UNIVERSIDADE FEDERAL DO RIO GRANDE - FURG
INSTITUTO DE CIÊNCIAS COMPUTACIONAIS
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

Gustavo Lameirão de Lima

DESENVOLVIMENTO DE UM SIMULADOR DE NOCS EM UM AMBIENTE MULTIAGENTE

Rio Grande
2019

Gustavo Lameirão de Lima

DESENVOLVIMENTO DE UM SIMULADOR DE NOCS EM UM AMBIENTE MULTIAGENTE

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia de Computação, da Universidade Federal do Rio Grande - FURG, como requisito parcial para a obtenção do grau de Mestre.

Orientador: Prof. Dr. Odorico Machado Mendizabal

Coorientador: Prof. Dr. Eduardo Wenzel Brião

Rio Grande

2019

Ficha catalográfica

L732d Lima, Gustavo Lameirão de.
Desenvolvimento de um simulador de NoCs em um ambiente multiagente / Gustavo Lameirão de Lima. – 2019.
110 f.

Dissertação (mestrado) – Universidade Federal do Rio Grande – FURG, Programa de Pós-Graduação em Engenharia de Computação, Rio Grande/RS, 2019.
Orientador: Dr. Odorico Machado Mendizabal.
Coorientador: Dr. Eduardo Wenzel Brião.

1. Network-on-Chip 2. NoC 3. SMA 4. Sistemas Multiagente 5. Simulação I. Mendizabal, Odorico Machado II. Brião, Eduardo Wenzel III. Título.

CDU 004.738.5

Catalogação na Fonte: Bibliotecário José Paulo dos Santos CRB 10/2344

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO RIO GRANDE
CENTRO DE CIÊNCIAS COMPUTACIONAIS
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO
CURSO DE MESTRADO EM ENGENHARIA DE COMPUTAÇÃO

DISSERTAÇÃO DE MESTRADO

Desenvolvimento de um simulador de NOCS em um ambiente multiagente

Gustavo Lameirão de Lima

Banca examinadora:

Profª. Drª. Raquel de Miranda Barbosa

Prof. Dr. Ewerson Luiz de Souza Carvalho

Prof. Dr. Cleo Zanella Billa

Prof. Dr. Odorico Machado Mendizabal
Orientador

Prof. Dr. Eduardo Wenzel Brião
Co-orientador

Agradecimentos

A desenvolvimento desta dissertação de mestrado não seria possível sem o apoio de várias pessoas.

Gostaria de agradecer a minha família, em especial aos meus pais, Luiz Fernando Moura de Lima e Mara Rosani Lameirão de Lima por todo o amor, não apenas para o desenvolvimento deste estudo, mas em toda a minha vida.

Agradeço também aos meus amigos e a minha namorada. Por todo o apoio e motivação para que eu conseguisse chegar no objetivo almejado.

Agradeço imensamente ao meu orientador, Prof. Dr. Odorico Machado Mendizabal, e também ao meu coorientador, Prof. Dr. Eduardo Wenzel Brião. Sou grato por toda as orientações, ideias, sugestões e revisões que fizeram com que o que era inicialmente uma simples ideia, se tornasse um trabalho de mestrado.

Agradeço a Prof.ª Diana Francis Adamatti e a Prof.ª Cristina Meinhardt por todo o acompanhamento, desde o início do mestrado, e por continuar contribuindo com o trabalho até o fim.

Agradeço a CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) pelo apoio financeiro para auxiliar a realização deste estudo.

Resumo

A evolução dos processos de fabricação de circuitos integrados permite que sistemas grandes, implementando diversas funcionalidades, sejam construídos dentro de um mesmo chip. Este conceito é conhecido como SoC (*System-on-a-Chip*). Porém, estes sistemas complexos requerem o desenvolvimento de mecanismos de comunicação entre as partes que compõem esse sistema. A forma de comunicação entre componentes de um SoC, feita com o auxílio de roteadores, é conhecida como NoC (*Network-on-Chip*). Para explorar melhores opções de projeto durante a criação de uma NoC, o uso de simuladores torna-se fundamental. Ao se utilizar simuladores, é possível criar e testar vários cenários, variando configurações da NoC, como tamanho de *buffers* ou tamanho da rede. Deste modo, os simuladores possibilitam a verificação do impacto das alterações nas configurações da NoC sem a necessidade de uma implementação em *hardware*, tornando o processo mais rápido e barato. Este trabalho visa o desenvolvimento de um simulador de NoCs. Para a implementação do simulador, foi utilizado um sistema multiagente. A escolha de desenvolver um simulador de NoCs na forma de um sistema multiagente se deu através da análise das características que os roteadores, que compõem as NoCs, têm em comum com os agentes, que compõem os sistemas multiagente. O simulador permite aos projetistas de NoCs avaliar diferentes configurações e estratégias de roteamento. Como resultados de simulação, podem ser observados indicadores como taxa de utilização de roteadores, contenção da rede e atraso latência no envio de mensagens. Para a validação do simulador, são comparados cenários com diferentes configurações e o impacto destes na utilização de recursos é analisado.

Palavras chave — Network-on-Chip; NoC, SMA; Sistemas Multiagente; Simulação.

Abstract

The evolution of integrated circuits manufacturing processes allows large systems, implementing several functionalities, to be built within the same chip. This concept is known as SoC (System-on-a-Chip). However, these complex systems require the development of mechanisms of communication between the parts that compose this system. The communication between components of a SoC, performed by routers, is known as NoC (Network-on-Chip). To explore better design solutions during the creation of a NoC, the use of simulators becomes crucial. When using simulators, it is possible to create and test various scenarios, varying NoC configurations, such as buffers size or network size. Thus, simulators allow verification of the impact caused by NoC configuration changes without the need for a hardware implementation, making the process faster and cheaper. This work aims at the development of a NoC simulator. For the implementation of the simulator, a multi-agent system was used. The choice of developing a NoC simulator in the form of a multi-agent system was based on the analysis of the characteristics that the routers have in common with the agents. The simulator allows NoC designers to evaluate different routing configurations and strategies. As simulation results, indicators such as the rate of utilization of routers, network contention and latency delay in sending messages can be observed. In order to validate the simulator, scenarios with different configurations are compared and their impact on resource utilization are analyzed.

Keywords - Network-on-chip; NoC, MAS; Multi-agent System; Simulation.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

ABMS	*Agent-based Modeling and Simulation*
AS	*Area Consumption*
BS	*Buffer Size*
CSV	*Comma-separated Values*
D	Disponibilidade
DVD	*Digital Versatile Disc*
FIFO	*First-In, First-Out*
GPU	*Graphics Processing Unit*
GUI	*Graphical User Interface*
IA	Inteligência Artificial
IAD	Inteligência Artificial Distribuída
IP	*Intellectual Property*
L	*Latency*
NF	*Negative-First*
NL	*North-Last*
NS	*Network Size*
NoC	*Network-On-Chip*
OSI	*Open Systems Interconnection*
PC	*Power Consumption*
PD	*Packet Distribution*
PE	*Processing Element*
PIR	*Packet Injection Rate*
PPGComp	Programa de Pós-graduação em Computação
QOS	*Quality of Service*
RA	*Routing Algorithm*
RAM	R*andom Access Memory*
S	Síntese de *Hardware*

SAF	*Store-and-Forward*
SMA	Sistema Multiagente
SOC	*System-On-Chip*
SS	*Selection Strategy*
T	*Throughput*
TCP	*Transmission Control Protocol*
TD	*Traffic Distribution*
VCT	*Virtual-Cut-Through*
WF	*West-First*

LISTA DE FIGURAS

LISTA DE TABELAS

1 INTRODUÇÃO

Em 1965, a Lei de Moore (MOORE, 1965) apresentou projeções sobre o avanço no processo de fabricação de circuitos eletrônicos. Esta lei diz que o número de transistores que compõem circuitos iria dobrar a cada 18 meses pelo mesmo custo. Este avanço nas tecnologias de fabricação permite que sejam desenvolvidos circuitos integrados cada vez menores e mais eficientes, abrindo espaço para a criação de sistemas embarcados. Os sistemas embarcados são sistemas eletrônicos que estão acoplados em produtos e funcionam de forma transparente para o usuário (MARDEWEL, 2003). Estes sistemas são capazes de receber, processar e controlar informações de modo a entregar dados relevantes ao usuário. Sistemas embarcados são encontrados em *smartphones*, leitores de DVDs e centrais de injeção eletrônica de automóveis (GRAAF, 2003).

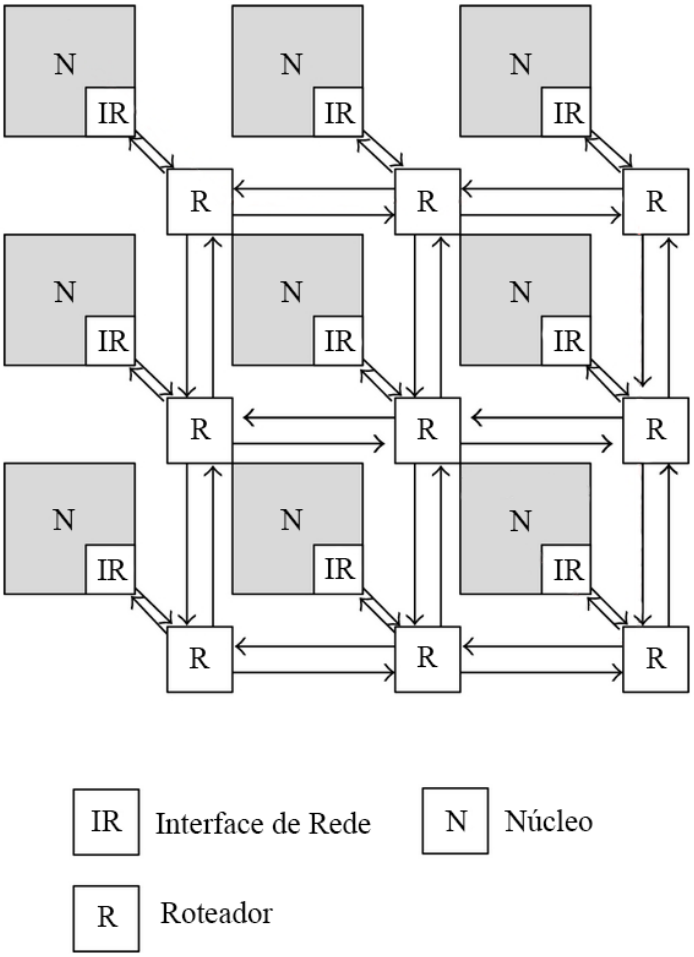
A evolução dos sistemas embarcados fez com que estes se tornassem mais complexos, através da integração de componentes com funcionalidade e propósitos específicos dentro de um único chip. Esta evolução levou à criação de sistemas de grande porte, trazendo funcionalidades tradicionalmente implementadas externamente para dentro do circuito. Estes sistemas são denominados SoC (*System-on-a-chip*) (CHANG, 1999) (BERTOZZI e BENINI, 2004).

Quando comparados com os sistemas compostos de componentes individuais, os SoCs podem apresentar melhorias no desempenho, redução no atraso de comunicação e diminuição dos custos de fabricação, já que módulos que funcionam em conjunto são alocados no mesmo circuito. Devido à complexidade criada nos SoCs, mecanismos para comunicação e interligação entre componentes dentro do circuito foram propostos, dentre elas as chamadas NoCs (*Network-on-Chip*) (HEMANI, 2000).

NoCs implementam uma rede de interconexão capaz de realizar a comunicação entre componentes de um circuito. Uma NoC tradicional é composta por roteadores que têm a função de realizar a troca de mensagens entre componentes da NoC. Um exemplo

de funcionamento de uma NoC é apresentado na Figura 1. Cada núcleo está ligado a um roteador através de uma interface de rede. Desta forma, se dois núcleos desejam se comunicar, os processos de envio e recebimento de mensagens, do nodo origem até o nodo destino, são realizados pelos roteadores.

Figura 1: Estrutura básica de uma NoC



Fonte: Adaptado de (TSAI, 2012).

O desempenho e a confiabilidade das NoCs são diretamente afetados pelo algoritmo de roteamento e topologia utilizados (ZHUANSUN, 2017). Os algoritmos de roteamento têm a função de determinar qual caminho os pacotes das mensagens irão seguir para que ocorra a troca de informações entre o nodo origem e o nodo destino. Para cada par de nodos, podem existir diversos caminhos possíveis. Então, cabe ao algoritmo de roteamento determinar qual caminho seguir. Neste contexto, é necessária a

criação de algoritmos que sejam capazes de determinar o melhor caminho para as trocas de mensagens, levando em conta requisitos de desempenho ou tolerância a falhas, por exemplo. Portanto, ao desenvolver um algoritmo de roteamento, é necessário que ele garanta algumas propriedades de segurança (*safety*) e vivacidade (*liveness*), como a capacidade de evitar impasses (*deadlocks*) e inanição (*starvation*) (COTA, 2012).

Porém, desenvolver estes algoritmos e avaliar as suas propriedades não é uma tarefa fácil. Aspectos como congestionamento e estado dos *buffers* dos roteadores precisam ser analisadas para determinar qual a melhor opção dentre as alternativas de rotas. O congestionamento pode ocasionar atrasos na comunicação, comprometendo o desempenho do sistema (COTA, 2012). A taxa de ocupação dos *buffers* deve ser avaliada de modo a não ocasionar contenção, nem a subutilização de recursos. Dessa forma, para projetar e avaliar algoritmos de roteamento, ferramentas de simulação podem ser utilizadas.

O uso de simuladores permite avaliar diferentes projetos de *hardware*, sem que seja necessária a construção física destes sistemas. Dessa forma, o processo de desenvolvimento torna-se mais barato e permite a exploração de um maior conjunto de parâmetros. Além disso, simuladores permitem testar cenários de *hardware* compostos por topologias grandes e complexas, muitas vezes representando arquiteturas que teriam um custo de implementação elevado, o que pode ser inviável em um ambiente de experimentação em dispositivos físicos. Simuladores para NoCs têm sido explorados academicamente e pela indústria, como é apresentado em (ACHBALLAH e SAOUD, 2013) (ALALAKI e AGYEMAN, 2017) (CATANIA, 2015) (LIS, 2010) (NS-2, 2018) (AL-BADI, 2009) (CHAWADE, 2012) (KAMALI, 2018) (KAMALI e HESSABI, 2016) (OXMAN e WEISS, 2016) (GHOSH, 2014) (CHIOU, 2011) (ZOLGHADR, 2011) (SEICULESCU, 2009) (ATLAS, 2018) (PALERMO e SILVANO, 2004) (MURALI, 2004) (EVAIN, 2004) (CHAN e PARAMESWARAN, 2004) (FLEXNOC, 2018) (INOC, 2018) (CHAIN, 2018) (JIANG, 2013) (NIKOUNIA e MOHAMMADI, 2015) (BEN, 2013) (CARLSON, 2015) (HOSSANI, 2008) (POWER, 2014) (KAMALI e HESSABI, 2016) (VIGLUCCI e CARPENTER, 2016) (WEHNER, 2015) e (WANG, 2014).

Os simuladores podem utilizar diferentes graus de abstração. Por um lado, simuladores com baixo grau de abstração se assemelham mais ao sistema real,

utilizando um maior número de detalhes no funcionamento interno do sistema. Este tipo de simulador normalmente apresenta maior custo computacional, impactando no tempo de execução das simulações e maior complexidade para descrever os cenários de teste, dado o nível de detalhes que precisa ser modelado. Por outro lado, os simuladores que utilizam um alto grau de abstração buscam apresentar ao usuário apenas os comportamentos principais do sistema, de forma a realizar uma simulação que se aproxima ao comportamento real, omitindo detalhes internos específicos. Estas ferramentas podem realizar simulações que representam dispositivos de *hardware* de tamanho grande, sem o custo de implementação direta no *hardware*.

1.1 Objetivo Geral

O principal objetivo deste trabalho é a criação de uma ferramenta de simulação para NoCs que permite a descrição de cenários de simulação em alto nível. Este tipo de abstração provê independência do *hardware*, o que possibilita que a grande maioria das NoCs sejam implementadas e testadas. A possibilidade de simular NoCs é necessária para acompanhar o avanço com que as NoCs estão evoluindo. Para se ter uma ideia do aumento no tamanho das NoCs, em 2007 já existiam NoCs simuladas pela Intel utilizando malhas 2-d com 80 nodos (VANGAL, 2007), e em 2014 foram feitos testes com 256 nodos, mostrando que o tamanho das NoCs está aumentando significativamente (CHEN, 2014).

1.2 Objetivos Específicos

Como elementos para análise, o simulador deve apresentar indicadores de uso dos recursos, os quais possibilitam que sejam observados o uso dos roteadores da NoC e latência na comunicação. Um exemplo de indicador é a taxa de utilização dos roteadores da NoC. Dados coletados devem permitir a avaliação dos recursos tanto durante, quanto ao final da simulação, auxiliando projetistas na escolha do algoritmo de roteamento que melhor se adapte à determinados requisitos.

Parâmetros tais como o tamanho de *buffer*, tipos de carga ou tamanho da rede de uma NoC podem afetar em suas métricas de desempenho, como a latência. Assim, como objetivos específicos, pode-se listar a exploração de diferentes configurações de NoCs de forma a verificar comportamentos, principalmente observando o impacto de determinados atributos, como:

- na utilização dos *buffers* de cada roteador;
- no desempenho de uma NoC;
- no desempenho do algoritmo de roteamento.

Estes objetivos parciais e os resultados levantados através da exploração de diferentes parâmetros demonstram a capacidade do simulador em avaliar diferenças, de forma configurável e com baixo tempo de simulação.

1.3 Motivação

A motivação para a adoção de um ambiente multiagente vem das semelhanças entre os roteadores, entidades que compõem uma NoC, e os agentes, entidades que compõem um sistema multiagente. Uma das vantagens de adotar uma modelagem de Sistema Multiagente (SMA) é a facilidade em realizar análises da NoC tanto como um todo quanto observar as características de um único roteador (representado por um agente) para detectar congestionamentos.

Além disso, busca-se aprimorar algumas limitações das ferramentas já existentes, como a falta de interface gráfica. O uso de uma interface gráfica auxilia não só na usabilidade do sistema, mas também permite que os usuários possam observar o comportamento da NoC em tempo de execução. Utilizando uma interface gráfica, também é possível observar a simulação passo a passo, permitindo analisar comportamentos com a possibilidade de interromper e retomar a simulação conforme seja necessário. Além da interface gráfica, também se busca aprimorar outras características, como a utilização de código livre, permitindo que o simulador seja usado tanto na indústria quanto no meio acadêmico.

