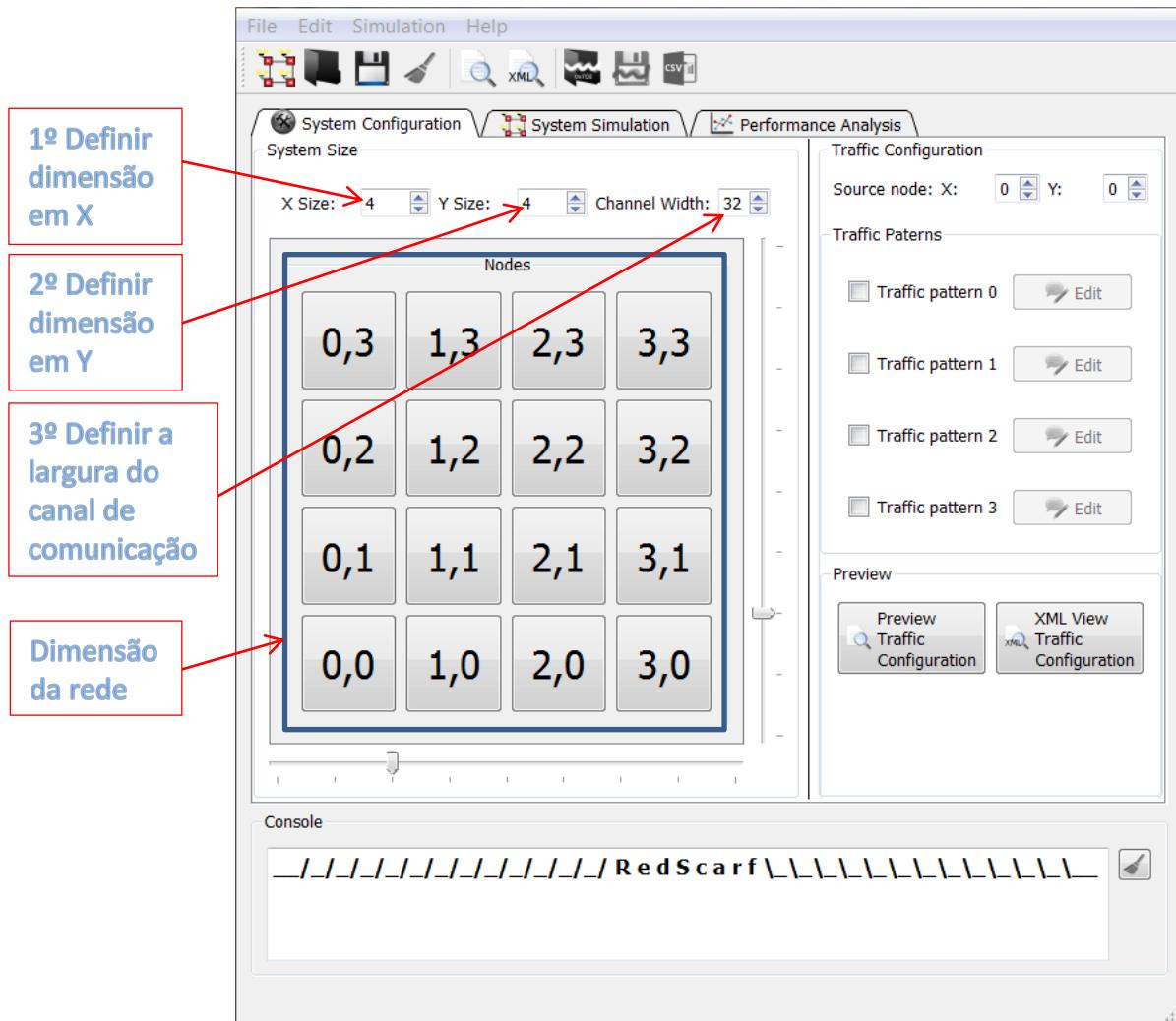
Enunciado

Criar uma Rede-em-Chip numa malha 4x4 de 32 bits e definir para cada nodo terminal um padrão de tráfego com distribuição espacial Uniforme, injeção constante de pacotes, chaveamento *Wormhole*, com 50 pacotes por fluxo, sem prazo de entrega definido, largura de banda requerida de 320 Mbps e tamanho de mensagem de 128 bits. Realizar 3 experimentos com a rede SoCIN sem canais virtuais (roteador ParIS), algoritmo de roteamento XY, controle de fluxo baseado em créditos (para chaveamento *Wormhole*), árbitro *Round-robin* e sem *buffers* de saída, variando a profundidade dos *buffers* de entrada em 2, 4 e 8 *flits* nos experimentos. Rodar a simulação com frequência de operação de 5 a 100 MHz com passo de +5 MHz até todos os pacotes serem entregues. Analisar todos os pacotes e apresentar os resultados para todos os fluxos considerando: (i) a latência média (em ciclos) pela frequência de operação; (ii) a latência média (em ciclos) pelo tráfego oferecido; (iii) o tráfego aceito pela frequência de operação; (iv) o tráfego aceito pelo tráfego oferecido; (v) a distribuição de latência na menor frequência antes do ponto de saturação de todas as três configurações; e (vi) a distribuição de latência na maior frequência após o ponto de saturação de todas as três configurações. Identificar os resultados pela profundidade dos *buffers* de entrada e pela arquitetura do roteador. Na análise, indicar os pontos de saturação da rede em cada experimento, citando os valores das frequências de operação próximas a esses pontos. Também analisar o efeito da saturação sobre a distribuição de latências.