1.4 Organização do Trabalho

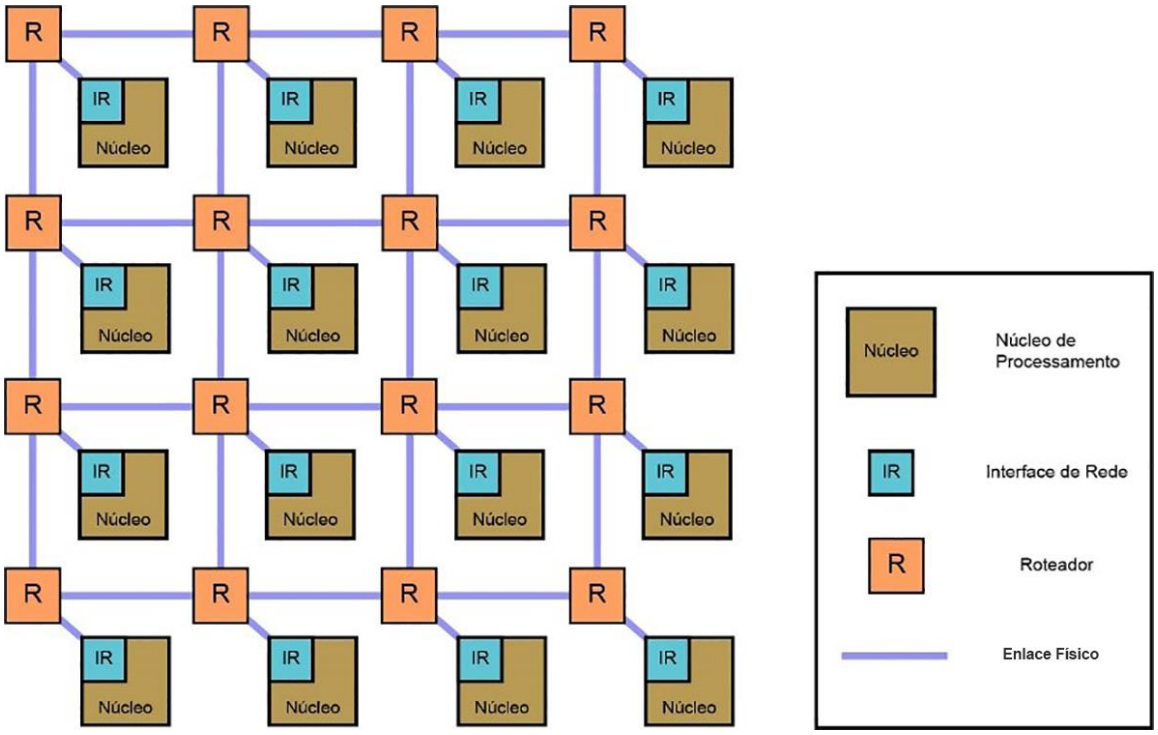
Esta dissertação está estruturada da seguinte maneira: o Capítulo 2 conceitua NoCs. O capítulo 3 apresenta fundamentos sobre Sistemas Multiagente. O Capítulo 4 apresenta os simuladores de NoC, comparando as principais características dos simuladores encontrados na literatura com o simulador proposto. O Capítulo 5 descreve detalhes sobre o funcionamento e a implementação do simulador. O Capítulo 6 apresenta a avaliação da ferramenta, além dos os resultados obtidos através dos cenários de teste. Por fim, o Capítulo 7 relata as conclusões, além dos trabalhos futuros.

2 NETWORK-ON-CHIP

A complexidade criada pela integração de vários sistemas em um circuito faz com que seja repensada a forma com que estas partes se comunicam. As antigas arquiteturas de comunicação (como barramentos ou fios dedicados), se utilizadas nos SoCs, oferecem baixa escalabilidade. Além disso, os barramentos apresentam limitações de transmissão, pois apenas uma palavra pode ser transmitida por ciclo de *clock*. Da mesma forma, conexões com fios dedicados também não são uma boa opção, por possuírem baixo reuso e flexibilidade. Como alternativa para estas limitações, surgem outras propostas de como realizar esta comunicação, as NoCs (*Network-On-Chip*) (HEMANI, 2000) (BRIAO, 2008).

NoC é um paradigma/arquitetura de comunicação entre diferentes partes que compõem o circuito. De uma maneira geral, a comunicação é realizada através da criação de uma rede, composta por nodos de processamento conectados a nodos de chaveamento através de uma interface de rede, e os nodos de chaveamento interligados por enlaces (*links)* físicos. Um exemplo de organização de uma NoC é apresentado na Figura 2 (TOPÎRCEANU, 2013). Os nodos de processamento, também conhecidos como núcleos, PE (*Processing Element*) ou IP *cores*, são acoplados às NoCs e têm a função de executar as ações do sistema. Já os nodos de chaveamento, ou roteadores, (*routers*) são responsáveis por realizar a transferência de mensagens entre dois núcleos que desejam se comunicar, através de enlaces físicos (*links*). A forma com que os nodos de processamento e roteadores são dispostos depende da topologia adotada.

Figura 2: Organização de uma NoC



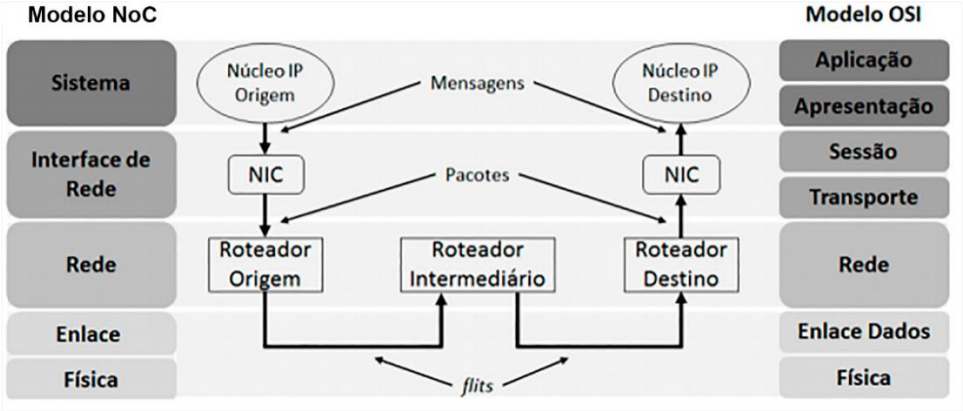
Fonte: Adaptado de (TOPÎRCEANU, 2013)

A possibilidade de trocas de mensagens entre dois pontos da NoC pode ocorrer por diversos caminhos. Este fato faz com que seja necessário adotar estratégias de roteamento de maneira similar ao que acontece na Internet para determinar o caminho por onde a troca de mensagens ocorrerá. A possibilidade de diversas comunicações acontecerem paralelamente, em caminhos distintos, faz com que a comunicação nas NoCs seja mais eficiente quando comparado com os barramentos convencionais, podendo ser observado qual o menor caminho entre os pontos, ou qual caminho está menos congestionado. Além disso, as decisões de roteamento dos pacotes que trafegam pela NoC podem ser distribuídas, auxiliando assim na escalabilidade do sistema.

A arquitetura de NoCs pode ser representada de maneira similar à estrutura do modelo OSI (*Open Systems Interconnection*) (OSI, 1977), que é utilizado em redes de computadores. Entretanto, o modelo OSI não se caracteriza como uma arquitetura, pois não especifica a maneira exata com que os serviços e protocolos devem ser utilizados pelas aplicações (CARDOZO, 2012). Este modelo de referência é dividido em 7 camadas, de forma a criar uma abstração entre cada camada. Cada uma das camadas é responsável por cuidar de uma parte do sistema de comunicação que irá transmitir a mensagem entre dois sistemas computacionais.

A Figura 3 ilustra a equivalência das camadas nos dois modelos. No modelo das NoCs, a camada Sistema desempenha as funções das camadas de Aplicação e Apresentação do modelo OSI. É nesta camada que são implementadas as funcionalidades de cada núcleo de processamento. A camada de Interface de Rede é responsável por integrar a camada Sistema com a Rede, provendo uma interface de comunicação. A camada de Rede é formada pelos roteadores que implementam a comunicação das diferentes partes que compõem a NoC. É nesta camada que são implementados os algoritmos de roteamento. A camada de Enlace tem por objetivo manipular a sincronização, codificação e manter a confiabilidade na transmissão de dados. Por fim, a camada Física trata das ligações elétricas pelas quais os dados irão trafegar.

Figura 3: Comparação entre o modelo OSI e a estrutura de uma NoC



Fonte: adaptado de (NUNES, 2015)

A comunicação entre componentes da NoC ocorre através do envio e recebimento de mensagens. Porém, como as mensagens que trafegam são grandes e de tamanhos variados, se faz necessário dividir estas mensagens em pacotes menores, que tenham um tamanho menor ou igual à largura de banda do Enlace (NUNES, 2015). Cada pacote é dividido em *flits*, que são a menor unidade de dados sobre a qual é realizado o controle de fluxo. Os *flits* são classificados em:

1. *Header* (cabeçalho) – contém as informações necessárias para rotear a mensagem pela rede;
2. *Payload* (carga útil) – o corpo da mensagem, ou seja, a informação relevante para a aplicação;
3. *Trailer* (terminador) – marcador que indica o término da mensagem.

Os roteadores são componentes cruciais dentro das NoCs, visto que são responsáveis por transmitir os dados que devem trafegar pela rede. A maioria dos roteadores implementam uma rede de interconexão (*crossbar*) para possibilitar que múltiplas conexões entre portas de entrada e saída aconteçam simultaneamente. Como os roteadores recebem uma grande quantidade de pacotes, eles devem prover filas eficientes onde estes pacotes ficam armazenados até serem processados e enviados aos seus destinatários. Os pacotes podem ser guardados em *buffers* de entrada, *buffers* de saída, em *buffers* de entrada-saída ou roteador com *buffer* central (HENNESSY e PATTERSON, 2008).

Uma vez armazenados os pacotes, estes precisam ser enviados segundo algum protocolo de roteamento. De forma geral, o roteamento é implementado utilizando uma máquina de estados finita ou uma tabela de encaminhamento (armazenada dentro da unidade de controle de roteamento no roteador). Na primeira opção, os dados de roteamento presentes no cabeçalho do pacote são processados pela máquina de estados, e esta determina para qual saída a mensagem deve ser encaminhada. Já na segunda opção, as informações de roteamento presentes no cabeçalho do pacote são utilizadas como um endereço de acesso à tabela de roteamento, carregada na inicialização do roteador, que contém as saídas a serem utilizadas.

A unidade de controle de roteamento ainda precisa desempenhar outra função, a arbitragem. Essa função se faz necessária pois como os pacotes podem chegar de maneira simultânea às portas de entrada, dois pacotes podem requerer uma mesma saída. Desta forma, cabe ao roteador determinar qual pacote terá prioridade ao acesso na porta requerida. Para desempenhar tal papel, a unidade pode ser centralizada ou distribuída. Na abordagem centralizada, todo o processo de distribuição dos pacotes às portas de saída é feito por um agente centralizador. Já na abordagem descentralizada, cada porta de entrada possui um módulo de roteamento, juntamente com um módulo de arbitragem ligado à sua respectiva porta de saída (NUNES, 2015).

A camada de enlace é responsável por realizar a interconexão entre os roteadores que compõem a NoC. A via de comunicação entre os roteadores pode ser *half* ou *full duplex*. Na comunicação *half duplex*, os dados podem trafegar apenas em um sentido (comunicação unidirecional). Já na comunicação *full duplex* os dados trafegam em

ambos os sentidos (comunicação bidirecional). Como as comunicações podem ocorrer por vários caminhos, geralmente a opção mais escolhida é *full duplex* (NUNES, 2015).

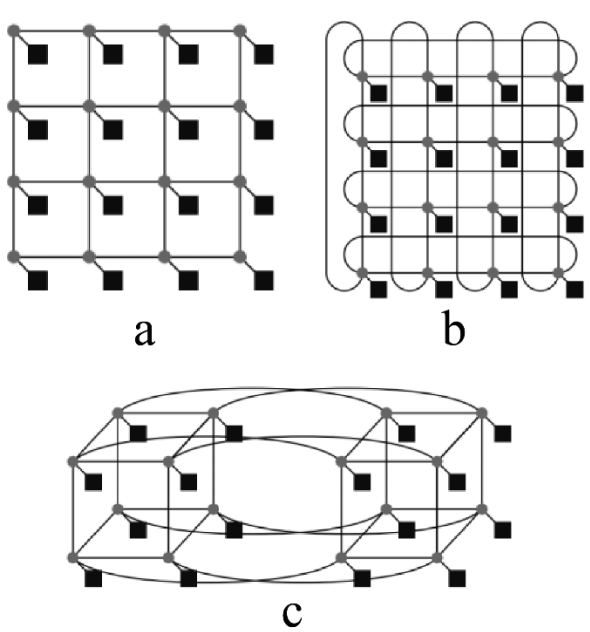
Dentre as principais características que definem uma NoC, as que mais se destacam são: topologia, algoritmo de roteamento, chaveamento, controle de fluxo, arbitragem e memorização. Devido à sua relevância, estas características serão explicadas a seguir.

2.1 Topologia

A topologia de rede trata da descrição de como os roteadores estão dispostos e conectados entre si. Estas redes de roteadores são classificadas em dois tipos: redes diretas ou indiretas (HENNESSY e PATTERSON, 2008).

Nas redes diretas, cada nodo de chaveamento está diretamente ligado a um nodo de processamento, e esta união fica conhecida como nodo. Cada nodo está diretamente ligado a nodos vizinhos. Dessa forma, a comunicação entre vizinhos acontece diretamente. Entretanto, a comunicação entre nodos não vizinhos precisa passar por um ou mais roteadores intermediários. Exemplos de topologias de redes diretas são malha 2D, toróide 2D e hipercubo 3D, conforme exemplificado na Figura 4 (a), (b) e (c), respectivamente.

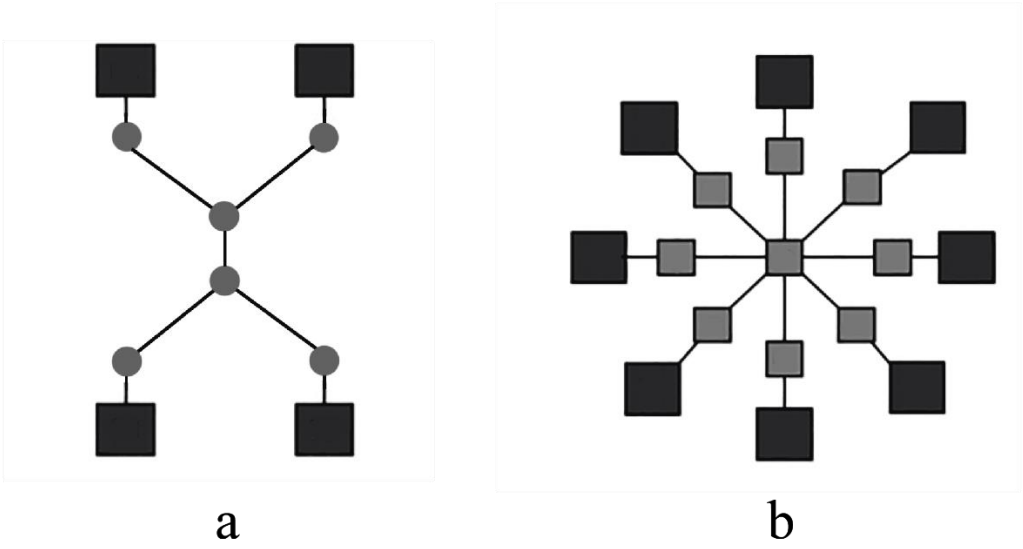
Figura 4: Redes nos formatos: (a) Malha 2D, (b) Toróide 2D e (c) Hipercubo 3D



Fonte: (HENNESSY e PATTERSON, 2008)

Já nas redes indiretas, os nodos de chaveamento possuem portas de comunicação bidimensionais para comunicação com outros nodos, e somente alguns destes nodos possuem conexão com os nodos de processamento, ou seja, apenas estes podem ser emissores ou destinatários de mensagens. Exemplos desta topologia são estrela e árvore (SANTANA 2014), como mostradas na Figura 5 (a) e (b).

Figura 5: Redes nos formatos: (a) Árvore e (b) Estrela



Fonte: (HENNESSY e PATTERSON, 2008)

2.2 Chaveamento

Chaveamento é a funcionalidade presente dentro dos roteadores que determina como um pacote recebido em uma de suas portas de entrada é encaminhado para suas portas de saída. Existem duas classificações para esse mecanismo: o chaveamento por circuito e o chaveamento por pacote.

O chaveamento por circuito acontece quando se estabelece todo o caminho entre a origem e o destino que a mensagem deve atingir. A liberação do caminho acontece quando a última parte da mensagem, o terminador, passa pelo caminho. Como esta modalidade de chaveamento possui um caminho dedicado, o tempo necessário para realizar a transmissão da mensagem é reduzido. Por outro lado, como se trata de um caminho único, caso este já esteja ocupado, não é possível transmitir pacotes de outra

mensagem por este caminho, fazendo com que novas mensagens precisem aguardar, ocasionando congestionamento. Outro ponto negativo do chaveamento por circuito é que, caso aconteça algum problema em uma das partes que estão envolvidas na comunicação, a conexão é interrompida (NUNES, 2015).

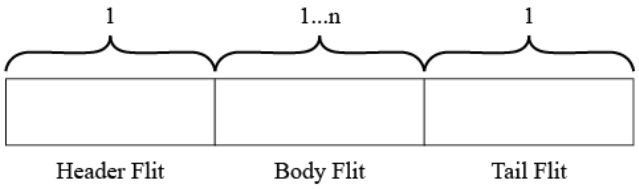
No chaveamento por pacote não existe a reserva de caminhos. A transmissão acontece encaminhando os pacotes para os roteadores, e através das informações já contidas no cabeçalho do pacote (bits de integridade, *payload* e terminador) é realizada a verificação de redundância. Neste tipo de chaveamento, os pacotes são armazenados no roteador antes de serem enviados para o próximo roteador. Como este chaveamento não exige um caminho dedicado, há uma diminuição nos custos de implementação e não acontece o problema de quebra de conexão presente no chaveamento por circuito. Porém, como as diversas partes em que o pacote é dividido podem percorrer diferentes caminhos, estas podem não chegar na mesma ordem. Existem vários métodos deste tipo de chaveamento, tais como SAF (*Store-and-Forward*), VCT (*Virtual-Cut-Through*) e *Wormhole* (NUNES, 2015).

O método SAF trata do processo onde o roteador recebe o pacote, armazena-o em um de seus *buffers*, lê o cabeçalho para identificar o destinatário, consulta o algoritmo de roteamento para escolher a porta de saída adequada e, por fim, envia o pacote para o roteador destino. Como todos os pacotes precisam ser armazenados nos *buffers*, este método aumenta o custo da rede, pois são necessários vários *buffers* para guardar as inúmeras partes do pacote, até que todas cheguem e o pacote seja montado. Além disso, como o roteador precisa realizar o processo de leitura dos cabeçalhos dos pacotes, este método coloca um tempo adicional no processo de comunicação.

O método VCT surgiu como uma melhoria do método SAF. Este apresenta como vantagem o fato de só armazenar os pacotes inteiros no roteador no caso da porta destino estar ocupada. Esta melhoria diminui a latência ocasionada no SAF, pois no caso da porta não estar ocupada, a mensagem é diretamente enviada sem a necessidade de realizar o armazenamento local. Ainda assim, o método VCT necessita ter *buffers* suficientes para armazenar todas as mensagens cujos canais estejam ocupados. Note, entretanto, que mesmo na pior das hipóteses, este método terá o comportamento igual ao SAF.

O chaveamento *Wormhole* é uma melhoria do método VCT, que visa diminuir a quantidade de *buffers* que mantêm os pacotes bloqueados na rede. Para isto, os pacotes são divididos em *flits* (partes da mensagem) e transmitidos para os roteadores intermediários. Este mecanismo cria uma espécie de *pipeline* na rede, pois os *flits* do cabeçalho (que contém as informações sobre o destino) trafegam pela rede, seguidos dos *flits* de dados e de fim de mensagem (*tail*). Caso os *flits* de cabeçalho sejam bloqueados, os de dados ficam armazenados nos roteadores intermediários. A liberação do canal de comunicação só acontece quando todos os *flits* da mensagem chegarem ao destino, e isto aumenta o risco de *deadlocks* acontecerem. Embora esta seja uma das principais desvantagens, ela pode ser contornada através de canais virtuais, que são vários canais lógicos que compartilham um mesmo canal físico. Como os roteadores não precisam armazenar pacotes inteiros, as filas dos roteadores intermediários são diminuídas, o que possibilita que os roteadores sejam menores, mais rápidos e, por consequência, mais baratos. A estrutura da divisão de um pacote em *flits* é ilustrada na Figura 6.