Resolução

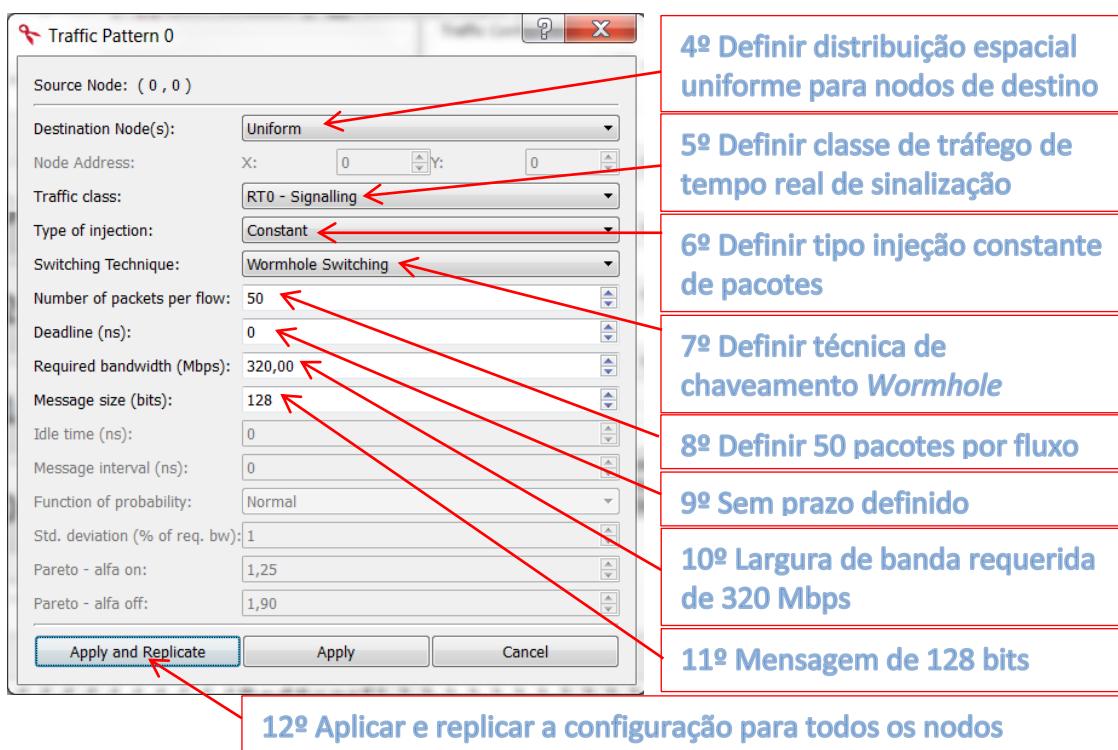
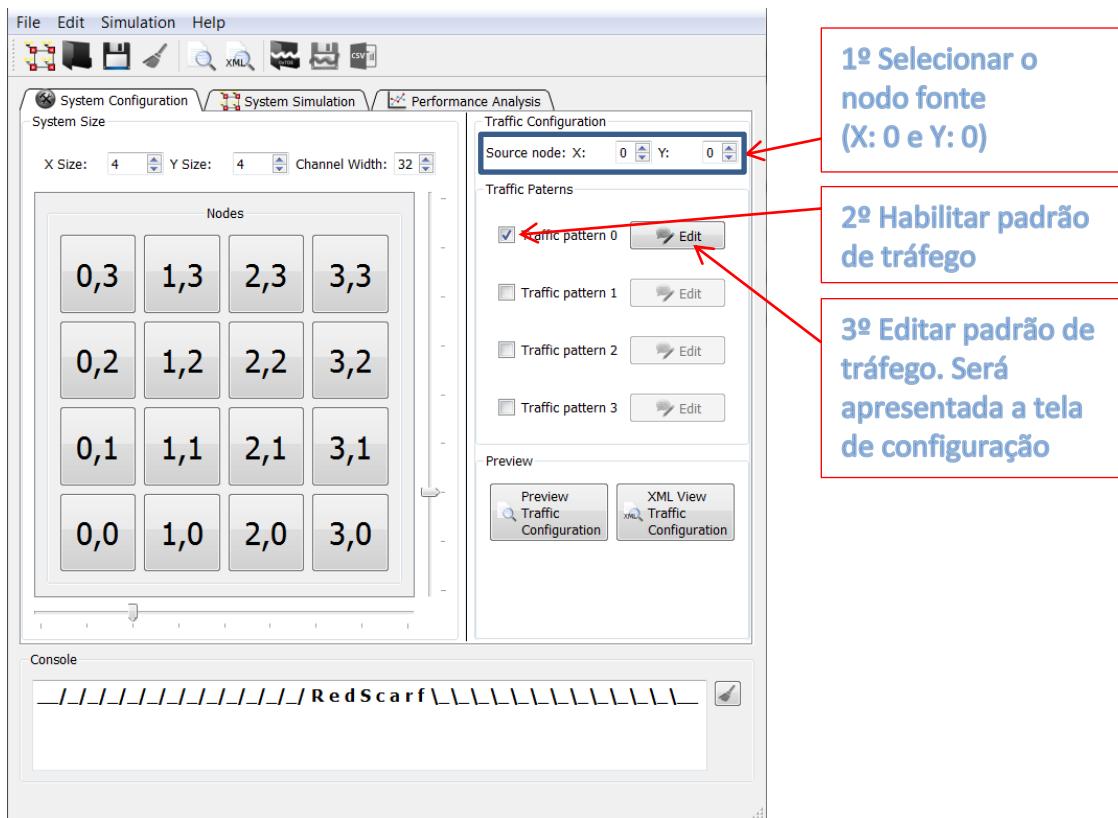
A resolução deste exercício será apresentada seguindo os passos descritos no início deste capítulo.

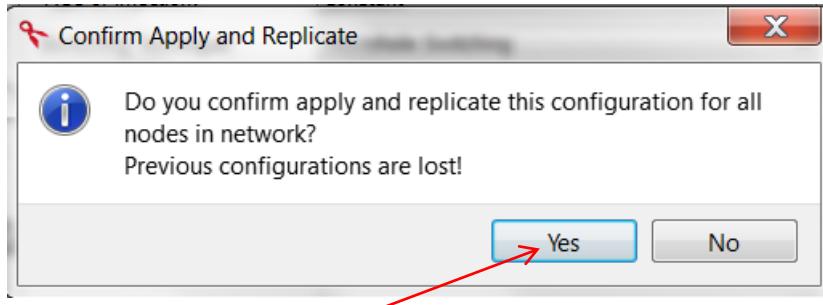
1-) Definir as dimensões do sistema e a largura do canal de comunicação



2-) Definir padrões de tráfego para os nodos na rede

Primeiramente deve ser selecionado o nodo que se deseja configurar. Como é para definir a mesma configuração para todos os nodos, será selecionado o nodo (0, 0), pois este é o único permite replicar a sua configuração para os demais nodos. Será habilitado o padrão de tráfego 0, mas poderia ser qualquer um dos demais.