Figura 6: Principais *flits* de um pacote



Fonte: do autor

2.3 Controle de Fluxo

Este componente tem por objetivo controlar o que acontece no caso de um pacote requisitar um recurso que já está sendo utilizado. O controle acontece na camada de Enlace, e cada uma das portas dos roteadores deve conter um *buffer* que guarda as informações que serão enviadas. As três estratégias de controle de fluxo mais frequentemente utilizadas são: *handshake, slack buffer* e baseado em créditos.

No protocolo *handshake* são utilizados sinais que indicam que o emissor pretende enviar dados pela rede, e sinalizam que os recursos estão disponíveis para receber e transmitir os dados. Enquanto isso, no controle por *slack buffer* são os próprios *buffers* que sinalizam se estão disponíveis para utilização ou não. No controle de fluxo baseado em créditos quando um transmissor deseja enviar uma informação, ele a disponibiliza no barramento de comunicação e indica a existência do dado para o destino através de uma linha direta, além de enviar um sinal de *clock* utilizado na sincronização da transferência. Quando o receptor percebe o dado disponível, verifica se existe o espaço necessário para o armazenamento; caso exista, devolve um sinal de crédito ao emissor. Ao perceber que não existe espaço suficiente, o destinatário sinaliza ao emissor através do sinal de crédito. Ao receber o sinal de crédito, o transmissor se compromete de manter o dado no barramento até que o receptor indique novamente que pode receber os dados (através do sinal de crédito) (NUNES, 2015).

2.4 Arbitragem

No funcionamento de uma NoC, as mensagens podem transitar por diversos roteadores até chegar no destino. Portanto, os roteadores recebem diversas mensagens em suas entradas e devem encaminhá-las para uma saída, de forma com que avancem em direção ao destino. Existe a possibilidade de que mais de um pacote requisite uma mesma saída em um roteador. Nesse caso, é necessária a utilização de um árbitro. A função do árbitro é decidir qual porta de entrada tem prioridade de acesso a uma porta de saída em caso de múltiplas requisições simultâneas.

Além de solucionar concorrências, o árbitro pode direcionar os pacotes para portas não utilizadas ou roteadores com menos tráfego. Dessa forma, o uso de um árbitro eficiente permite que seja possível obter uma melhor utilização da rede, diminuindo possíveis sobrecargas.

Existem algumas características que definem um árbitro. A primeira é relacionada à sua implementação, ou seja, qual algoritmo o árbitro executa para escolher a porta de maior prioridade. Existem diversas implementações de árbitros. Um exemplo famoso é o algoritmo Round-Robin (HAHNE e GALLANGER, 1986). Este algoritmo faz com que a prioridade das portas mude a cada ciclo, de forma circular. Outros

exemplos de implementações de árbitros são encontrados em (GUDERIAN, 2011) e em (KUMAR e KUMAR, 2014).

Além do algoritmo, os árbitros também podem ser classificados segundo a sua abordagem, podendo ser centralizada ou descentralizada. Na abordagem centralizada, o algoritmo está implementado em um módulo central, que realiza a arbitragem de todos os roteadores. Já na abordagem descentralizada, a implementação e execução do árbitro é feita localmente, dentro de cada roteador. Desta forma, na abordagem descentralizada, a definição das prioridades de utilização das portas é feita por cada roteador, individualmente (NUNES, 2015).

2.5 Memorização

Nas implementações de NoCs que fazem uso do chaveamento por pacotes, cada roteador deve ser capaz de armazenar todos os *flits* de uma mensagem até que ela seja completamente transmitida. Por esse motivo, os roteadores devem contar com um sistema de memorização de mensagens. O tipo de armazenamento depende do tipo de *buffers* que o roteador utiliza. Existem duas abordagens, sendo uma centralizada, e uma distribuída.

A abordagem centralizada é utilizada por roteadores que usam um *buffer* central. Todas as partes das mensagens são armazenadas neste *buffer* único. Além disso, esse *buffer* é compartilhado por todas as portas de entrada e também todas de saída (SANTANA, 2014).

Já a abordagem distribuída é utilizada por roteadores que usam *buffers* separados, seja apenas nas entradas, nas saídas, ou em ambos (entrada/saída). Nessa abordagem, cada porta armazena os pacotes que está transmitindo em um *buffer* local, dedicado apenas para o armazenamento dos pacotes destinados àquela porta (SANTANA, 2014).

2.6 Roteamento

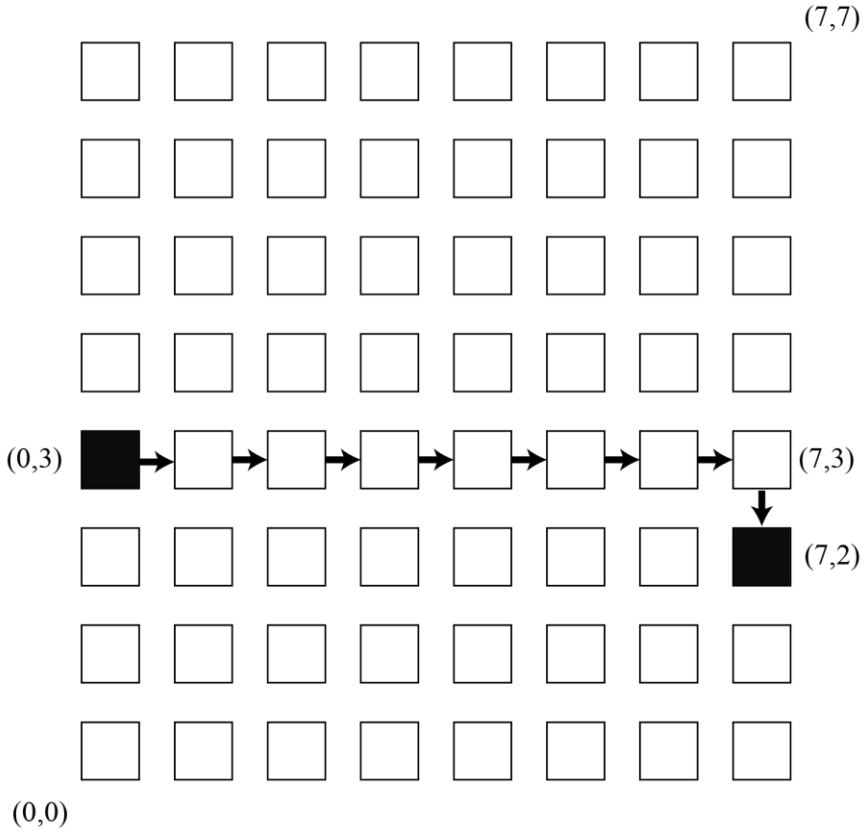
O algoritmo de roteamento de uma NoC determina qual caminho um pacote partindo de um nodo emissor deve tomar para chegar a um nodo destino. Estes algoritmos, além de determinarem o caminho, devem prover mecanismos para evitar

problemas de comunicação. Existem dois principais problemas de comunicação em NoCs: *deadlocks* e *livelocks*. *Deadlocks* são problemas onde os roteadores estão com os *buffers* ocupados e esperam por outros roteadores que também estão na mesma situação serem liberados, fazendo com que nenhuma mensagem chegue ao destino (HENNESSY e PATTERSON, 2005). Diferentemente, *Livelock* ocorre quando um pacote circula continuamente pela rede por um grande intervalo de tempo, possivelmente infinito, sem chegar ao destinatário (HENNESSY e PATTERSON, 2005).

Existe uma série de fatores que diferenciam algoritmos de roteamento. Os principais algoritmos citados na literatura são: XY, NL (*North-Last*), NF (*Negative-First*) e WF (*West-First*) (CARARA, 2004). Como todo pacote parte de um ponto de origem e deve chegar a um ponto de destino, cria-se uma convenção, onde a posição do nó fonte (*source*) é dada por coordenadas (Xs, Ys) e do nó destino (*target*) por coordenadas (Xt, Yt). Essa convenção é utilizada na descrição dos algoritmos a seguir.

O Algoritmo XY é um algoritmo determinístico. Isto significa que para uma dada entrada, o algoritmo sempre irá produzir a mesma saída. Neste algoritmo, os *flits* de um pacote são roteados na direção X até atingir a coordenada Xt. Em seguida, são roteados na direção Y até atingir Yt. Um exemplo de funcionamento do algoritmo XY é mostrado na Figura 7 (CARARA, 2004). Nesta Figura, o roteador à esquerda (0,3) está enviando pacotes para o roteador à direita (7,2). De acordo com o algoritmo, o roteamento é feito primeiramente no eixo X, até chegar no roteador (7,3). Chegando nele, o pacote atingiu o mesmo valor do eixo X do destino. A partir desse momento, o pacote é roteado no eixo Y, chegando ao destino ao se deslocar uma posição para baixo.

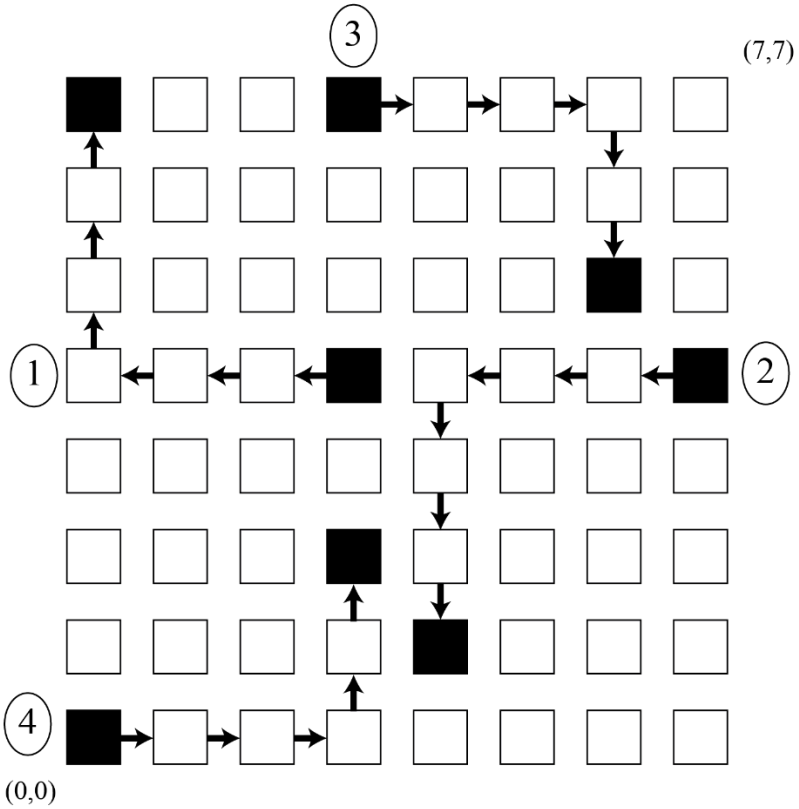
Figura 7: Estrutura do algoritmo XY



Fonte: adaptado de (CARARA, 2004).

O algoritmo NL (*North-Last*) é um algoritmo parcialmente adaptativo, pois é capaz de evitar congestionamento para algumas direções de roteamento. No NL se Yt ≥ Ys, o roteamento dos pacotes acontece da mesma forma que no algoritmo XY. Exemplos desse caso são as representações 1 e 4 da Figura 8. Se Yt < Ys, então o roteamento dos pacotes acontece de maneira adaptativa nas direções sul, leste ou oeste. As representações 2 e 3 da Figura 8 mostram exemplos desses casos (CARARA, 2004).

Figura 8: Estrutura do algoritmo NL

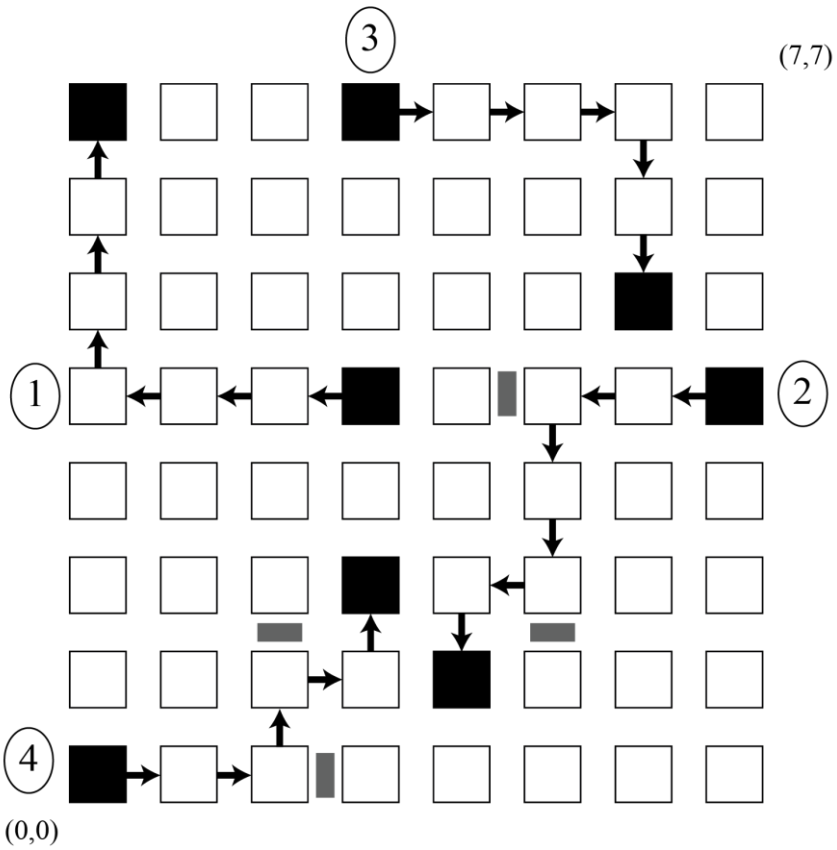


Fonte: (CARARA, 2004)

O algoritmo NF (*Negative First*), da mesma forma que o algoritmo NL, é um algoritmo parcialmente adaptativo. Para determinar o caminho de roteamento, este algoritmo analisa e roteia, primeiramente, as direções negativas, ou seja, sul e oeste. Se ($Xt \leq Xs$ e $Yt \geq Ys$) ou ($Xt \geq Xs$ e $Yt \leq Ys$), o roteamento dos pacotes acontece de forma determinística. Para as demais possibilidades, o roteamento é adaptativo.

Na Figura 9 são apresentados exemplos de funcionamento desse algoritmo. Os casos 1 (fonte (3,4) e destino (0,7)) e 3 (fonte (3,7) e destino (6,5)) são exemplos do roteamento determinístico. Os demais exemplos (casos 2 e 4) são casos onde o roteamento é adaptativo. Ainda sobre a figura, as barras cinzas representam possíveis congestionamentos na rede. Neste caso, o algoritmo pode direcionar os pacotes por outro caminho para evitar atrasos (CARARA, 2004).

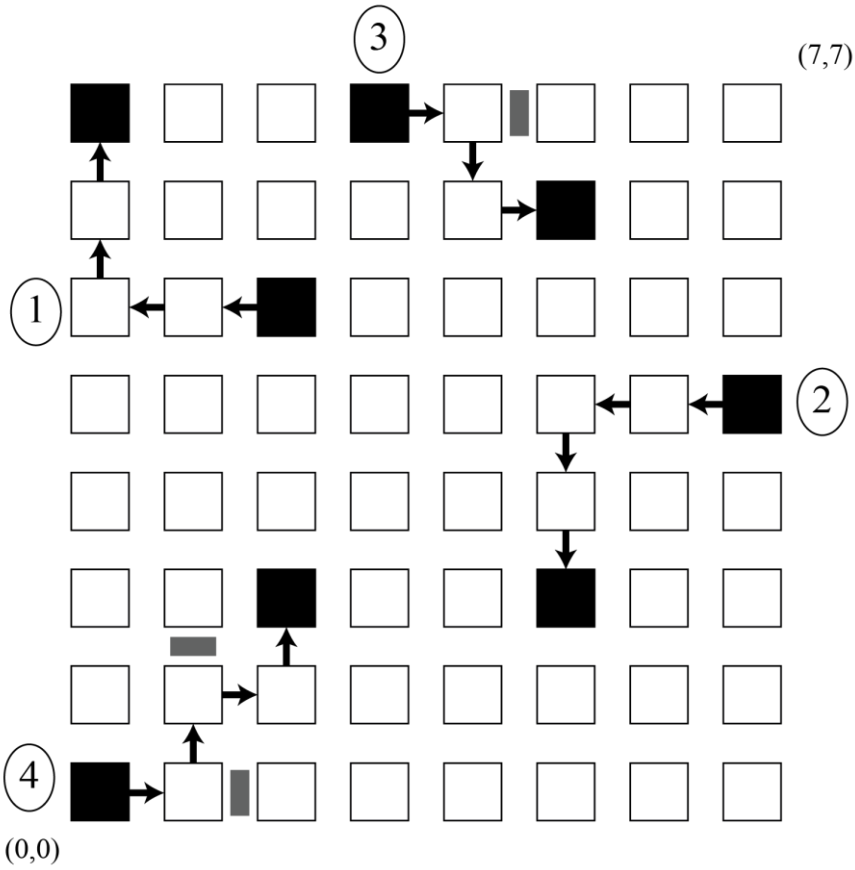
Figura 9: Estrutura do algoritmo NF



Fonte: (CARARA, 2004)

Outro exemplo de algoritmo parcialmente adaptativo é o WF (*West-First*). O funcionamento do algoritmo se dá analisando as posições Xt e Xs. Se Xt ≤ Xs, o roteamento acontece de forma determinística, da mesma forma que o algoritmo XY. Caso contrário, os pacotes são roteados de maneira adaptativa nas direções leste, norte e sul. A Figura 10 apresenta dois exemplos do primeiro caso (1 e 2) e dois do segundo caso (3 e 4). Nos casos onde existem congestionamentos, da mesma forma que o algoritmo anterior, o algoritmo pode direcionar os pacotes por rotas menos congestionadas (CARARA, 2004).

Figura 10: Estrutura do algoritmo WF



Fonte: adaptado de (CARARA, 2004).

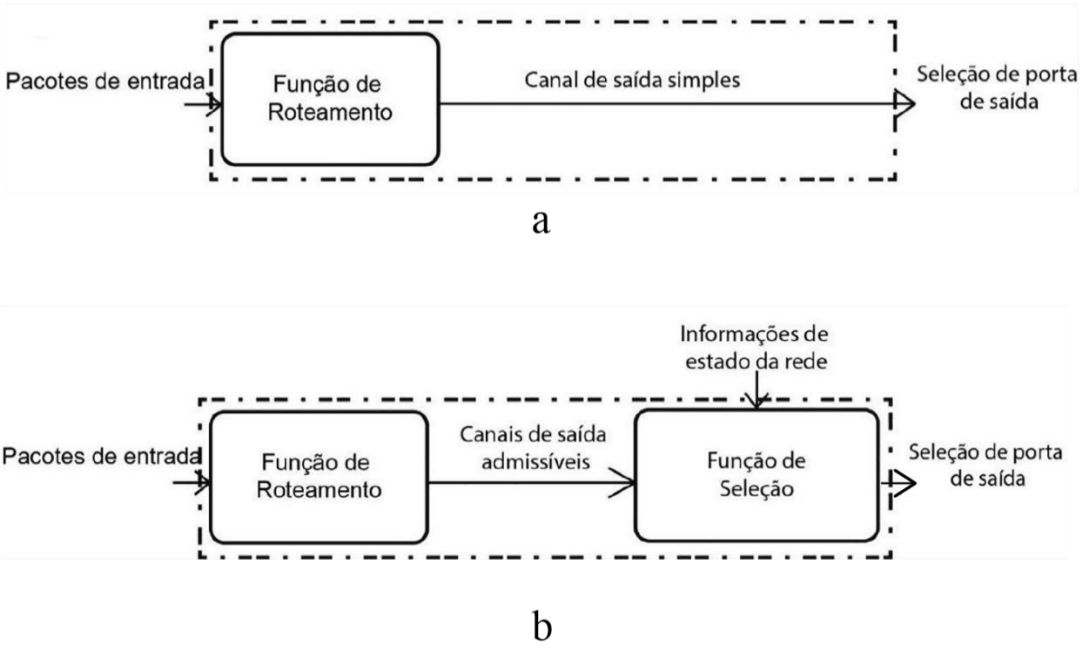
Em algoritmos de roteamento de origem, a sequência de saltos para todo o caminho entre a origem e o destino são codificados no cabeçalho do pacote no nó de origem. A cada roteador durante a transmissão entre os nós de origem e destino, o cabeçalho do pacote é decodificado para que o pacote vá em direção ao destino. As principais desvantagens do roteamento de origem em NoCs são a adaptabilidade limitada de caminhos e a grande sobrecarga para guardar a informação do caminho no cabeçalho do pacote. Por outro lado, nos algoritmos de roteamento distribuídos, o endereço destino só é inserido no cabeçalho do pacote. Cada roteador obtém a decisão de roteamento independentemente com base no conhecimento do endereço de destino. A desvantagem dos algoritmos de roteamento distribuído está na complexidade em construir um roteador que consiga lidar com o esquema distribuído (EZZ-ELDIN e EL-MOURSY e HAMED, 2015).

Algoritmos de roteamento determinísticos direcionam os pacotes em um determinado caminho entre a origem e o destino. Um exemplo típico deste tipo de algoritmo é o XY. A desvantagem do uso de algoritmos desta natureza é que o caminho é determinado sem levar em conta características dinâmicas da rede, como o congestionamento.

Algoritmos de roteamento adaptativo são classificados em parcialmente ou completamente adaptativos. Os algoritmos de roteamento parcialmente adaptativos permitem o roteamento de pacotes apenas pela rota com o menor caminho. Ao contrário destes, nos algoritmos de roteamento completamente adaptativos, cada pacote pode ser encaminhado ao longo de qualquer caminho mínimo disponível entre a origem e o destino. Geralmente, os algoritmos de roteamento adaptativo podem diminuir a latência e aumentar a taxa de transferência.

Algoritmos de roteamento são divididos em dois blocos principais, a função de roteamento e a função de seleção. Nos algoritmos de roteamento determinísticos, a função de roteamento retorna apenas uma porta de saída única. Consequentemente, o bloco da função de seleção não existe, como apresentado na Figura 11 (a). Nos algoritmos de roteamento adaptativos, a função de roteamento gera um grupo de canais de saída admissíveis por onde o pacote pode trafegar até o nó destino. O bloco da função de seleção é empregado para escolher um dentre os canais de saída admissíveis retornados pela função de roteamento baseado nas informações de estado atuais da rede, como mostrado na Figura 11 (b) (EZZ-ELDIN e EL-MOURSY e HAMED, 2015).

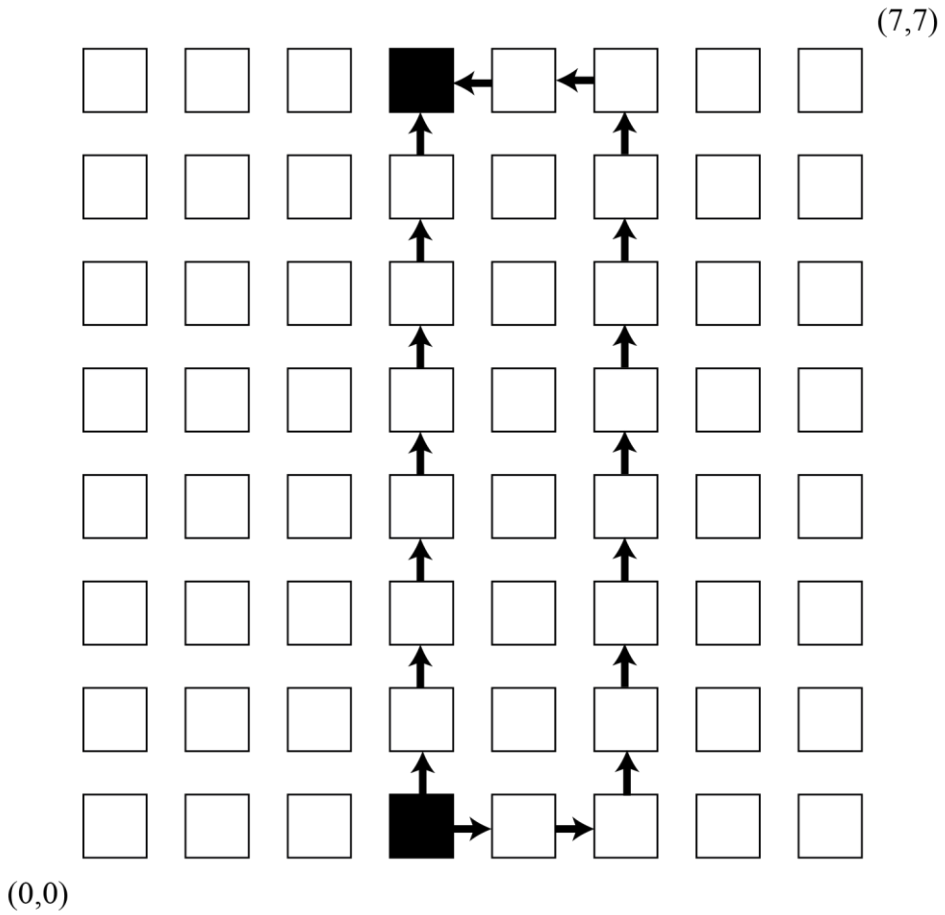
Figura 11: Funções de roteamento e seleção para (a) algoritmos de roteamento determinístico e (b) algoritmos de roteamento adaptativos.



Fonte: Adaptado de (EZZ-ELDIN e EL-MOURSY e HAMED, 2015).

Os algoritmos de roteamento também podem ser classificados como mínimo e não-mínimo. Algoritmos de roteamento mínimo garantem encontrar o menor caminho entre o emissor e o receptor. Os *deadlocks* podem ser evitados usando caminhos mínimos. Algoritmos de roteamento mínimo podem causar um atraso médio excessivo, pois eles não são sempre capazes de evitar congestionamento de tráfego. Algoritmos de roteamento não-mínimo permitem que as mensagens sejam roteadas por caminhos maiores para evitar congestionamento ou prover tolerância a falhas, como é ilustrado na Figura 12 (EZZ-ELDIN e EL-MOURSY e HAMED, 2015). Na Figura, é possível observar que existem diversos caminhos pelos quais uma mensagem que deve ser enviada do roteador origem (0,3) para o roteador destino (7,3) pode seguir.

Figura 12: Caminhos de roteamento mínimos e não-mínimos



Fonte: (EZZ-ELDIN e EL-MOURSY e HAMED, 2015).

Outra classificação dos algoritmos considera se a solução produzida é de um roteamento consciente ou inconsciente de congestionamento. Em algoritmos de roteamento inconsciente, múltiplos caminhos da origem ao destino estão disponíveis e a seleção do caminho de roteamento não depende do estado atual da rede. Consequentemente, os algoritmos de roteamento inconsciente, como o algoritmo XY, negligenciam as condições de congestionamento em suas decisões de roteamento. Isto pode levar a interrupções no balanceamento de carga, fazendo com que certos pontos da rede fiquem sobrecarregados enquanto outros são pouco utilizados. Por outro lado, os esquemas de seleção nos algoritmos de roteamento conscientes de congestionamento são baseados nas estimativas de congestionamento local. O número de canais virtuais disponíveis em uma porta de saída é proposto como uma métrica de contenção para algoritmos de roteamento consciente de congestionamento. Algoritmos de roteamento

conscientes de congestionamento têm melhor desempenho do que os algoritmos inconscientes pois focam em escolher os caminhos menos congestionados para produzir o balanço de carga na rede, especialmente em cargas de tráfego realistas. Vale ressaltar que os algoritmos determinísticos são inconscientes de congestionamento (EZZ-ELDIN e EL-MOURSY e HAMED, 2015).

Por fim, o tipo de algoritmo de roteamento escolhido pode variar dependendo da proposta da NoC, visando atender diferentes requisitos de sistema. Para permitir uma melhor comparação entre os algoritmos apresentados, a Tabela 1 os classifica segundo os conceitos de origem versus distribuído, determinístico versus adaptativo, mínimo versus não-mínimo e consistência quanto ao congestionamento.

Tabela 1: Classificação dos algoritmos de roteamento

Roteamento	**XY**	**NL**	**NF**	**WF**
Origem x Distribuído	Distribuído	Distribuído	Distribuído	Distribuído
Determinístico x Adaptativo	Determinístico	Parcialmente adaptativo	Parcialmente adaptativo	Parcialmente adaptativo
Mínimo x Não-Mínimo	Mínimo	Parcialmente mínimo	Parcialmente mínimo	Parcialmente mínimo
Congestionamento	Inconsciente	Parcialmente consciente	Parcialmente consciente	Parcialmente consciente

Fonte: do autor

3 SISTEMAS MULTIAGENTE

A Inteligência Artificial (IA) é uma parte do campo de ciência da computação que visa a criação de sistemas inteligentes, ou seja, sistemas que são capazes de perceber o seu ambiente de atuação e realizar decisões que maximizem seu objetivo principal (RUSSEL e NORVIG, 2004). Segundo POOLE e MACKWORTH (1998), estes sistemas podem ser projetados com base no uso de agentes inteligentes. Os programas desenvolvidos nessa área geralmente são desenvolvidos sob duas perspectivas:

- Programas capazes de imitar o pensamento e o comportamento humano (KURZWEIL, 1990);
- Programas que pensam e atuam racionalmente, ou seja, são projetados para tomar decisões com base em seu conhecimento (CHARNIAK e MCDERMOTT, 1985).

Com a evolução da tecnologia de computadores, foi percebido que a IA poderia explorar os novos recursos disponíveis, como o paralelismo de execução viabilizado por sistemas multiprocessadores ou multicomputadores. Dessa forma, surge um novo conceito, a Inteligência Artificial Distribuída (IAD). Essa área tem por objetivo dividir os problemas que serão resolvidos pelos sistemas desenvolvidos em subproblemas, de forma que possa ser aproveitada toda a arquitetura de computadores disponível para resolver esses problemas de forma mais rápida (FERBER, 1999).

Os Sistemas Multiagente (SMA) são uma subdivisão da Inteligência Artificial Distribuída (IAD). Estes sistemas têm por objetivo estudar o comportamento dos agentes artificiais autônomos em um universo onde estes trabalham de forma coletiva com alguma finalidade. De maneira geral, um sistema multiagente é caracterizado por ser um conjunto de agentes autônomos, ou seja, que tem capacidade de pensar e interagir entre si e com o ambiente, trabalhando para atingir um bem maior coletivo

(AMANDI, 1997). Para atingir o objetivo coletivo, este pode ser dividido em objetivos menores, onde cada agente será capaz de resolver uma parte deste problema. Além disso, para solucionar o problema, os agentes precisam ser coordenados para desempenharem seus papéis em conjunto.

Um agente pode ser definido como uma entidade, seja ela real ou virtual, que apresenta um comportamento autônomo, capaz de agir sozinho, visando tomar as melhores ações possíveis dentro de seu conhecimento, para atingir objetivos. Estas entidades são capazes de perceber o mundo exterior através de sensores e agir neste mundo empregando atuadores (RUSSEL e NORVIG, 2004). Estes também têm a capacidade de interagir entre si, através da troca de mensagens, e também com o ambiente onde estão inseridos, percebendo alterações e até mesmo alterando este ambiente. Os agentes possuem objetivos, sendo que o objetivo coletivo prevalece, crenças e ações. Para atingir os objetivos, os agentes fazem uso de seus recursos. Como normalmente os agentes são analisados em conjuntos, formando sistemas multiagente, estes podem possuir uma representação parcial do ambiente, pois seria difícil e custoso se cada agente tivesse que analisar o ambiente inteiro. Visando melhorar o desempenho das simulações desenvolvidas, normalmente os agentes trabalham com visões parciais, ou seja, veem apenas o que está à sua volta.

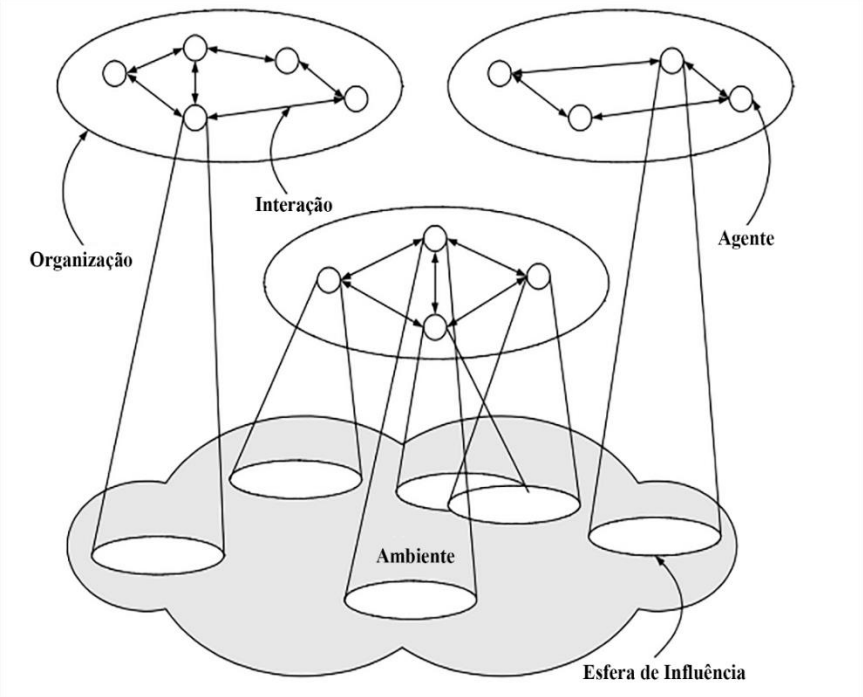
Dentre as diversas características que classificam os agentes, são destacadas as seguintes categorias, quanto à sua resposta para eventos no ambiente e/ou nos demais agentes (HUBNER, 2003):

- **Agentes reativos simples**: respondem a percepções, interpretando entradas de sinal, verificando a regra para tal sinal correspondente e agindo.
- **Agentes reativos baseados em modelo**: mantêm o estado interno para aspectos não percebidos. O estado interno modela o mundo e pode utilizar percepções do ambiente para auxiliar em decisões.
- **Agentes cognitivos**: possuem mecanismos de tomada de decisões avançados, como a troca de mensagens com outros agentes para obter mais informações e tomar decisões melhores. Por conta dessas trocas de mensagens, as interações dos agentes cognitivos são consideradas sofisticadas. Por fim, estes agentes tem um objetivo fortemente estabelecido.

- **Agentes baseados em objetivos**: estes agentes podem mudar de objetivos para atingir os objetivos globais. Por conta desse comportamento dinâmico, estes agentes são considerados mais flexíveis que agentes reativos.
- **Agentes baseados na utilidade**: este tipo de agente pode mudar suas estratégias visando maximizar a probabilidade de sucesso em relação à importância do objetivo.

A Figura 13 ilustra a estrutura de um sistema multiagente. Nela cada agente possui sua própria capacidade de percepção e sua esfera de influência no ambiente, ou seja, os agentes podem influenciar o ambiente de forma diferente uns dos outros (REIS, 2003).

Figura 13: Esferas de influência dos agentes no ambiente



Fonte: (REIS, 2003)

As aplicações desenvolvidas sob o ponto de vista dos SMA têm normalmente o objetivo de fazer uma simulação de algum cenário do mundo real, melhorando sua teoria, modelo e projeto, implementando sistemas mais complexos. O objetivo global é decomposto em elementos e suas interações. A partir disto, cada elemento é criado como um agente, o que resulta em modelo global onde os agentes interagem entre si e o ambiente (FROZZA, 1997) (OPREA, 2017).

Segundo WOOLDRIDGE (2009), existem diversos fatores que apontam para a possibilidade de uma abordagem baseada em agentes, tais como:

- Uso de um ambiente aberto, dinâmico ou complexo – Existem diversos tipos de ambientes onde os agentes podem estar inseridos, como ambientes estáticos, ambientes onde determinadas características podem mudar e ambientes com muitas variáveis. Nestes ambientes complexos, sistemas com ações autônomas flexíveis são utilizados por serem provavelmente a única solução viável.
- Agentes são uma metáfora natural – Muitos ambientes, como organizações ou ambientes competitivos ou comerciais, são modelados naturalmente como uma sociedade de agentes, estejam estes agentes cooperando uns com os outros para solucionar problemas complexos, ou competindo.
- Distribuição de dados, controle ou perícia – Em ambientes como um sistema de banco de dados distribuído, a distribuição de dados, controle ou perícia significa que utilizar uma abordagem centralizada é na melhor das hipóteses extremamente difícil ou, na pior das hipóteses, impossível. Em uma modelagem de agentes, cada banco de dados pode ser tratado como um componente semiautônomo.
- Sistemas legado - Sistemas legado são programas que utilizam tecnologias obsoletas, mas que seu funcionamento é essencial para uma organização. Estes sistemas geralmente não podem ser descartados por conta do seu custo de reescrita à curto prazo. Existem ocasiões onde estes sistemas legados precisam realizar interações não planejadas inicialmente com outros sistemas. Para prover a interoperabilidade, os sistemas do tipo legado podem interagir através de uma "camada de agente", que permite que eles se comuniquem e cooperem com outros componentes de *software*. (GENESERETH e KETCHPAEL, 1994).

As aplicações que exploram as características de um SMA se mostram mais eficientes quando comparadas a aplicações com abordagem monolítica, que é uma arquitetura mais antiga. Dentre as vantagens, é possível citar:

- Redução no volume de dados de comunicação – Para atingir esta diminuição, os agentes trocam mensagens curtas, que contém apenas uma pequena parte de dados parciais;
- Maior rapidez na resolução de problemas grandes – Os problemas grandes são resolvidos através da exploração do paralelismo, executando pequenas partes simultaneamente;
- Maior flexibilidade – Os agentes podem possuir características e funções diferentes, mas são coordenados de forma a atingir um objetivo único;
- Maior segurança – Em cenários onde os agentes podem falhar, estes agentes falhos podem ser substituídos por outros agentes similares.

Além da eficiência, os sistemas multiagente são utilizados para entender comportamentos que emergem de sistemas complexos. Simulações deste tipo são conhecidas como Modelagem e Simulação baseada em Agentes (ABMS, do inglês *Agent-based Modeling and Simulation*).

As simulações que utilizam o paradigma ABMS fazem o uso de agentes autônomos e sistemas multiagente para reproduzir e explorar um fenômeno em investigação. Este paradigma pode ser utilizado em diversas áreas de aplicação, como tráfego, ecologia, economia e epidemiologia (MACAL e NORTH, 2014).