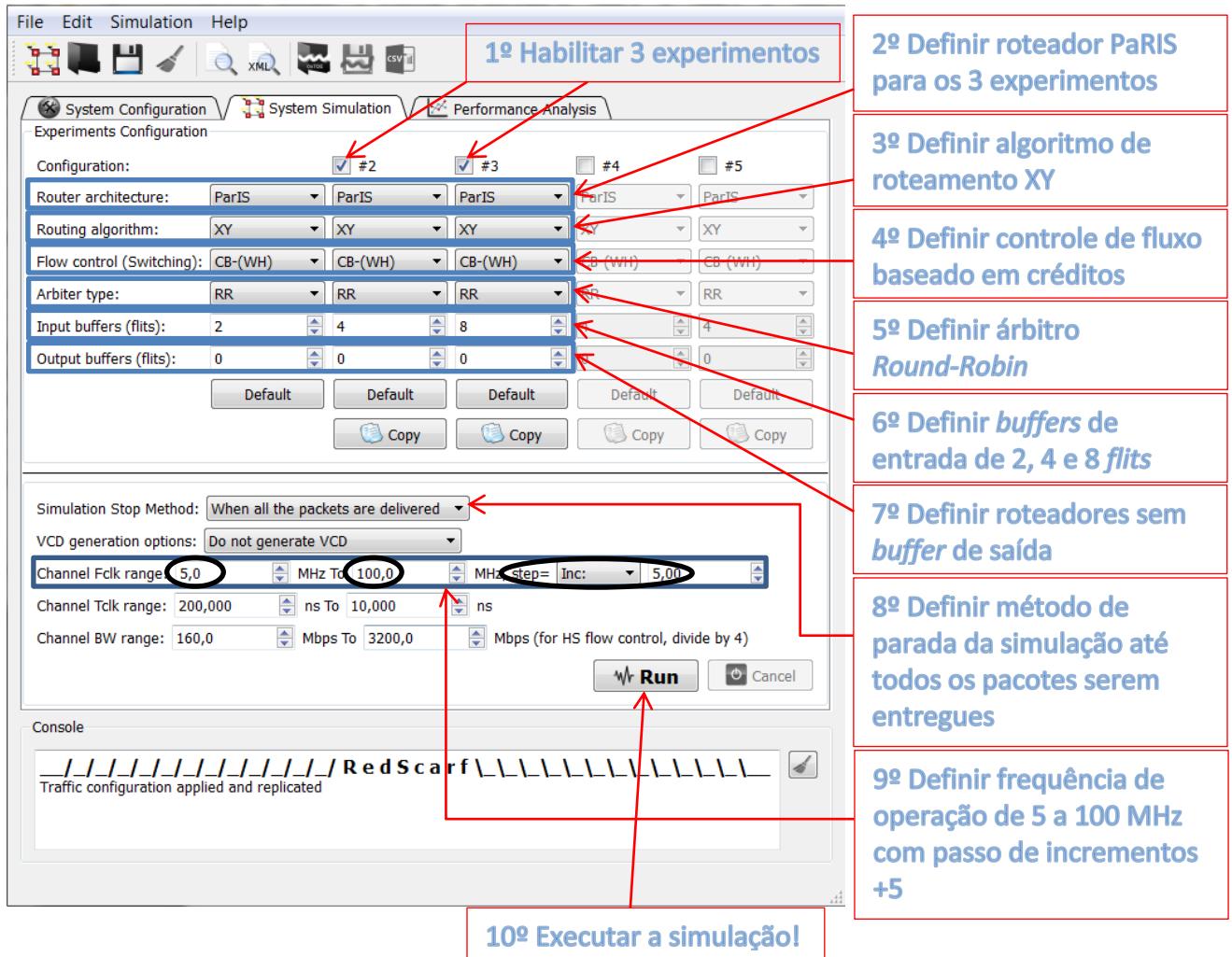


13º Confirmar replicação da configuração

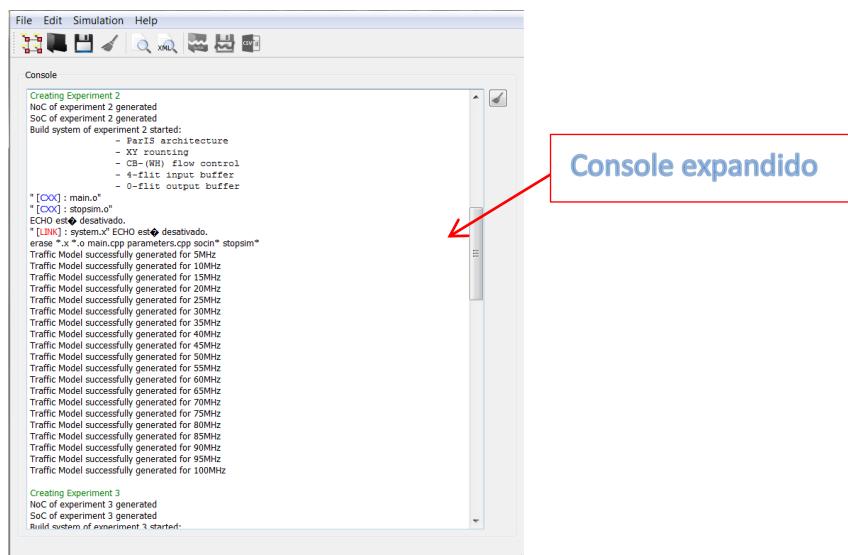
Após replicar a configuração, é possível visualizar a configuração realizada em forma de lista ou árvore, apenas para confirmar se a configuração está correta. Neste caso, deve ser exatamente como apresentado nas figuras a seguir.

Source Node	Destination Node	Traffic Class	Injection Type	Switching Type	Packets to send	Deadline (ns)	Required Bandwidth (Mbps)	Message Size (bits)
(0,0): 0	Uniform	RT0 - Signalling	Constant	Wormhole Switching	50	0	320	128
(0,1): 0	Uniform	RT0 - Signalling	Constant	Wormhole Switching	50	0	320	128
(0,2): 0	Uniform	RT0 - Signalling	Constant	Wormhole Switching	50	0	320	128
(0,3): 0	Uniform	RT0 - Signalling	Constant	Wormhole Switching	50	0	320	128
(1,0): 0	Uniform	RT0 - Signalling	Constant	Wormhole Switching	50	0	320	128
(1,1): 0	Uniform	RT0 - Signalling	Constant	Wormhole Switching	50	0	320	128
(1,2): 0	Uniform	RT0 - Signalling	Constant	Wormhole Switching	50	0	320	128
(1,3): 0	Uniform	RT0 - Signalling	Constant	Wormhole Switching	50	0	320	128
(2,0): 0	Uniform	RT0 - Signalling	Constant	Wormhole Switching	50	0	320	128
(2,1): 0	Uniform	RT0 - Signalling	Constant	Wormhole Switching	50	0	320	128
(2,2): 0	Uniform	RT0 - Signalling	Constant	Wormhole Switching	50	0	320	128
(2,3): 0	Uniform	RT0 - Signalling	Constant	Wormhole Switching	50	0	320	128
(3,0): 0	Uniform	RT0 - Signalling	Constant	Wormhole Switching	50	0	320	128
(3,1): 0	Uniform	RT0 - Signalling	Constant	Wormhole Switching	50	0	320	128
(3,2): 0	Uniform	RT0 - Signalling	Constant	Wormhole Switching	50	0	320	128
(3,3): 0	Uniform	RT0 - Signalling	Constant	Wormhole Switching	50	0	320	128