3.1 NetLogo

O NetLogo, ferramenta utilizada para o desenvolvimento do simulador apresentado neste documento, é um ambiente de programação para o desenvolvimento de sistemas multiagente, criado por Uri Wilensky (WILENSKY, 1997). A ferramenta conta com uma biblioteca vasta de modelos implementados de diversas áreas, como artes, biologia, química, física, ciência da computação, ciências naturais, economia, ciências sociais e matemática (PEREIRA, 2015).

O NetLogo possui uma estrutura básica de programação utilizada para simular os principais componentes de um sistema multiagente. Esta ferramenta é multiplataforma, ou seja, funciona em diversos sistemas operacionais. Além disso, possui uma interface gráfica de fácil manipulação. O NetLogo permite a integração com

componentes externos, como a linguagem de programação Java e a comunicação com banco de dados (OMENA, 2014).

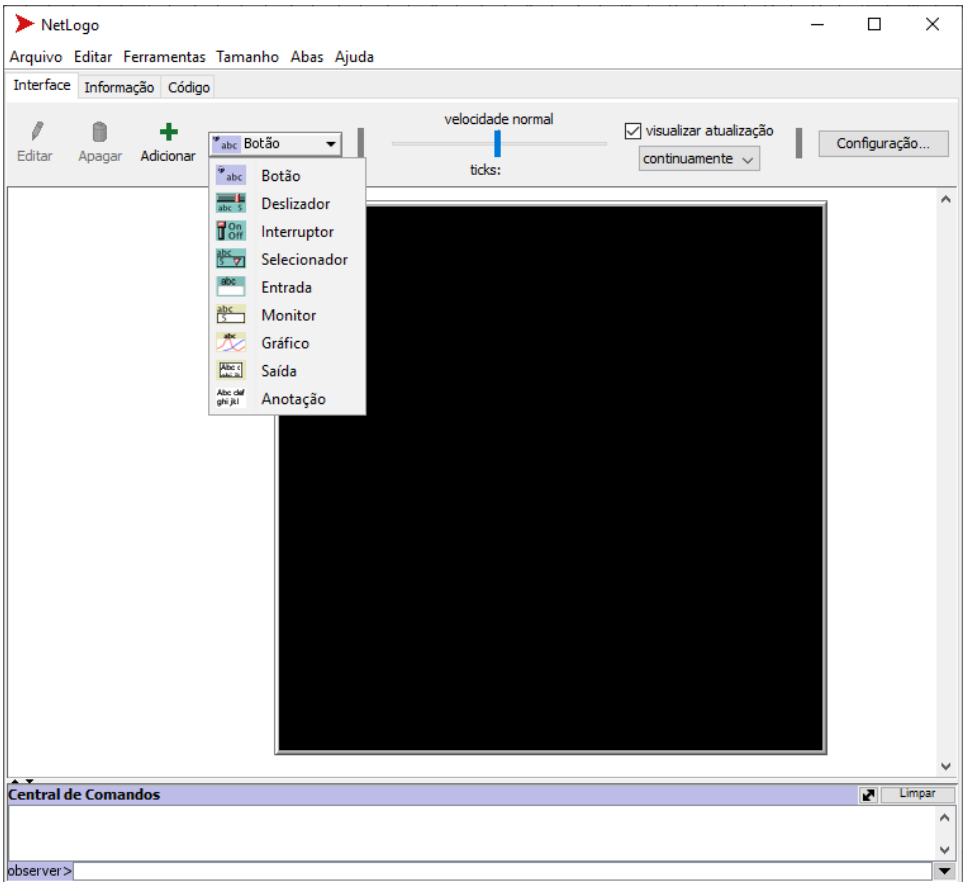
A linguagem de programação utilizada pelo NetLogo é própria, sendo baseada na linguagem Logo (GREGOLIN, 1994). A linguagem NetLogo dispõe de elementos que permitem a programação orientada a agentes, como variáveis, espécies (tipo de agentes) e funções (procedures).

O NetLogo trabalha com variáveis de tipagem dinâmica, ou seja, não é necessário especificar o tipo da varável na declaração. As variáveis podem ser *locais*, onde o acesso é privado apenas ao escopo onde a variável foi declarada, *globais*, podendo ser acessadas de qualquer parte do código, *de agentes*, que tratam de um tipo de variável similar aos atributos das linguagens orientadas a objetivos, ou seja, um determinado tipo de agente tem uma característica, ou *de patches*, que são variáveis ligadas à parte do ambiente (OMENA, 2014).

O NetLogo permite a criação de espécies, formando diferentes tipos de agentes. A espécie primitiva é a *turlte*. Utilizando diferentes espécies, é possível determinar características específicas para cada tipo de agente, tais como a forma do agente, tamanho e a posição.

O NetLogo conta com uma interface gráfica, o que torna os programas desenvolvidos nessa ferramenta mais amigáveis ao usuário. A Figura 14 apresenta uma ilustração da estrutura inicial da interface do NetLogo. O quadrado preto representa o ambiente onde os agentes são inseridos. O menu acionado mostra os tipos de entradas e saídas que podem ser apresentadas na interface do usuário, como botões, deslizadores, selecionadores, entradas, monitores, gráficos, saídas e anotações. Também é possível observar na parte superior a opção que define a velocidade com que os ciclos serão executados. A aba *Interface* apresenta a interface gráfica desenvolvida. A aba *Informação* é o local onde o desenvolvedor pode incluir informações sobre o modelo desenvolvido para que o usuário possa consultar. A aba *Código* é onde todas as linhas de programação do modelo são apresentadas.

Figura 14: Interface do NetLogo



Fonte: do autor

O aprendizado da ferramenta pode se dar através da análise no código dos diversos modelos já implementados ou então através da documentação. A documentação é bem completa, apresentando cada uma das funções com sua respectiva explicação e exemplo de uso.

Por fim, o NetLogo funciona através de funções ou *procedures*. Cada função criada pelo usuário desempenha um papel dentro da simulação criada. É importante ressaltar que a interface e o código compartilham as mesmas variáveis. Dessa forma, quando o valor de uma variável *X* é alterado na interface, essa alteração é refletida no código.

4 SIMULADORES DE NOC

Na literatura, os simuladores de NoCs são classificados por diversos fatores, como os parâmetros de entrada disponíveis, os tipos de resultados obtidos na simulação, o tipo de licença, a interface com o usuário, análise de consumo de energia e padrões de geração de tráfego. Pesquisas abrangentes sobre simuladores de NoCs são apresentadas por (ACHBALLAH e SAOUD, 2013) e (ALALAKI e AGYEMAN, 2017). Em (ACHBALLAH e SAOUD, 2013), os simuladores são classificados de acordo com os parâmetros de entrada e as métricas de saída. Em (ALALAKI e AGYEMAN, 2017), é apresentada uma comparação entre os simuladores de acordo com características como o *framework* utilizado, as topologias suportadas, o suporte a GUI (*Graphical User Interface*), ferramentas de *benchmark*, a data de lançamento e o tipo de funcionamento síncrono/assíncrono. Dessa forma, este capítulo apresenta uma visão geral sobre os simuladores de NoCs existentes, indicando alguns simuladores encontrados na literatura em conjunto com suas principais características. Dentre os simuladores, serão destacados os que possuem mais características em comum com o proposto neste estudo, apresentando além das semelhanças quais são as diferenças.

A classificação dos simuladores de NoCs pode levar em consideração diversas características. Por exemplo, ACHBALLAH e SAOUD (2013) analisam os simuladores sobre o ponto de vista dos seguintes parâmetros:

- NS – *Network Size* – Tamanho da rede. Determina quantos nodos serão gerados no eixo X e no eixo Y para a simulação;

- BS – *Buffer Size* – Tamanho dos *buffers*. Este parâmetro indica a capacidade dos buffers em armazenar informação sobre os pacotes. Normalmente é atribuída em *bytes* ou em número de *flits*;

- PD – *Packet Distribution* – Distribuição de pacotes. Este parâmetro está relacionado à geração de carga, determinando a forma com que os pacotes serão distribuídos durante a simulação;

- RA - *Routing Algorithm* - Algoritmo de roteamento. Este parâmetro escolhe o algoritmo de roteamento que pode ser utilizado para simulações;

- PIR – *Packet Injection Rate* – Taxa de injeção de pacotes. Parâmetro que define a taxa de injeção de pacotes na rede, para simular tráfego;

- SS – *Selection Strategy* – Estratégia de seleção ou árbitro. Determina quais estratégias de árbitros estão disponíveis para serem escolhidas em uma simulação;

- TD – *Traffic Distribution* – Distribuição de tráfego. Outra opção relacionada à geração de tráfego. Determina como é feita a distribuição do tráfego gerado pelo simulador ao longo da execução;

- AC – *Area Consumption* – Consumo de área. Apresenta qual será o tamanho do circuito que a NoC irá produzir;

- PC – *Power Consumption* – Consumo de potência. Esse parâmetro mostra qual será o consumo de potência do circuito gerado pela NoC;

- *T* – *Throughput* – Vazão da rede. Determina quantos pacotes a rede é capaz de processar a cada ciclo;

- *L* – *Latency* – Latência. Apresenta quantos ciclos leva para o pacote ou *flit* partir do roteador origem até chegar ao nodo destino. Normalmente são computados valores mínimos, máximos e médios para latência e vazão;

- *S* - Síntese de *Hardware*. Parâmetro que define se o simulador consegue gerar uma equivalência da NoC em um circuito de baixo nível, para implementação*;*

- *D* - Disponibilidade (licença). Este parâmetro define se o simulador está disponível para uso, se sua licença de código é livre para uso ou possui licença proprietária.

No estudo de ACHBALLAH e SAOUD (2013), os autores destacam alguns simuladores. O NS-2 (NS-2, 2018), que é um simulador originalmente desenvolvido para prototipagem e simulação de redes de computadores, pode ser utilizado para simular NoCs, através de algumas abstrações. O Noxim (CATANIA, 2015) é um

simulador que permite ao usuário alterar parâmetros da NoC, como tamanho da rede e de *buffers*, algoritmo de roteamento e geração de tráfego. O Darsim (LIS, 2010) é outro simulador apresentado no estudo. Nele, também é permitida a configuração de parâmetros de forma similar ao Noxim, além de simular malhas 2D e 3D. O SunFloor (SEICULESCU, 2009) é um simulador capaz de realizar síntese de *hardware*, além de apresentar modelos de consumo de energia. Já o ORION2.0 (KAHNG, 2009) é um simulador com enfoque na estimativa de consumo de energia e de área para as arquiteturas de NoCs.

Com o objetivo de apresentar as características específicas dos simuladores citados, a Tabela 2 mostra como os simuladores são classificados. Além dos simuladores citados, a tabela também apresenta informações sobre os simuladores: ATLAS (ATLAS, 2018), PIRATE (PALERMO e SILVANO, 2004), SUNMAP (MURALI, 2004), uSpider (EVAIN, 2004), NoCGEN (CHAN e PARAMESWARAN, 2004), FlexNoC (FLEXNOC, 2018), iNoC (INOC, 2018) e The CHAIN works tool suíte (CHAIN, 2018). O ATLAS (ATLAS, 2018) é um simulador com ênfase na simulação da NoC Hermes (MORAES, 2004) e suas variações. O PIRATE (PALERMO e SILVANO, 2004) é um simulador que permite a avaliação de diferentes tipos de topologia. O SUNMAP (MURALI, 2004) também tem foco em topologias, dando opções de seleção automática para a topologia que mais se adapta a uma determinada aplicação. O uSpider (EVAIN, 2004) é um simulador que suporta diferentes níveis de qualidade de serviço (QoS). O NoCGEN (CHAN e PARAMESWARAN, 2004) é um simulador que permite o uso de diferentes larguras de bandas, tamanho de buffer e a conexão entre os roteadores pode ser descrita linguagens de descrição de *hardware*. Os simuladores FlexNoC (FLEXNOC, 2018), iNoC (INOC, 2018) e The CHAIN works tool suíte (CHAIN, 2018) se destacam por possuírem todos os atributos que são usados para classificação dos simuladores no trabalho, porém possuem licença do tipo comercial, limitando sua utilização.

Na Tabela 2, o símbolo + representa se o simulador possui ou não a característica apresentada na coluna. A coluna de disponibilidade tem um destaque por possuir três opções: o símbolo - significa que não há versão para teste, + significa que existe versão para teste e o código é aberto e a opção “Comercial” significa que existe versão para teste, mas o código utiliza licença proprietária.

Tabela 2: Classificação de Simuladores – Estudo 1

#	Ferramenta	NS	BS	PD	RA	SS	PIR	TD	AC	PC	T	L	S	D
1	NS-2	+	+	+	+	+	+	+	+	+	+	+	-	+
2	NOXIM	+	+	+	+	+	+	+	-	+	+	+	-	+
3	DARSIM	+	+	+	+	+	+	+	-	-	+	+	-	-
4	SunFloor – 3D	+	-	-	-	-	-	-	-	+	+	+	+	-
5	ORION 2.0	-	+	-	-	-	-	-	+	+	-	-	-	-
6	ATLAS	+	-	+	+	-	+	-	-	-	+	+	+	+
7	PIRATE	+	+	-	-	-	-	-	+	+	-	-	-	-
8	SUNMAP	+	+	-	-	-	-	-	-	+	+	+	+	-
9	uSpider	+	+	-	-	-	-	-	-	-	+	+	+	-
10	NoCGEN	-	+	-	-	-	-	-	-	-	+	+	+	-
11	FlexNoC	+	+	+	+	+	+	+	+	+	+	+	+	Comercial
12	iNoC	+	+	+	+	+	+	+	+	+	+	+	+	Comercial
13	The CHAIN works tool suite	+	+	+	+	+	+	+	+	+	+	+	+	Comercial

Fonte: adaptado de (ACHBALLAH e SAOUD, 2013)

Na Tabela 2 também é possível perceber que os simuladores que permitem a análise de algoritmos de roteamento são: NS-2 (NS-2, 2018), Noxim (CATANIA, 2015), DARSIM (LIS, 2010), ATLAS (ATLAS, 2018), FlexNoC (FLEXNOC, 2018), iNoC (INOC, 2018) e *The CHAIN works tool suíte* (CHAIN, 2018). Ainda é possível destacar que os simuladores FlexNoC, iNoC e *The CHAIN works tool suíte* possuem disponibilidade apenas comercial. Além disso, o ATLAS possui dependências que são proprietárias.

ALALAKI e AGYEMAN (2017) também apresentam um estudo sobre simuladores para NoCs. Os autores analisam os simuladores sobre outra perspectiva. No estudo, os autores levam em consideração:

- Ano – Indica o ano de em que a ferramenta foi lançada;

- *Benchmark* – Indica os simuladores que disponibilizam ferramentas para teste de desempenho;

- *Framework* – Apresenta a linguagem ou biblioteca utilizada para o desenvolvimento do simulador;

- *Open-Source* – Mostra o tipo de licença de código que o simulador apresenta. Pode ser do tipo código aberto (*Open-Source*) ou não;

- Topologias – Indica as topologias que são suportadas na simulação. São consideradas as topologias malha, toróide e árvore;

- Heterogêneas – Capacidade da ferramenta em simular NoCs com fio e sem fio (*wireless*);

- GUI – Mostra se o simulador possui Interface gráfica;

- Funcionamento – Indica se a ferramenta pode simular NoCs com funcionamento síncrono, assíncrono ou ambos.

No estudo de ALALAKI e AGYEMAN, os autores destacam os simuladores: Booksim (JIANG, 2013), Darsim (LIS, 2010), Noxim (CATANIA, 2015), Gem5 (NIKOUNIA e MOHAMMADI, 2015), HNOCS (BEN, 2013) e o Sniper (CARLSON, 2015). O Booksim é um simulador que permite a análise de modelos de tráfego, microarquitetura dos roteadores, e é aplicado em estudos de roteamento, topologia, controle de fluxo e qualidade de serviço. O Noxim e o Darsim são observados sobre outras características além das observadas no estudo anterior. O Darsim permite a execução de *benchmarks*, enquanto o Noxim não tem essa opção. Além disso, o Noxim permite a simulação de redes com e sem fio, enquanto o Darsim simula apenas redes com fio. O Gem5, que surgiu da junção das características dos simuladores GEMS e M5 *simulator*, apresentando alta configurabilidade, diversos modelos de núcleos para simulação e modelos de *cache*. O HNOCS, ou *Heterogeneous NoC Simulator*, é um simulador que suporta simulações paralelas, mecanismos de avaliação de qualidade de serviço, tecnologias para árbitros e estimativas de consumo de energia, além de utilizar licença do tipo código aberto. Dos simuladores apresentados no estudo, o HNOCS é o único que possui suporte a funcionamento síncrono e assíncrono. Por fim, o Sniper é um simulador que permite a execução de paralelismo e precisão X86 para simulações heterogêneas.

Na Tabela 3 é possível observar a classificação dos simuladores analisados segundo os critérios do estudo de ALALAKI e AGYEMAN. Além dos simuladores citados, no estudo e na tabela também constam informações dos simuladores: Gpnocsim (HOSSANI, 2008), GEM5 - GPU (POWER, 2014), AdapNoC (KAMALI e HESSABI, 2016), ENoCS (VIGLUCCI e CARPENTER, 2016), MPSoCSim (WEHNER, 2015) e o DART (WANG, 2014). O Gpnocsim (HOSSANI, 2008) é um simulador de uso geral, desenvolvido em JAVA, que conta com diversos padrões de tráfego, e permite incluir novos padrões na arquitetura da simulação. O GEM5 - GPU (POWER, 2014) é um simulador de NoCs permite a simulação de NoCs que podem utilizar tanto CPUs quanto

GPUs, ou até mesmo redes híbridas utilizando os dois. O AdapNoC (KAMALI e HESSABI, 2016) é um simulador que, além do funcionamento do *clock* comum, permite a utilização de um *clock* virtual, que diminui o tempo de simulação de processos que não são paralelos. O ENoCS (VIGLUCCI e CARPENTER, 2016) é um simulador que foi incorporado em um curso de arquitetura de computadores, onde foi constatado que os estudantes que utilizaram o simulador puderam compreender melhor os conceitos de NoCs, além de demonstrar interesse em continuar utilizando o simulador em questão. O MPSoCSim (WEHNER, 2015) é um simulador que possui diversos geradores de tráfego. Os autores do MPSoCSim destacam a sua utilização no estágio inicial de levantamento de requisitos dos sistemas, para realizar estimativas. O DART (WANG, 2014) é um simulador que utiliza linguagem de descrição de *hardware*. Os autores destacam que a utilização destas linguagens permite a quebra do funcionamento geral do simulador em pequenos módulos, que podem ser facilmente substituídos sem afetar a execução geral do simulador.

Esta tabela adota símbolos semelhantes à Tabela 2. A coluna de topologia tem um destaque por possuir as opções *algumas*, quando disponibiliza apenas parte das topologias citadas, ou *todas* quando possui suporte a todas as topologias listadas. Além disso, a coluna de funcionamento tem os parâmetros *Síncrono*, caso a NoC utilize uma execução síncrona, ou *Ambos*, se tiver suporte para estruturas síncronas e assíncronas.

Tabela 3: Classificação de Simuladores – Estudo 2

Simulador	Ano	*Benc h.*	*Framework*	*Open-Source*	Topologia	Heterogê nea	GUI	Funciona mento
Booksim	2010	-	C++	+	Algumas	-	-	Síncrono
DARSIM	2010	+	C++	+	Todas	-	-	Síncrono
Gem5v	2015	+	C++	+	Algumas	-	-	Síncrono
NOXIM	2010	-	SystemC	+	Algumas	+	-	Síncrono
HNOCS	2013	-	OMNet++	+	Todas	+	+	Ambos
Gpnocsim	2007	-	Java	+	Todas	-	-	Síncrono
Gem5-GPU	2015	+	C++	+	Todas	+	-	Síncrono
AdapNoC	2016	-	C++	-	Algumas	-	-	Síncrono
ENoCS	2015	-	Java	+	Algumas	+	+	Síncrono
MPSoCSim	2015	+	SystemC	+	Algumas	+	+	Síncrono
DART	2014	+	SystemC	+	Todas	+	+	Síncrono
Sniper	2015	+	SystemC	+	Algumas	+	+	Síncrono

Fonte: adaptado de (ALALAKI e AGYEMAN, 2017).

Para encontrar estudos similares ao proposto neste trabalho, o enfoque da pesquisa foi em simuladores que possibilitem escolher e analisar algoritmos de roteamento. Além disso, também se observou se estes simuladores possuem licença de código aberto e permitem o uso de uma interface gráfica. Depois, foram observados quais parâmetros de entrada eram configuráveis e quais métricas eram obtidas na simulação. Por exemplo, foram observadas as capacidades dos simuladores de descrever o tamanho da NoC, o tamanho dos *buffers*, a geração de tráfego, a licença de uso e a interface com o usuário. Quatro simuladores apresentam semelhanças com o simulador proposto neste trabalho: o Noxim (CATANIA, 2015), o Booksim (JIANG, 2013), o DARSIM (LIS, 2010) e o NS-2 (NS-2, 2018).

O Noxim permite a configuração de parâmetros de entrada, como o tamanho da rede, dos *buffers* e dos pacotes. Como resultados, o simulador apresenta estatísticas de latência e consumo de energia. O Noxim tem a capacidade de configurar diversos parâmetros de maneira similar ao simulador aqui proposto, como o número de nodos, tamanho de *buffers*, tamanho dos pacotes (número de *flits*). A injeção de pacotes que geram o tráfego no Noxim segue uma distribuição de probabilidade discreta, enquanto o simulador proposto além do envio de pacotes seguindo uma distribuição aleatória, também disponibiliza o envio de mensagens manuais, e com a possibilidade de definir explicitamente um conjunto de nodos remetentes e destinatários para a troca de mensagens.

O Booksim, da mesma maneira que o Noxim, permite a configuração de diversos parâmetros para simular a NoC, como a quantidade de nodos, topologia, algoritmo de roteamento e possui diversos tipos de geração de carga. Além disto, uma semelhança interessante do Booksim com o simulador desenvolvido neste estudo é a possibilidade de associar custos, em número de ciclos, para ações como o cálculo de roteamento e transmissão de pacotes entre nodos.

O DARSIM possui diversos padrões de geração de carga, tais como: *transpose*, *bit-complement*, *shuffle*, *H.264 decoder profile* (LIS, 2010). Além disso, o DARSIM pode ler um arquivo de injeção de pacotes, para a utilização de um tráfego pré definido. Este simulador executa simulações paralelas, atingindo alto *desempenho*. Execuções paralelas permitem ao DARSIM comparar mudanças no *design* da NoC com alta velocidade quando comparado aos simuladores de NoCs tradicionais. O simulador

também permite a configuração de parâmetros de *hardware*, como o tamanho de banda e *pipelines*. O DARSIM usa portas separadas para entrada e saída com cada vizinho, onde cada porta tem *buffers* separados. Os roteadores podem ter tantas portas quanto o usuário julgar necessário. O simulador suporta diversas topologias, como malhas com múltiplos níveis ou até mesmo em formato de anel. No simulador proposto, as portas funcionam da mesma maneira que o DARSIM, com *buffers* separados para cada porta, e também para entrada e saída.

O NS-2 é um simulador frequentemente utilizado para simular e prototipar redes de computadores. Ele suporta simulações TCP, roteamento e protocolos *multicast* através de redes cabeadas ou sem fio. Como as NoCs e as redes de computadores têm diversas características em comum, o NS-2 foi utilizado em (ALI, 2006) para simular NoCs. No estudo, os autores utilizaram o NS-2 para simular um protótipo de protocolo de tolerância a falhas para NoCs. Para a simulação, os autores simularam uma NoC com a topologia malha e um algoritmo determinístico. Ao detectar que algum *link* apresentou defeito ou está com problemas, as tabelas de roteamento são atualizadas, e a NoC continua funcionando através dos caminhos que ainda estão disponíveis.

Tanto o Noxim quanto o Booksim e o DARSIM não possuem interface gráfica. Os três funcionam unicamente através de linha de comando, mecanismo para os usuários configurarem e executarem as simulações. O simulador proposto, através do uso de uma ferramenta de Sistemas Multiagente, disponibiliza uma interface gráfica amigável, que faz com que a ferramenta seja utilizada de maneira fácil. Os parâmetros de entrada são configurados através de deslizadores, selecionadores e botões, o que propicia que novos usuários possam explorar as mudanças na simulação de maneira intuitiva. A interface gráfica também permite que os usuários possam observar o comportamento da NoC em tempo de execução. Também é possível observar a simulação passo a passo na interface, podendo analisar comportamentos com a possibilidade de interromper e retomar a simulação conforme seja necessário. Outra limitação importante destes simuladores é a falta do detalhamento sobre quais comportamentos internos da NoC os simuladores levam em consideração durante sua execução. É possível que, ao tentar simular uma NoC nos simuladores, mesmo ao se utilizar o maior número de parâmetros parecidos (existem diferenças entre os parâmetros que cada simulador disponibiliza), a simulação não seja exatamente igual.

Este comportamento foi observado ao simular uma NoC 3x3, com a topologia malha, algoritmo de roteamento XY, com uma taxa de injeção de pacotes de 0.15, pacotes de 8 *flits* com uma simulação que dura 10.000 ciclos. Embora todos esses parâmetros tenham sido escolhidos para realizar uma simulação no Booksim e no Noxim, os resultados de latência e vazão fornecidos pelos simuladores foram muito diferentes. O NS-2 pode ser manipulado para simular NoCs mas, para que isso aconteça, algumas abstrações precisam ser implementadas manualmente pelo usuário, como fazer com que os blocos individuais da NoC funcionem como nodos, e que as conexões entre esses nodos funcionem como *links* do NS-2. Por fim, vale ressaltar também que, embora o DARSIM tenha licença do tipo *open-source*, não foi encontrada nenhuma versão disponível para *download* e teste.

Com a análise destes simuladores, é possível observar que existe espaço para desenvolver outros simuladores, com características específicas. Dentre as características relevantes que podem ser esperadas de simuladores, é possível destacar que estes disponibilizem a configuração de parâmetros de entrada das NoCs; apresentem indicadores de saída para analisar os resultados da simulação, ou possivelmente apresentarem resultados em tempo de execução; tenham uma interface com o usuário que seja fácil e intuitiva; tenham seu código aberto; e tanto o código quanto suas dependências sejam de acesso livre.

Além da análise geral de simuladores, também foi realizada uma busca por trabalhos envolvendo NoCs e SMA. Foi encontrado um estudo que já utilizam a abordagem cognitiva de agentes para o roteamento em NoCs (NAIR e SOODA, 2017). No estudo, os autores destacam a importância e a dificuldade em desenvolver um algoritmo de roteamento para NoCs eficiente, e propõem um roteamento cognitivo utilizando um algoritmo evolutivo. Este estudo faz inferência a um “agente inteligente”, capaz de agir na rede quando o algoritmo precisa ser computado. Vale ressaltar que, por esse algoritmo utilizar uma estrutura baseada em agentes, esse algoritmo inteligente poderia ser modelado e avaliado através do simulador proposto neste estudo.

5 SIMULADOR

Este capítulo tem por objetivo apresentar detalhes sobre a implementação do simulador de NoCs proposto no trabalho. Este simulador utiliza um alto nível de abstração, modelado como um sistema multiagente. O simulador tem como objetivo: permitir que os usuários simulem NoCs, podendo alterar uma série de parâmetros, como tamanho da rede (quantidade de nodos), tamanho dos pacotes, algoritmo de roteamento e geração de tráfego para a simulação; obter dados importantes durante e depois da simulação, como o tempo de execução, taxa de ocupação de *buffers* dos roteadores (média da quantidade de buffers ocupados ao longo da execução) e média de salto dos *flits* até o destino.

Existem diversas características desejáveis em um simulador de NoC, variando desde aspectos como a facilidade de uso e configuração, capacidade de interpretação dos resultados, desempenho ou tipo de acesso. Para o simulador proposto, optou-se por:

- Distribuição de código aberto – Diversos simuladores possuem licenças pagas ou então utilizam dependências proprietárias. Com o objetivo de facilitar a utilização do simulador, seja para fins acadêmicos ou para uso na indústria, o simulador desenvolvido tem seu código aberto. Desta forma, usuários e desenvolvedores podem fazer atualizações e adaptações no código;
- Facilidade de instalação – Alguns simuladores no mercado exigem a instalação e configuração de pacotes adicionais e dependências para garantir o funcionamento do simulador. Com o objetivo de criar um simulador que seja fácil de instalar e configurar, o simulador proposto não depende de bibliotecas externas, limitando-se apenas à instalação do NetLogo, *software* onde a ferramenta foi desenvolvida;

- Facilidade de configuração – Através do uso de uma interface gráfica, o usuário pode ajustar as configurações de entrada e obter os resultados da simulação, tornando o uso da ferramenta fácil e intuitivo. Além disso, o simulador não requer configurações de variáveis de ambiente ou personalização do ambiente de execução;
- Avaliação em tempo real – Através do uso de uma interface gráfica, o usuário pode visualizar o que está acontecendo na simulação em tempo real, com a possibilidade de interromper e retomar a execução passo a passo. Isto permite observar comportamentos momentâneos da rede como um todo, ou de estados internos dos roteadores. É importante ressaltar que os resultados gerados em uma simulação também são disponibilizados na forma de um arquivo de *log*;
- Alto nível de abstração – A simulação independe do *hardware* (sem síntese física). Detalhes de baixo nível são omitidos para permitir que a configuração e execução sejam simples. Além disso, com mais alta abstração é possível realizar simulações grandes;
- Extensibilidade – O simulador permite que novos algoritmos de roteamento possam ser implementados e acoplados ao simulador. Desta forma, pode-se testar e comparar novos algoritmos. De forma análoga, novos padrões de geração de carga também podem ser facilmente adicionados ao simulador;

5.1 NoCs e SMA

A análise das principais semelhanças entre os agentes e os roteadores é apresentada na Tabela 4. Nesta tabela, é possível observar a equivalência de cada característica de um agente de um SMA e de um roteador de uma NoC. O agente pode ser uma entidade real ou virtual, o roteador é uma entidade real. O agente está inserido em um ambiente, no caso do roteador, o ambiente é a própria NoC. Um agente pode se comunicar com outros agentes, e um roteador se comunica o tempo todo com outros roteadores, através de suas interligações. Um agente tem objetivos e satisfações, e um

roteador tem como seu principal objetivo realizar comunicações de forma eficiente. Um agente tem recursos próprios, da mesma maneira que um roteador, que possui *buffers* de entrada e saída e capacidade de processamento. Os agentes são capazes de perceber mudanças no ambiente, e os roteadores têm a capacidade de detectar congestionamento nos canais de comunicação com os vizinhos. Cada agente pode perceber apenas uma representação parcial do ambiente, da mesma forma que um roteador, que só é capaz de detectar congestionamento com os vizinhos que estão diretamente conectados.

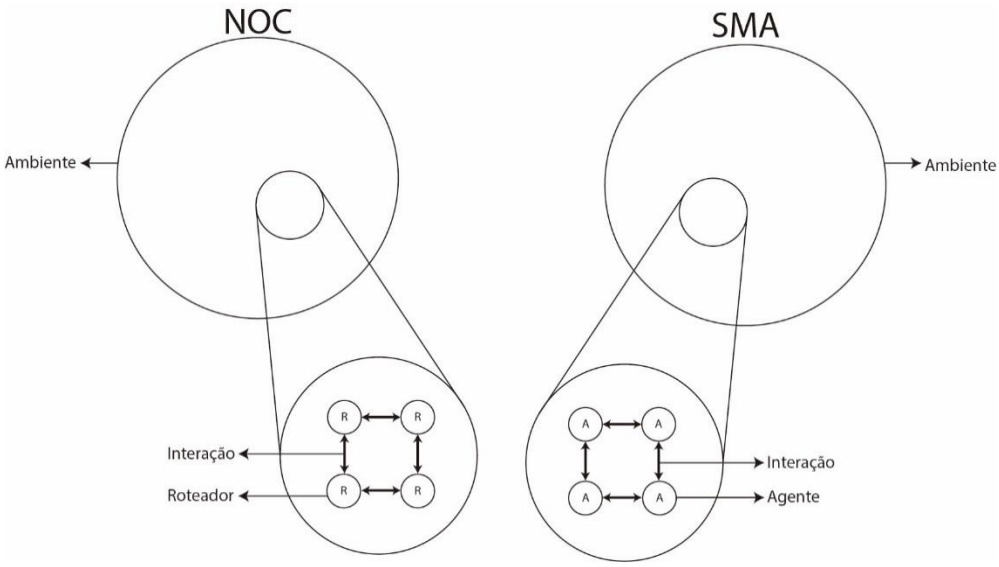
Tabela 4: Mapeamento entre componentes do agente e roteador

Agente	**Roteador**
Entidade Real/Virtual	Entidade Real
Ambiente	NoC
Comunicação com outros agentes	Enviar mensagens aos destinatários
Objetivos/Satisfações	Comunicação eficiente
Recursos Próprios	*Buffers* e processamento
Percebe o ambiente	Detecta congestionamento entre vizinhos
Representação parcial	Envia mensagens diretamente apenas para os vizinhos conectados

Fonte: do autor

A Figura 15 apresenta como elementos de um roteador de NoC podem ser descritos em um sistema multiagente. Nesta ilustração, são apresentados o ambiente e uma representação parcial dele, contendo agentes/roteadores e suas interações. Os círculos grandes representam ambientes, sendo à esquerda, o ambiente de uma NoC e, à direita, o ambiente de um sistema multiagente. Os círculos pequenos representam uma visão parcial deste ambiente. Na representação parcial, a interação entre os roteadores R é apresentada através das flechas bidirecionais, representando possíveis trocas de mensagens. Da mesma maneira, a ligação entre cada agente A é descrita por flechas bidirecionais, representando interações possíveis.

Figura 15: Analogia entre NoC e SMA

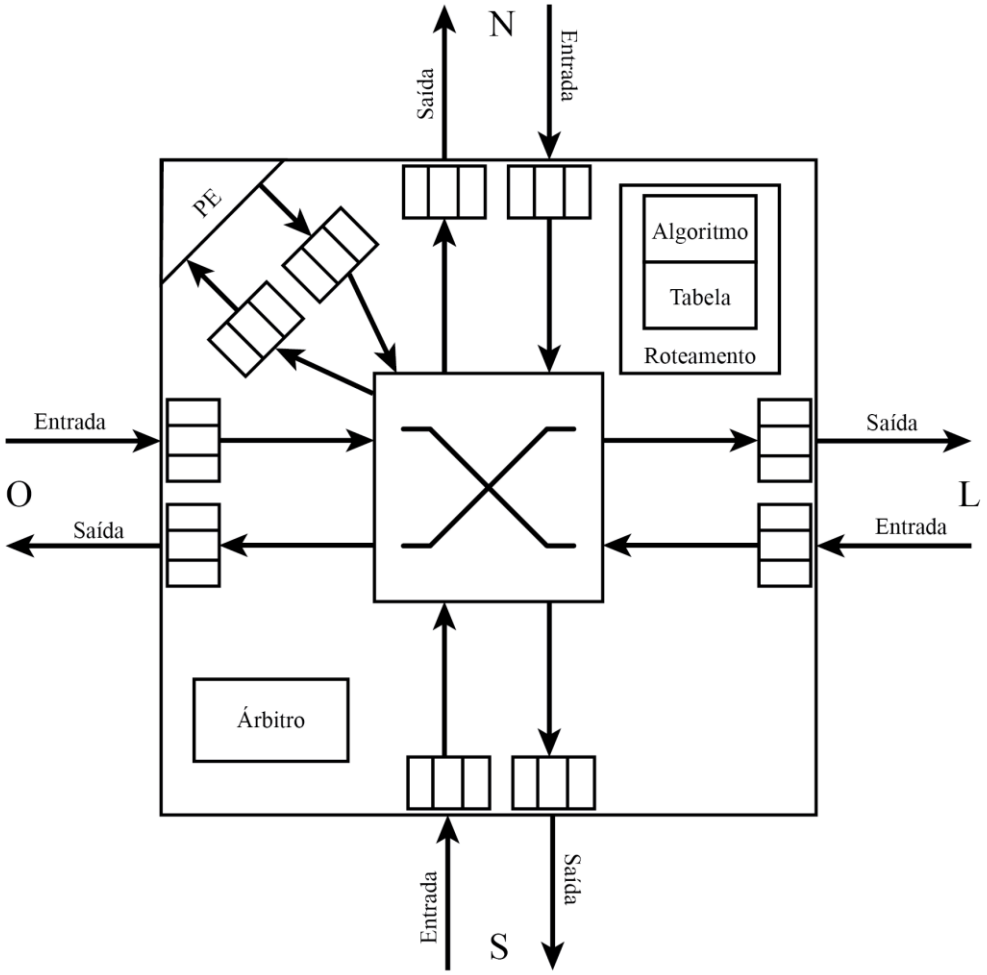


Fonte: do autor

5.2 Visão Geral

Para se ter uma visão geral do funcionamento do simulador, se faz necessário analisar a estrutura do roteador que o simulador implementa. A Figura 16 apresenta a estrutura interna de um roteador. Cada roteador comunica com os demais a partir de portas. São 8 portas para acesso externo, sendo 4 portas de entrada: Norte, Sul, Leste e Oeste; e 4 portas de saída: Norte, Sul, Leste e Oeste. Além disso, há duas portas, uma de entrada e outra de saída, para acesso à unidade de processamento (ou *Processing Element*, PE). Os roteadores utilizam *buffers* separados para entrada e saída em cada uma das 5 portas (Norte, Sul, Leste, Oeste e porta interna para o PE). Os *buffers* são implementados como listas, com espaço para armazenamento limitado e definido pelo usuário. Apenas um *flit* é retirado ou adicionado por vez e o acesso ao *buffer* se dá na ordem FIFO (*First-In, First-Out*). O tamanho destes *buffers* é definido pelo usuário.