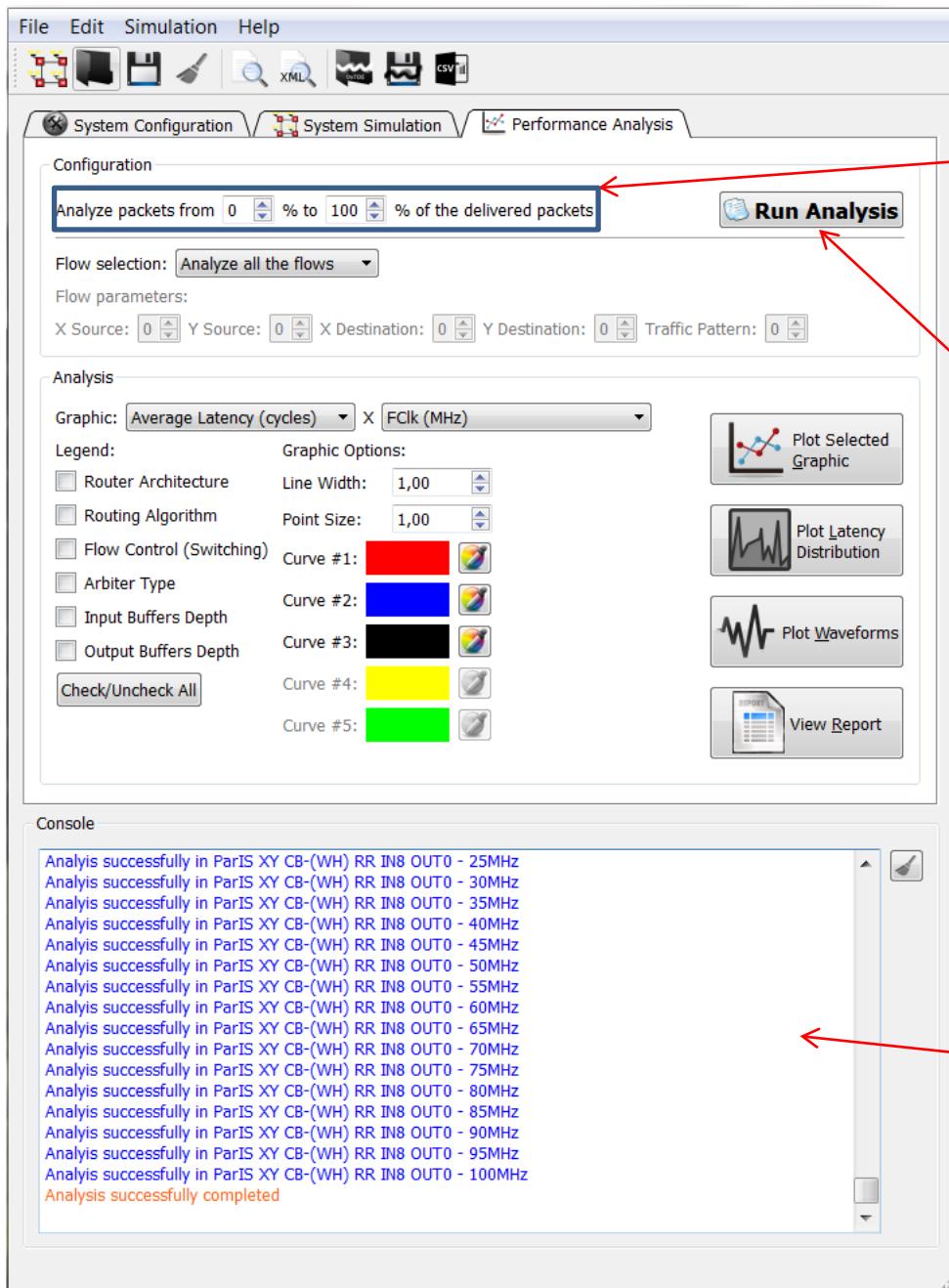
3-) Configurar os experimentos, definir o intervalo de frequência de operação e executar a simulação



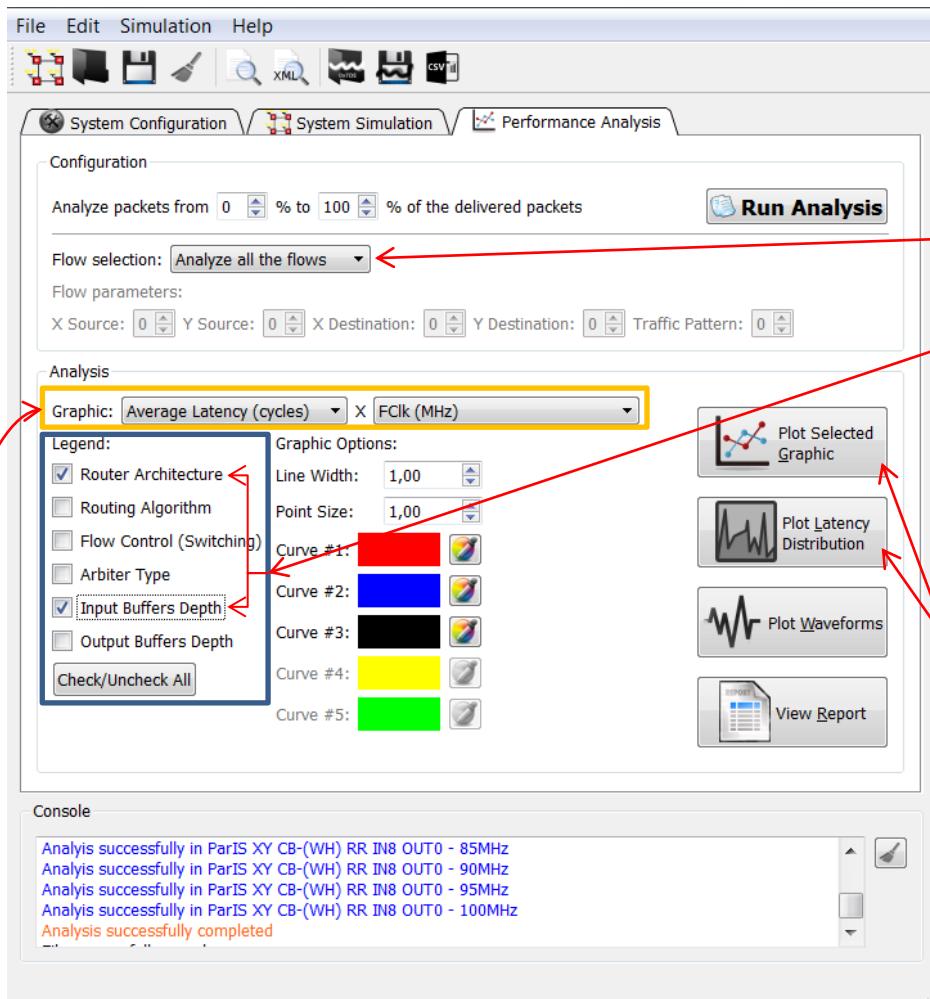
A simulação demora um pouco. É possível expandir o console para melhor visualizar as mensagens do progresso da simulação.



4-) Realizar a análise



5-) Visualizar os resultados



1º Resultados para todos os fluxos

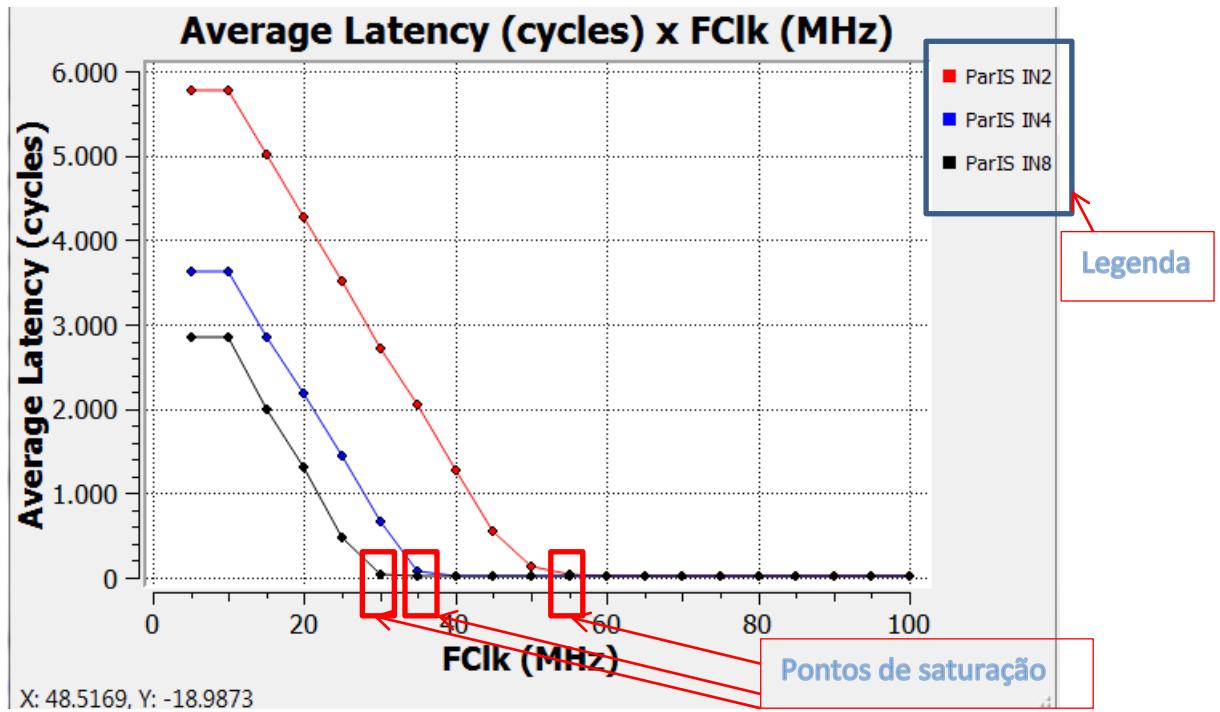
2º Identificar resultados pela arquitetura do roteador e profundidade dos buffers de entrada

4º Visualizar gráfico selecionado ou distribuição de latência

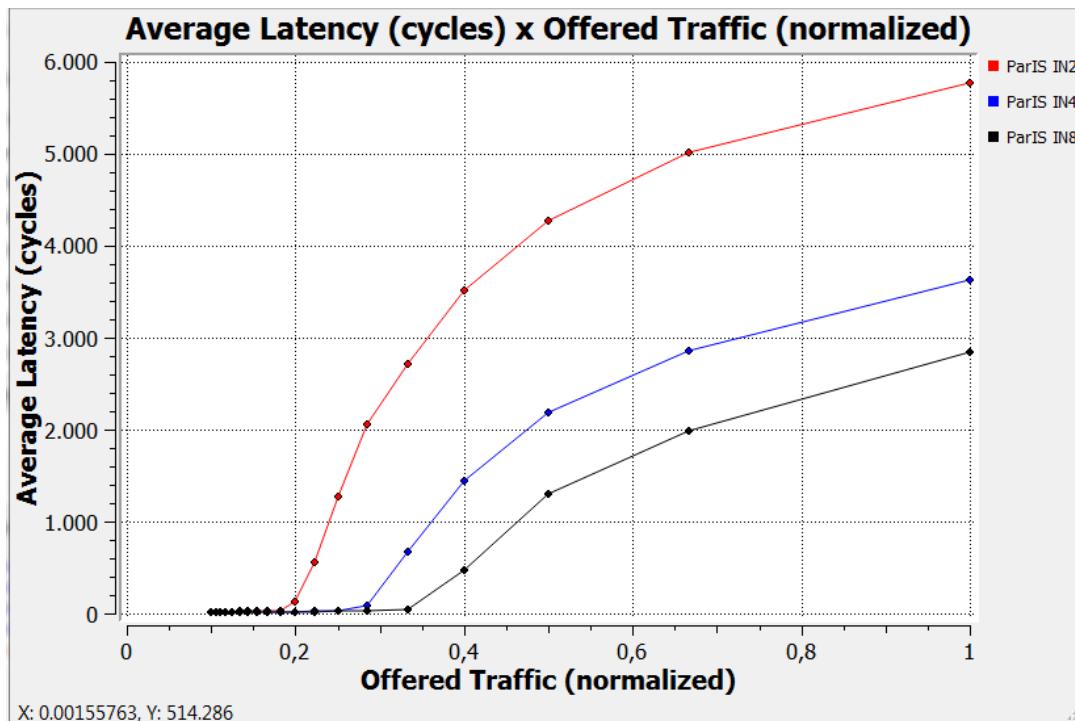
3º Selecionar o gráfico

- Latência média X Frequência de operação
- Latência média X Tráfego oferecido
- Tráfego aceito X Frequência de operação
- Tráfego aceito X Tráfego oferecido
- Distribuição de latência na menor frequência antes do ponto de saturação para os três experimentos
- Distribuição de latência na maior frequência após o ponto de saturação para os três experimentos