Figura 16: Estrutura interna do roteador



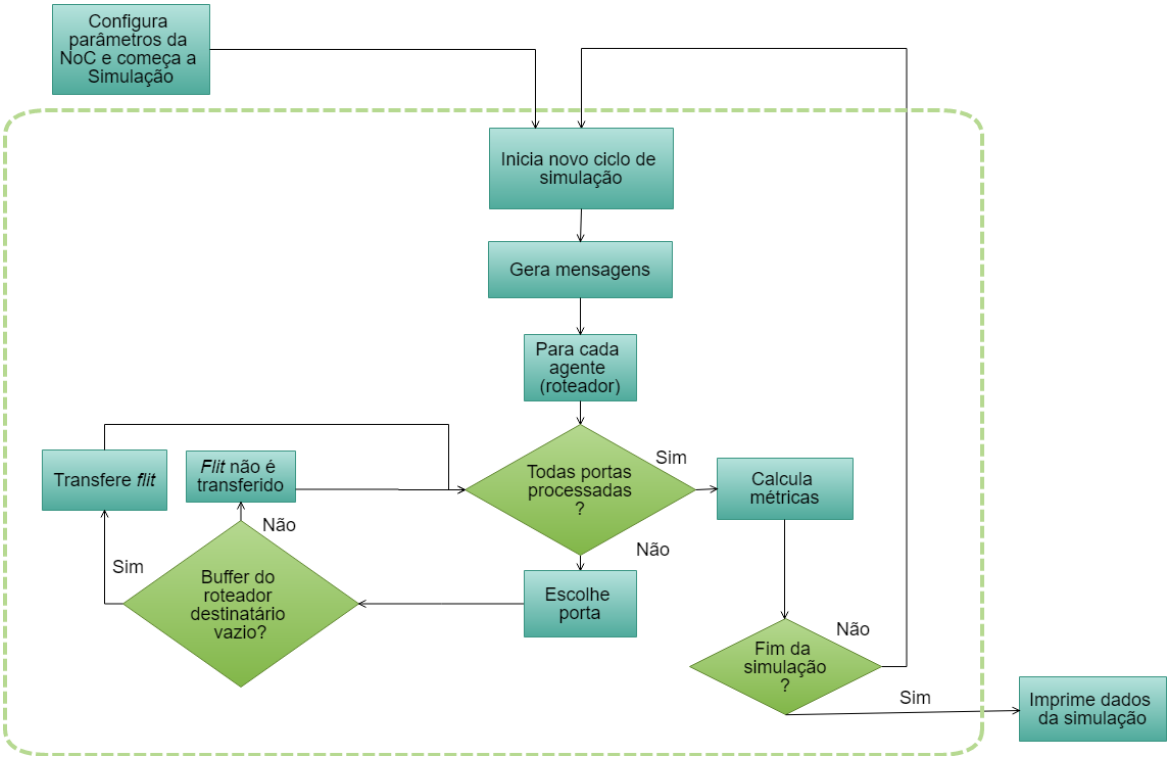
Fonte: do autor

Os roteadores armazenam as informações sobre a disponibilidade de cada porta para uso, para que seja possível verificar se um *flit* pode ou não ser enviado a uma porta. As portas do roteador são interligadas através do *crossbar*. Quando um *flit* deve ser transferido, o *crossbar* realiza a interconexão do *buffer* de entrada da porta origem com o *buffer* de saída da porta destino. Uma vez que uma porta é alocada para um pacote, nenhum outro pacote pode acessar a porta, até que a mesma seja liberada. Esse comportamento da alocação da porta acontece porque o simulador utiliza um controle de fluxo baseado em créditos (NUNES, 2015). Quando duas ou mais portas tentam acessar um mesmo recurso, o árbitro determina qual porta tem prioridade de acesso em cada ciclo de execução, por exemplo, adotando uma estratégia Round Robin. O roteamento é responsável por manter informações sobre quais pacotes estão sendo

encaminhados e tomar decisões para qual porta os *flits* de um pacote devem ser transferidos.

A Figura 17 descreve um fluxograma do comportamento geral do sistema.

Figura 17: Fluxo do funcionamento básico do simulador



Fonte: do autor

No fluxo principal, a simulação começa com a inicialização da NoC com os parâmetros fornecidos na interface do sistema. O laço de repetição principal, indicado pelo retângulo verde tracejado, é executado até que o total de ciclos atinja o número de ciclos da simulação, também escolhido na interface. A cada ciclo, todos os roteadores são processados um a um, de maneira síncrona. No início do processamento do roteador, os *flits* gerados pelo gerador de mensagens são transferidos para o *buffer* de entrada da porta do núcleo de processamento (PE). Em cada roteador com mensagens em trânsito, será escolhida uma porta de destino para o próximo *flit*, de acordo com o algoritmo de roteamento escolhido na inicialização da NoC. Para processar uma porta, é verificado se a primeira posição do *buffer* dessa porta contém um *flit.* Apenas a primeira posição é verificada, pois o acesso ao *buffer* respeita a ordem FIFO (*First-In, First-Out*). Caso exista um *flit*, verifica-se se o *buffer* destino (seja interno ou de outro roteador) tem uma posição livre e se o *flit* tem acesso a porta. Se as duas condições

forem satisfeitas, o *flit* é encaminhado para o seu destino. Caso contrário, o *flit* permanece no *buffer*, devendo aguardar o próximo ciclo. Uma vez que todas as portas do roteador sejam processadas, calcula-se as métricas. Quando todos os roteadores forem processados, inicia-se um novo ciclo, desde que o total de ciclos ainda não tenha sido atingido.

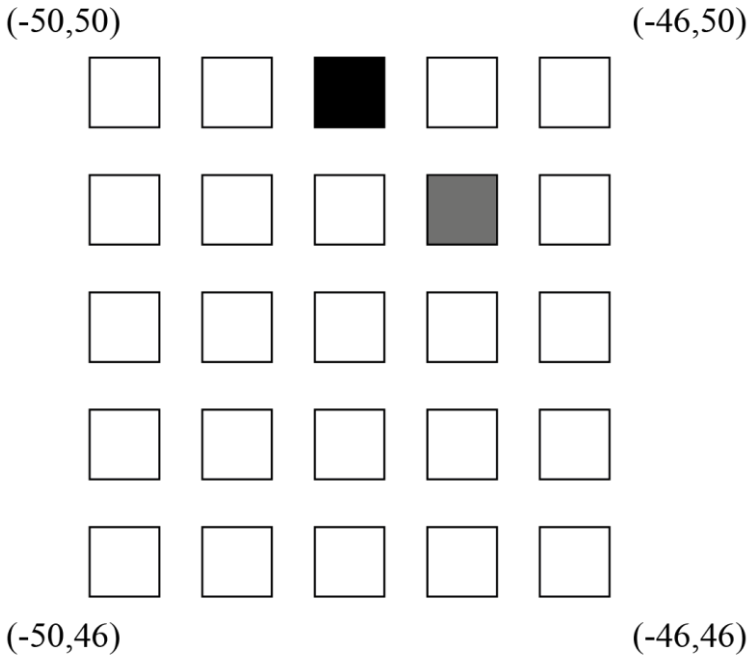
5.3 Detalhes de Implementação

Nesta seção a implementação do simulador será apresentada com mais detalhes. A seção conta com detalhes sobre a descrição da rede, os pacotes, árbitro, roteamento e da geração de carga.

5.3.1 Descrição da Rede

As redes descritas no simulador são representadas por malhas do tipo 2d. Os roteadores são inseridos na malha conforme o número de linhas (*L*) e colunas (*C*) escolhidos na interface. Com base nos valores *L e C* escolhidos, será gerada uma NoC contendo *L* linhas compostas de *C* roteadores cada. Os roteadores recebem seu identificador (id), começando com o id 0 até o id que corresponde ao total de roteadores menos 1. Já os valores de *x* e *y* que cada roteador recebe depende da quantidade de roteadores. A Figura 18 ilustra uma rede 5x5 montada segundo a malha do simulador. O primeiro roteador, com id 0, é sempre posicionado no canto superior esquerdo do ambiente do NetLogo, recebendo o valor *x* -50 e y 50. Os demais roteadores recebem os valores *x* e *y* baseados no total de roteadores por linha e coluna. No exemplo abaixo, por ser uma NoC 5x5, a primeira linha terá os roteadores com id de 0 a 4, a segunda linha de 5 a 9, e assim por diante. Ainda na figura, o roteador em preto representa o roteador com id 2, enquanto o roteador em cinza é o roteador com id 8.

Figura 18: Malha 5x5 com identificação das posições x e y



Fonte: do autor

5.3.2 Pacotes

Aplicações em uma NoC trocam mensagens, que podem ter tamanhos variados. Para permitir que quaisquer quantidades de informação possam circular na rede, estas mensagens são quebradas em pacotes. Os pacotes carregam, além do dado propriamente dito, informações adicionais, indicando, por exemplo, o início e fim de uma mensagem, assim como o endereço do nodo destino. O formato dos pacotes muitas vezes depende do tipo de chaveamento da NoC (NUNES, 2015). O simulador utiliza o chaveamento por pacotes do tipo *wormhole*.

No simulador, cada pacote é formado de pelo menos 3 *flits*, sendo um *header*, um *body* e um *tail*. Essa estrutura é mantida, pois cada tipo de *flit* desempenha funções diferentes ao longo da transmissão do pacote. Se a simulação utilizar pacotes de tamanho maior que 3, os *flits* adicionais são do tipo *body*. Por exemplo, ao configurar o

simulador com pacotes de tamanho 5, os pacotes serão formados por um *flit header*, 3 *flits body* e um *flit tail*.

Os *flits* do tipo *header* são sempre os primeiros a chegar nos roteadores pelos quais o pacote irá passar até chegar no destino. Dessa forma, quando um roteador recebe um *flit header*, são extraídas as informações do destinatário para executar o algoritmo de roteamento e determinar por qual porta os *flits* desse pacote devem seguir para chegar ao destino.

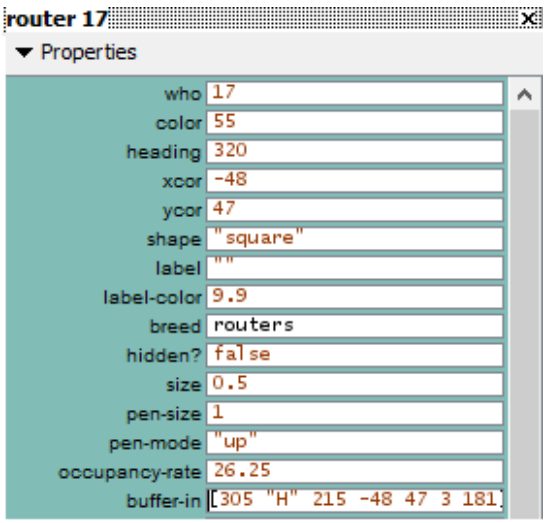
A Figura 19 apresenta parte dos atributos de um roteador específico dentro do simulador. Na linha *buffer-in* (última linha) é possível observar um *flit* que está armazenado no *buffer* de entrada. As informações armazenadas nos *flits* dependem se ele está armazenado no *buffer* de entrada ou de saída e do tipo do *flit*. Um *flit* em um *buffer* de entrada armazena 7 informações, no formato [ID "Tipo" Chegou X Y Hops Gerado]. Estas informações significam, respectivamente:

- Identificação do pacote – Essa informação serve para que seja possível identificar a qual pacote cada *flit* pertence. O identificador é utilizado na tabela de roteamento, indicando para qual porta os *flits* com o identificador em questão devem seguir, e também na ocupação das portas, pois o *header* reserva a porta para todos os *flits* do pacote;
- Tipo do *flit* (*header*, *body* ou *tail*) – Identifica o tipo de *flit*. A informação do tipo de *flit* é necessária para que o roteador saiba quais funções deve acionar, como executar o algoritmo de roteamento e reservar portas para *flits* do tipo *header* e fazer a exclusão dos dados da tabela de roteamento e liberação de portas para *flits* do tipo *tail*;
- Ciclo em que o *flit* chegou no roteador – Para manter o comportamento síncrono da execução, os *flits* armazenam o ciclo em que chegaram no roteador. Antes de processar um *flit*, os roteadores verificam se o ciclo que o *flit* chegou não é o mesmo ciclo atual. Dessa forma, se em um ciclo o *flit* avança de um roteador para outro, o segundo roteador irá constatar que recebeu o *flit* no mesmo ciclo atual de execução e, dessa forma, não irá processá-lo, evitando que o *flit* avance duas vezes em um mesmo ciclo de execução;

- Posição X do roteador destino – Identificação da posição X do roteador destino do *flit*;
- Posição Y do roteador destino – Identificação da posição Y do roteador destino do *flit*;
- Número de saltos (*hops*) até o momento – Esse dado é um contador do número de saltos que o *flit* dá até chegar ao destino;
- Ciclo em que o *flit* foi gerado – O *flit* armazena o ciclo em que foi gerado. Esse dado é utilizado para realização do cálculo da latência quando o *flit* chega ao destino. Para saber a latência, subtrai-se o ciclo atual do ciclo em que o *flit* foi gerado.

Pela identificação das informações apresentadas, o *flit* apresentado na linha sobre o *buffer-in* na Figura 19 pertence ao pacote 305, é um *flit* do tipo *header*, chegou no roteador no ciclo 215, o roteador destino do pacote deste *flit* está na posição cujo valor X/Y corresponde a -48/47, até o momento o *flit* deu 3 saltos e o *flit* foi gerado no ciclo 208.

Figura 19: Informações sobre o *buffer* de entrada de um roteador na interface do simulador

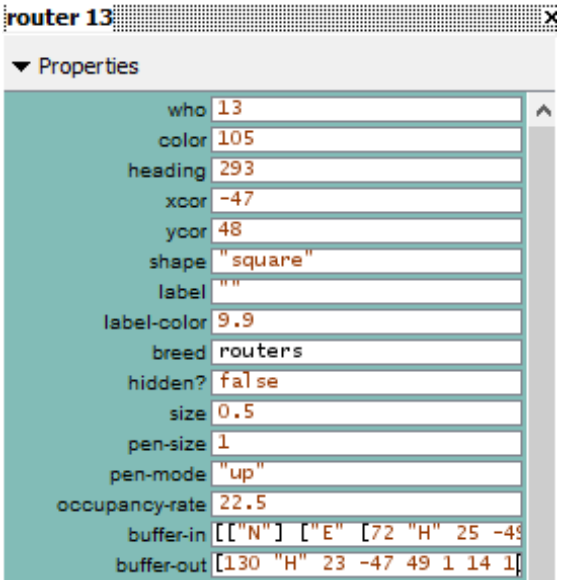


Fonte: do autor

Já quando os *flits* estão nos *buffers* de saída, além de ter todas as informações que são armazenadas quando estão no *buffer* de entrada, guardam mais um valor, que é o custo de transmissão, para que o roteador saiba quando pode rotear o *flit* para o

roteador destino. A Figura 20 apresenta parte dos atributos de um roteador específico dentro do simulador. Na linha *buffer-out* (última linha) é possível observar um *flit* que está armazenado no *buffer* de saída. Pelas informações apresentadas, o *flit* pertence ao pacote 130, é um *flit* do tipo *header*, chegou no roteador no ciclo 23, o roteador destino do pacote deste *flit* está na posição cujo valor X/Y corresponde a -47/49, até o momento o *flit* deu 1 salto, o *flit* foi gerado no ciclo 14 e ainda precisa aguardar 1 ciclo para que possa avançar para o próximo roteador (custo de transmissão).

Figura 20: Informações sobre o *buffer* de saída de um roteador na interface do simulador



Fonte: do autor

5.3.3 Árbitro

Durante a simulação, mais de um pacote, provenientes de portas diferentes podem requisitar a utilização de uma mesma porta destino. Cabe ao árbitro definir qual porta possui prioridade de acesso à porta destino. Para garantir justiça (*fairness*) na atribuição de acesso às portas, o simulador implementa um árbitro seguindo o algoritmo *Round-Robin* (HAHNE e GALLANGER, 1986). Este algoritmo faz com que a prioridade das portas mude a cada ciclo, de forma circular.

Supondo que em um ciclo *c1* a ordem de processamento das portas, ou seja em qual ordem os *buffers* terão seus *flits* analisados e encaminhados, seja Norte-Leste-Sul-Oeste-PE, no próximo ciclo *c2* as portas serão executadas na ordem Leste-Sul-Oeste-

PE-Norte, em *c3* Sul-Oeste-PE-Norte-Leste, em *c4* Oeste-PE-Norte-Leste-Sul, em *c5* PE-Norte-Leste-Sul-Oeste, de forma que em *c6*, que corresponde a 5 execuções após a execução inicial, as ordens de execução das portas se repetem de forma cíclica, sendo este comportamento mantido ao longo da execução.

A Figura 21 ilustra a função que implementa o árbitro. No caso do primeiro *ifelse* encontrado na Figura, é a implementação do árbitro atual, *round-robin*. O parâmetro *arbitrator* é escolhido na interface e é ele quem determina qual árbitro será executado. De maneira simplificada, a implementação do *round-robin* atual funciona através do deslocamento circular do *buffer-in*, para que a execução das portas aconteça conforme a ordem estabelecida pelo árbitro. Para isso, a primeira posição da lista é armazenada em uma variável temporária e é removida da lista, depois todas as outras posições são deslocadas para a esquerda, e então a variável temporária é inserida no final da lista.

Figura 21: Código do árbitro

```
173 to update-arbitrator
174   ifelse (arbitrator = "Round-Robin")
175   [
176     ask routers
177     [
178       let temporary-list []
179       set temporary-list item 0 buffer-in
180       set buffer-in remove-item 0 buffer-in
181       set buffer-in lput temporary-list buffer-in
182     ]
183   ]
184   [
185     ;;Para implementar outros árbitros
186   ]
187 end
```

Fonte: do autor

O simulador é extensível e pode ser alterado para implementar e testar novos árbitros. Basicamente, apenas a função apresentada na Figura 21, *update-arbitrator,* precisa ser alterada no simulador, que é a função que atualiza a ordem de execução das portas segundo o árbitro. Conforme observado, nesta função existe um conjunto de *ifelse* que verificam qual algoritmo foi escolhido para o árbitro na interface e ordena as

portas segundo esse algoritmo. Dessa forma, para incluir um novo árbitro, basta incluir o nome no seletor da interface e adicionar mais uma condição *ifelse* contendo o algoritmo que implementa a lógica do novo árbitro e este árbitro já estará disponível para simulação.

5.3.4 Roteamento

Ao receber um *flit header*, o roteador precisa, com base nas informações de destinatário contidas neste *flit*, determinar por qual porta de saída o *flit* deve ser roteado para que possa avançar em direção ao seu destino. Cabe ao algoritmo de roteamento escolher a porta mais adequada.

No simulador, o roteamento é dividido em duas partes: algoritmo e tabela de roteamento. O algoritmo de roteamento tem a função de determinar por qual porta o pacote deve seguir para que consiga chegar no roteador destino. Quando o roteador recebe um *flit* do tipo *header*, ele analisa as informações da posição X/Y do roteador destino que estão contidas no pacote e executa o algoritmo de roteamento para determinar por qual porta os *flits* do referido pacote devem sair. Depois de executar o algoritmo, é inserida uma informação na tabela de roteamento. Esta tabela armazenas as informações para que o roteador saiba para qual porta deve encaminhar *flits*, sem a necessidade de executar novamente o algoritmo de roteamento quando um *flit* deve ser transferido.

Após calcular o algoritmo de roteamento, é adicionada uma informação na tabela de roteamento do roteador, contendo informações como a identificação do pacote e a porta pela qual os *flits* desse pacote devem seguir. Depois de escrever na tabela de roteamento, o pacote só é enviado para a porta de saída quando o algoritmo de roteamento terminar de ser executado e a porta em questão estiver liberada. Quando o *flit header* for para a porta de saída, ele reserva da porta, fazendo com que apenas os *flits* que fazem parte deste mesmo pacote possam avançar para esta porta. Todo este comportamento é repetido por todos os roteadores pelos quais o pacote passar, ou seja, de maneira resumida o *header* tem como função abrir espaço para os demais *flits* chegarem no roteador, executando o algoritmo de roteamento para determinação da porta de saída e reservando estas portas.

Já os *flits* do tipo *body* chegam depois dos *flits header*, portanto, tem um comportamento mais simples. Quando um roteador recebe em uma de suas entradas um *flit body*, entende-se que o *header* deste pacote já passou pelo roteador, portanto, não é necessário realizar novamente o cálculo do algoritmo de roteamento. O roteador irá consultar a tabela de roteamento para determinar para qual porta deve enviar o *flit*. Além de consultar a tabela, o roteador também deve verificar se a porta solicitada está reservada para o *flit*. Se estiver reservada, o *flit* avança para a porta de saída, caso contrário, o *flit* deve aguardar pela liberação da porta para que possa avançar.