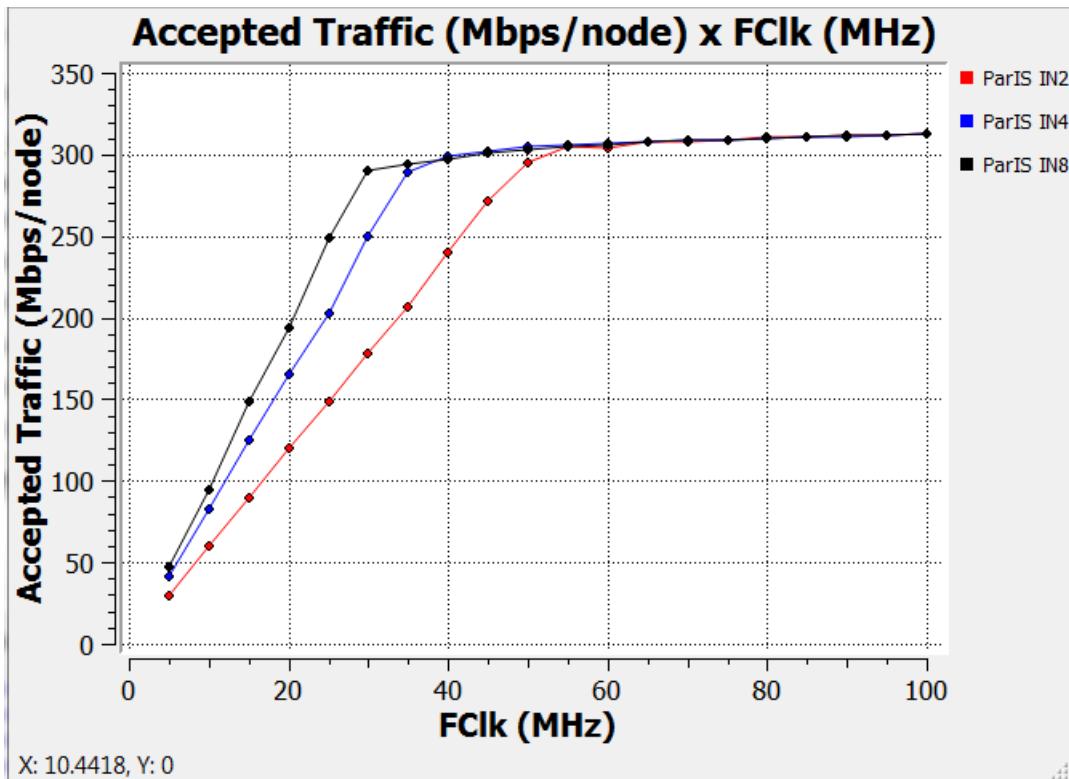
(i) Latência média X Frequência de operação



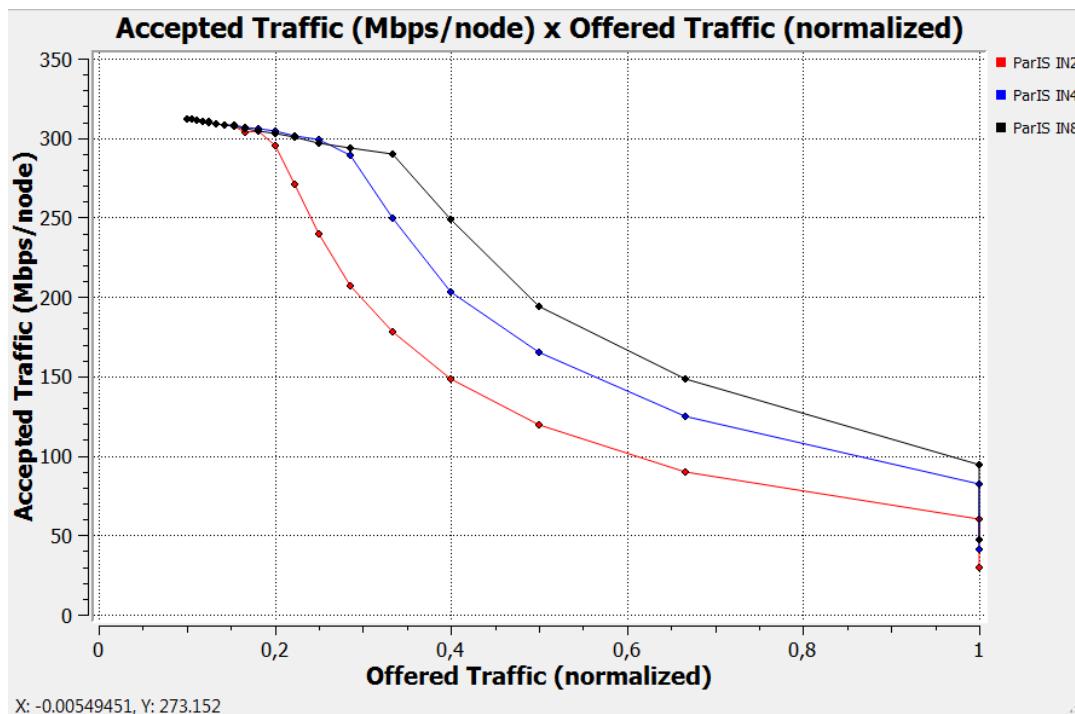
(ii) Latência média X Tráfego oferecido



(iii) Tráfego aceito X Frequência de operação



(iv) Tráfego aceito X Tráfego oferecido



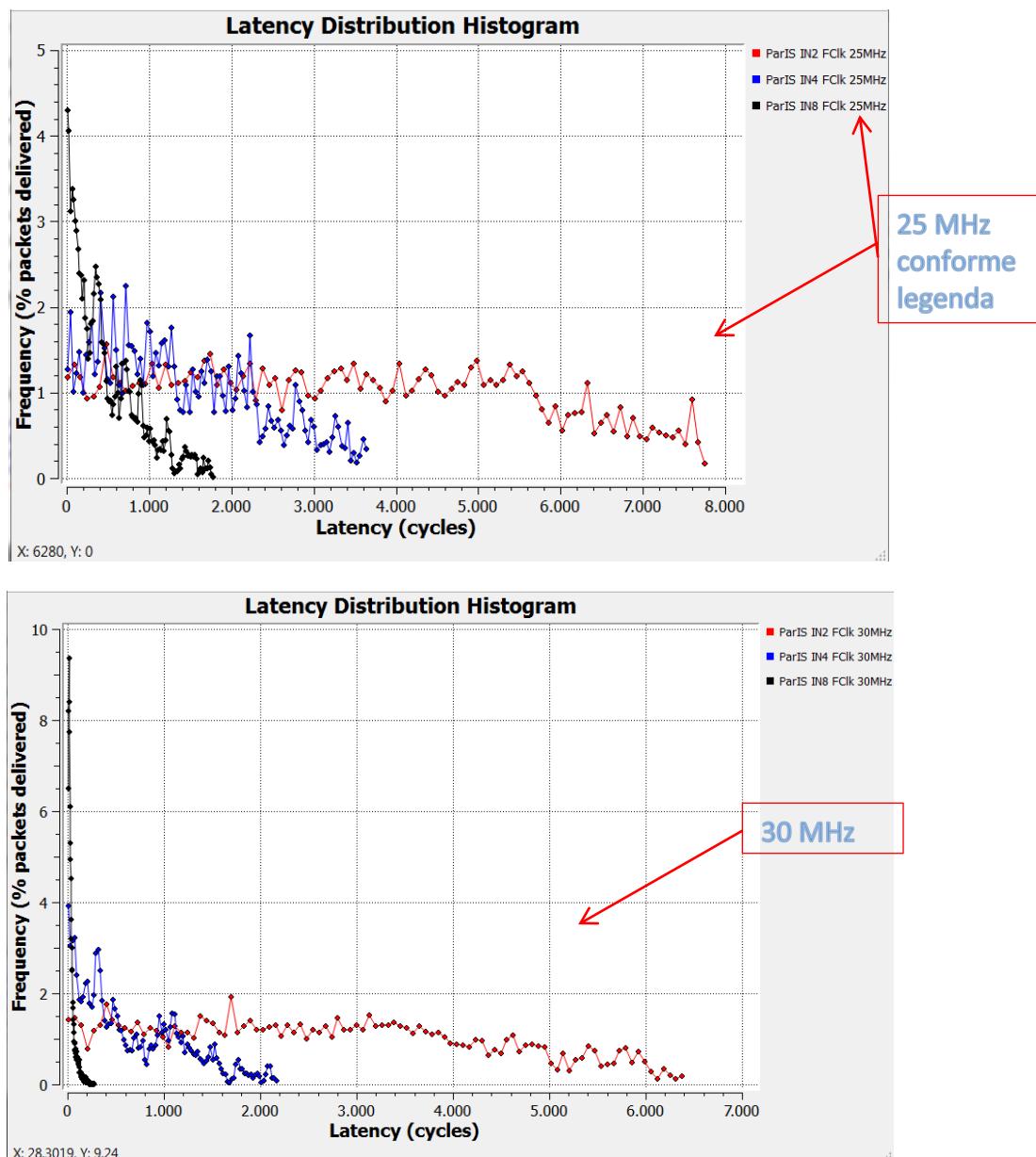
Para as distribuições de latência é necessário selecionar os experimentos e as frequências de operação desejadas. A figura a seguir apresenta a tela de seleção.

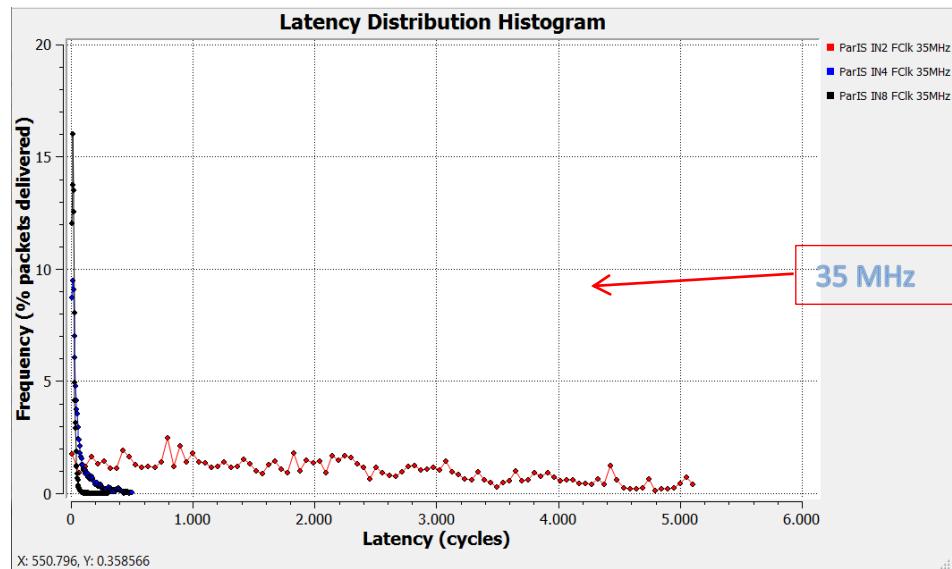


Note que os itens possuem o diretório de trabalho, informações sobre os experimentos e a frequência de operação.

Serão apresentados os gráficos de distribuição de latência com as curvas dos três experimentos com frequência de operação de 25 à 55 MHz, a qual é a faixa de frequências em que se encontram os pontos de saturação dos experimentos,

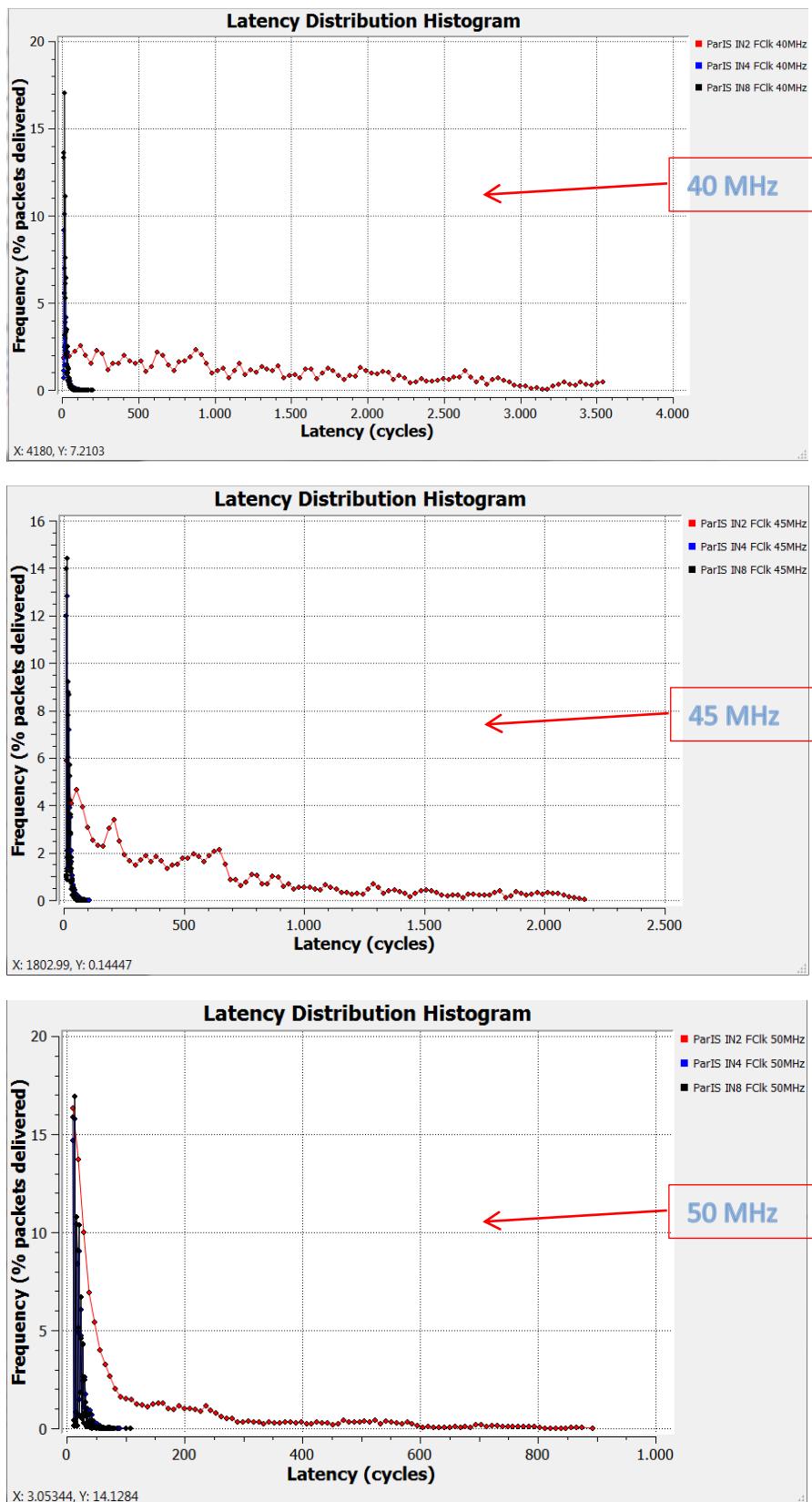
conforme visto no gráfico do primeiro item (i) Latência Média X Frequência de operação, deste exercício .

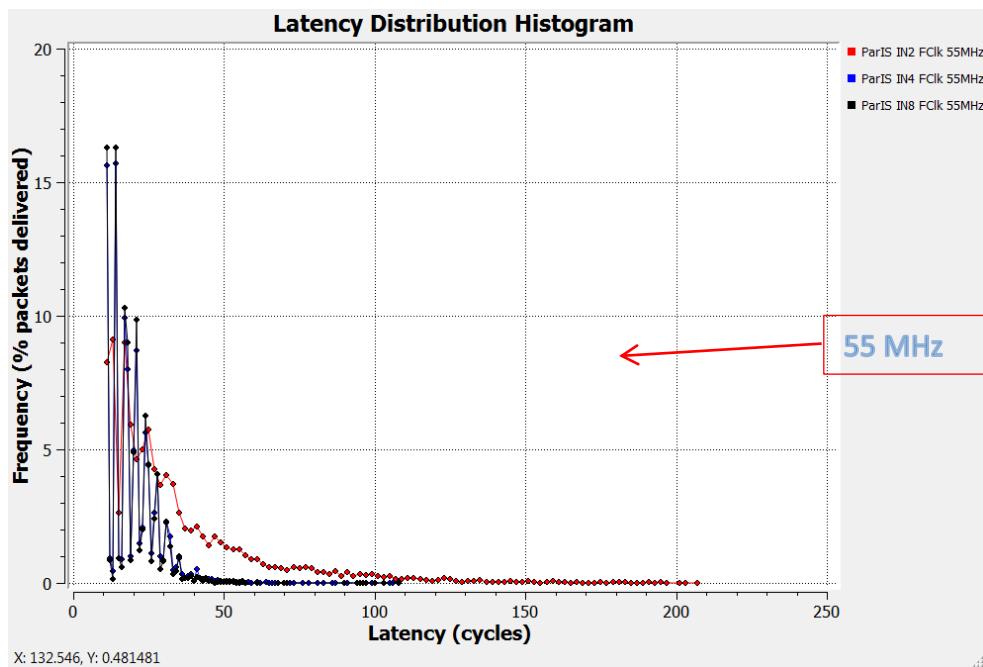




Observa-se, no primeiro gráfico, que a rede está bastante saturada para todos as configurações operando a 25 MHz, dado o espalhamento e os valores das latências (muitos pacotes com latência de várias centenas a milhares de ciclos). Já com 30 MHz, a configuração com 8 *flits* nos *buffers* de entrada (curva preta) apresenta uma concentração de pacotes entregues com latência próxima da mínima, enquanto as demais configurações apresentam um grande espalhamento de latências. Com 35 MHz, a configuração com *buffers* de 4 *flits* (curva azul) também apresenta baixo espalhamento de latências, enquanto a com *buffers* de 2 *flits* (curva vermelha) ainda exibe latências da ordem de milhares de ciclos.

Os gráficos a seguir ilustra o efeito do aumento da frequência de operação na rede, em especial para a configuração com *buffers* de 2 *flits*. Percebe-se que a latência com 2 *flits* nos *buffers* de entrada (curva vermelha), começa a melhorar à 50 MHz, pois as latências estão abaixo de 1000 ciclos.





De todos os gráficos de distribuição de latência, conclui-se que o espalhamento de latências é inversamente proporcional à profundidade dos *buffers* e à frequência de operação da rede.

- (v) Distribuição de latência na menor frequência antes do ponto de saturação para os três experimentos

Com os gráficos das distribuições de latências próximas aos pontos de saturação dos experimentos já apresentados, pode-se concluir que a menor frequência de operação antes da saturação para os experimentos é de 30, 35 e 55 MHz para as configurações de 8, 4 e 2 *flits*, respectivamente.

- (vi) Distribuição de latência na maior frequência após o ponto de saturação para os três experimentos

E para a maior frequência de operação após a saturação para os experimentos é de 25, 30 e 50 MHz para os experimentos de 8, 4 e 2 *flits*, respectivamente, pois são os pontos em que se deve aprimorar a análise para se chegar a um valor mais preciso.

6 Referências Bibliográficas

- BRUCH, Jaison Valmor; PIZZONI, Magnos Roberto; ZEFERINO, Cesar Albenes. BrownPepper: a SystemC-based simulator for performance evaluation of Networks-on-Chip. In: IFIP/IEEE INT. CONFERENCE ON VERY LARGE SCALE INTEGRATION (VLSI-SOC 2009), 17., 2009, Florianópolis. **Proceedings...** [S.l.: s.n.], 2009. p. 1-4.
- ZEFERINO, Cesar Albenes; SANTO, F. G. M. E.; SUSIN, A. A. ParIS: A Parameterizable Interconnect Switch for Networks-on-Chip. In: 17th (INT.) SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN (SBCCI), 2004, Porto de Galinhas. **Proceedings...** New York: ACM Press, 2004. p. 204-209.