Por fim, o último tipo de *flit* a ser recebido é o *tail*. Ele tem por objetivo encerrar as alocações feitas pelo pacote. Portanto, quando o *flit tail* avança para a porta de saída, ele faz a remoção da informação da tabela de roteamento, para evitar o acúmulo de informações que não serão mais utilizadas, uma vez que todos os *flits* já passaram. Além disso, quando o *tail* avança de um roteador para outro, ele também faz a liberação da porta que estava em uso, para que outros pacotes possam acessar a porta, conforme a ordem de prioridade determinada pelo árbitro.

É importante ressaltar que todo o avanço de *flit* respeita o espaço disponibilizado pelo *buffer* da porta. Dessa forma, se um *flit* tem a permissão para avançar, seja de um *buffer* interno para outro, ou então para o *buffer* de outro roteador, primeiramente é verificado se existe uma posição livre no *buffer* destino. Caso o *flit* esteja autorizado a avançar, mas o *buffer* destino esteja cheio, este *flit* permanece na mesma posição, e aguarda o próximo ciclo.

Além da verificação de disponibilidade da porta, existe outra restrição no avanço dos *flits*, que é a análise do ciclo em que o *flit* chegou no roteador. Se o ciclo em que o *flit* chegou no roteador é o mesmo ciclo atual de simulação, então não pode ocorrer esta transmissão, fazendo com que o *flit* aguarde. Isso se deve ao fato de que deve existir a atomicidade da execução, pois os roteadores são processados de forma sequencial. Por exemplo, durante um ciclo o roteador r1 envia um *flit* para o roteador r2. Quando acontecer o processamento de r2, o roteador irá perceber que ciclo que o *flit* foi recebido é o mesmo ciclo atual de simulação e não irá permitir que o *flit* avance para um próximo roteador. Se o *flit* avançasse para um roteador r3, iria existir o avanço do *flit* por dois roteadores em um mesmo ciclo, o que causaria uma inconsistência. Portanto, na execução normal, r1 envia o *flit* para r2; ao processar r2, o roteador percebe que não

deve processar esse *flit* neste ciclo, e o *flit* só avança para r3 no próximo ciclo. Dessa forma, temos que um *flit* pode avançar até, no máximo, uma posição/roteador em direção ao seu destino final.

A Figura 22 apresenta a implementação da função responsável por executar o algoritmo de roteamento e inserir a informação gerada pelo algoritmo na tabela de roteamento. É possível ver a implementação do algoritmo de roteamento XY (CARARA, 2004). Através do parâmetro *routing-algorithm*, escolhido na interface, a função escolhe qual algoritmo de roteamento deve ser executado. Com base nos dados da posição *x* e *y* do roteador destino, o algoritmo determina por qual porta o pacote irá sair. A porta de destino é escolhida como um número de 0 a 4, que corresponde, respectivamente, às portas Norte, Leste, Sul, Oeste e PE. No fim da execução do algoritmo de roteamento, é inserido o dado gerado pelo algoritmo de roteamento na tabela de roteamento. Esse dado contém o identificador do pacote, porta de destino e custo do roteamento

Figura 22: Código do roteamento

```
1752 to update-routing-table [ID X-destination Y-destination]
1753   ;Função recebe ID do pacote, X e Y do roteador destino
1754   let destination-port ""
1755
1756   ifelse (routing-algorithm = "XY")
1757   [
1758     ifelse (X-destination = xcor) ;X do destino = X atual?
1759     [
1760       ifelse (Y-destination = ycor) ;Y do destino = Y atual?
1761       [
1762         set destination-port 4 ;;X e Y iguais, é o destino, Envia para o PE
1763       ]
1764       [
1765         ifelse (Y-destination > ycor) ;;Mesmo X, Y diferente, então testa se Y-destino é maior que Y-atual?
1766         [
1767           set destination-port 0 ;;Envia para o Norte
1768         ]
1769         [
1770           set destination-port 2 ;;Y-destino é menor que Y-atual, então envia para baixo (S)
1771         ]
1772       ]
1773     ]
1774     [
1775       ifelse (X-destination > xcor) ;;X diferente, então, X destino > X atual?
1776       [
1777         set destination-port 1 ;;Envia para Leste
1778       ]
1779       [
1780         set destination-port 3 ;;X destino < X atual, então envia para Oeste
1781       ]
1782     ]
1783   ]
1784   [
1785     ;;Para implementar novos algoritmos de roteamento
1786   ]
1787   ;No fim da execução do algoritmo de roteamento, a função deve inserir a tupla na tabela de roteamento,
1788   ;contendo: ID do pacote, porta de destino e custo do roteamento
1789   set routing-table lput (list ID destination-port routing-cost) routing-table
1790 end
```

Fonte: do autor

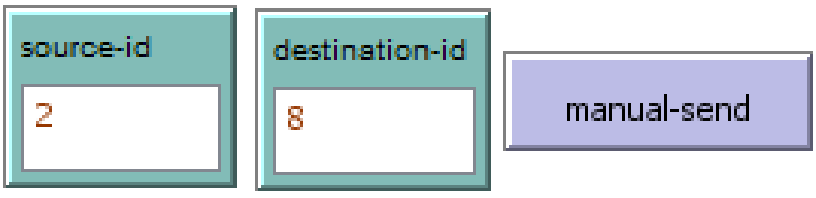
O simulador é extensível e pode ser alterado para implementar e testar novos algoritmos de roteamento. Para tal, apenas uma função precisa ser alterada, que é a função que atualiza a tabela de roteamento (*update-routing-table*), apresentada na Figura 22. Nessa função, existe um conjunto de *ifelse* que verificam qual algoritmo de roteamento foi escolhido na interface, realiza o cálculo e escreve na tabela. Dessa forma, para incluir um novo algoritmo, basta incluir o nome no seletor da interface e adicionar mais uma condição *ifelse* contendo a lógica do algoritmo de roteamento e este algoritmo já estará disponível para simulação.

5.3.5 Gerador de Carga

Para a realização das simulações, o simulador conta com opções de tráfego. Existem opções tanto para tráfego manual quanto para aleatório, ou seja, uma opção para caso o usuário saiba qual será o tráfego da rede, e outra para que seja gerado um tráfego aleatório de forma a testar a capacidade da rede e dos roteadores.

Para o tráfego manual, o usuário pode escolher qual roteador será origem e qual será destino através da interface, e ao ativar o botão de envio manual, será gerado um pacote no roteador escolhido como origem, que será enviado para o roteador destino. Para ilustrar esse funcionamento, a Figura 23 apresenta as entradas de identificação dos roteadores de origem e destino, e o botão de disparo. Ao ativar o botão “manual-send”, será gerado um pacote no roteador 2 que tem como destino o roteador 8. Estes roteadores são representados na malha apresentada na Figura 23.

Figura 23: Envio manual de pacotes

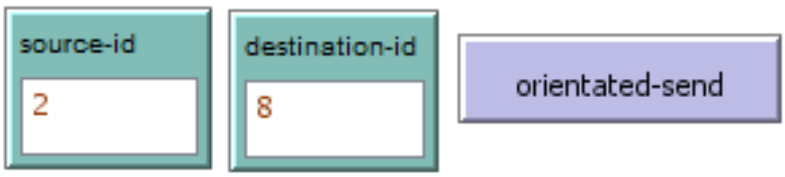


Fonte: do autor

Já para a geração de tráfego aleatório, o simulador conta com duas opções: tráfego aleatório direcionado ou tráfego aleatório pelo raio. No tráfego aleatório

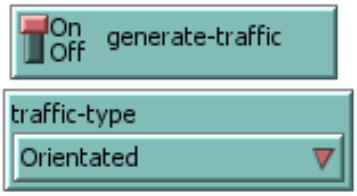
direcionado, o usuário pode indicar um ou mais pares de origem e destino de pacotes e cada vez que o simulador for gerar um pacote, irá escolher um destes pares aleatoriamente. Para indicar os pares, o usuário utiliza os mesmos campos de origem e destino do tráfego manual, porém ao invés de ativar o botão “manual-send”, o usuário deve ativar o botão “orientated-send”. O usuário pode indicar quantos pares de origem e destino quiser. Depois de indicar os pares, o usuário deve ativar a opção de gerar tráfego aleatório, através do selecionador “generate-traffic” e escolher o tipo de tráfego direcionado no selecionador do tipo de tráfego, “traffic-type”. A Figura 24 e Figura 25 mostram, respectivamente, a parte da interface onde os pares são escolhidos e a escolha deste tipo de tráfego. Estes roteadores são representados na malha apresentada na Figura 18.

Figura 24: Tráfego direcionado – Escolhendo um par



Fonte: do autor

Figura 25: Tráfego direcionado – Ativando o tipo de tráfego

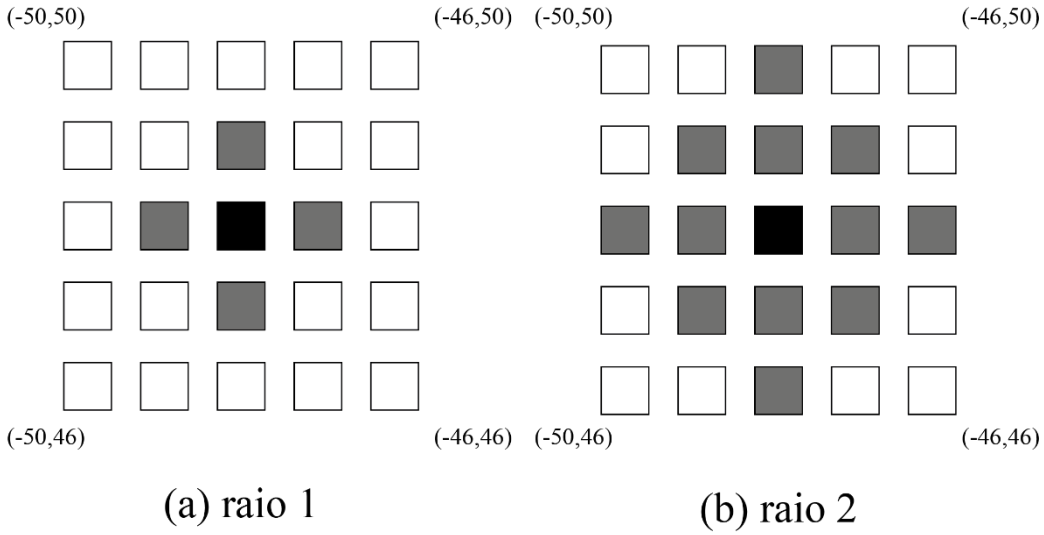


Fonte: do autor

Já no tráfego aleatório pelo raio, a cada geração de pacote o simulador escolhe um roteador que será a origem do pacote. O destinatário deste pacote será um roteador que está dentro de um círculo determinado pelo raio de mensagens. Para utilizar o tráfego aleatório pelo raio, o usuário precisa ativar a geração de tráfego aleatória, através do selecionador “generate-traffic”, escolher o tipo de tráfego “Random-By-

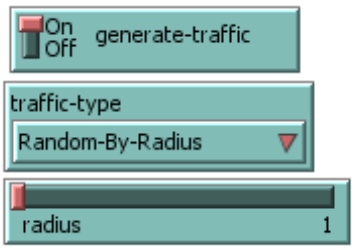
Radius” no selecionador do tipo de tráfego, e indicar o raio de escolha dos roteadores no selecionador “radius”. A Figura 26 ilustra uma representação de quais roteadores podem ser destinatários para uma mensagem que será enviada de um roteador localizado no centro da rede. Os possíveis destinatários para uma mensagem que parte de um roteador origem, representado pela cor preta, estão marcados em cinza, para os raios 1 e 2. É possível observar que, ao utilizar o raio 1, os destinatários das mensagens estarão a uma posição de distância do roteador origem. Já ao utilizar o raio 2, os destinatários podem estar a até duas posições de distância. Por conta de a malha representada ser 5x5, quando um roteador central é escolhido para gerar mensagens, o destinatário pode estar praticamente em qualquer lugar da rede, com exceção dos roteadores que estão perto dos cantos. A Por fim, a Figura 27 ilustra como é a ativação deste tipo de tráfego e como funciona a escolha do raio.

Figura 26: Tráfego aleatório – Representação raio 1 e 2



Fonte: do autor

Figura 27: Tráfego aleatório – Escolhendo o raio e ativando o tipo de tráfego



Fonte: do autor

Quando um pacote é gerado em um roteador, este pacote não é armazenado diretamente pelo *buffer* do PE, mas sim em um *buffer* temporário. Esse *buffer* é chamado de *temporary-buffer* e tem como função simular a aplicação que está sendo executada nessa NoC e gera pacotes que devem ser trocados entre os nodos. Esse *buffer* tem seu tamanho definido pela interface, através do parâmetro *temporary-buffer-size*. Cada vez que um pacote é gerado, o gerador de carga verifica se existe espaço no *buffer* temporário para armazenar todos os *flits*. Se existir espaço, todos os *flits* são armazenados no *buffer*. Porém, caso não exista espaço, o pacote é descartado. Este comportamento representa aplicações que tentam enviar pacotes, mas se deparam com o *buffer* do nodo cheio, necessitando, portanto, descartar os *flits* e tentar retransmití-los posteriormente. Esse descarte de mensagens do *buffer* temporário só ocorre quando a taxa de geração de pacotes é superior à taxa de serviço dos roteadores. Os roteadores transferem, a cada ciclo, todos os *flits* que estão no *buffer* temporário para o *buffer* de entrada do PE, conforme a sua capacidade.

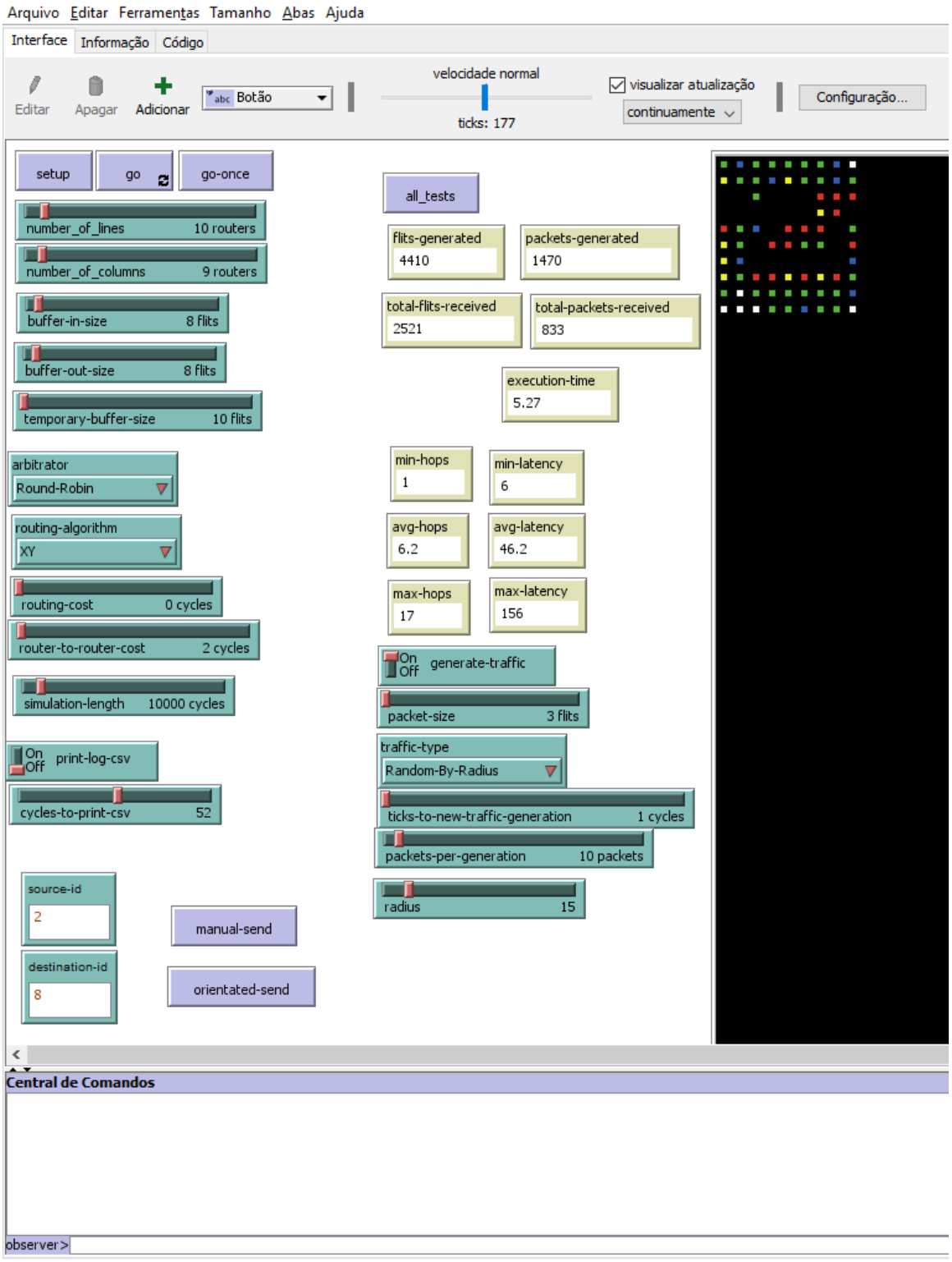
Por fim, embora esses três tipos de tráfegos tenham sido implementados, o simulador está preparado para receber outros padrões. Para gerar um novo padrão, primeiramente o usuário deve incluir o novo tipo de tráfego no selecionador referente ao tipo de tráfego (*traffic-type*). Depois disso, a função de geração de pacotes (generate-new-packets) contém uma série de *ifelse*, sendo um para cada tipo de tráfego que contém no selecionador. Portanto, basta incluir um novo *ifelse* na função e implementar a lógica da nova geração de tráfego e este padrão já estará disponível para simulação.

5.3.6 Informações gerais sobre o Código

O desenvolvimento do simulador no NetLogo gerou um código com aproximadamente 4.000 linhas, sendo 2.000 para depuração e testes da ferramenta. O código que implementa a interface do simulador não influencia nesta contabilização do número de linhas. O NetLogo utiliza uma programação procedural, portanto foram geradas 20 funções para desempenhar as principais atividades do simulador.

Além das informações sobre o código, também é importante observar a interface da ferramenta. Para ilustrar esta interface, a Figura 28 apresenta uma imagem da interface durante uma simulação. Na coluna da direita, são mostrados os botões para inicialização da NoC e execução da simulação, além dos parâmetros de configuração da rede. Além disso, nesta coluna também é possível acessar as configurações de impressão dos *logs*, do tamanho da simulação, e de alguns tipos de envio de pacotes. A coluna do meio apresenta algumas das métricas, como número de saltos e latência, além de apresentar as configurações de geração de tráfego. À direita é possível ver o *grid* com os roteadores da simulação, coloridos conforme a porta mais utilizada no momento. Mais detalhes sobre as métricas são apresentados na Seção 5.5.

Figura 28: Interface do simulador



Fonte: do autor

5.4 Parâmetros de Entrada

O simulador conta com diversos parâmetros que o usuário pode alterar para simular uma NoC. Dentre eles, é possível destacar:

- **Dimensões da malha:** dois parâmetros definem o tamanho da NoC a ser simulada: o número de linhas e de colunas. Com base nos valores *L e C* escolhidos, será gerada uma NoC contendo *L* linhas compostas de *C* roteadores cada. Parâmetro medido em número de roteadores.
- **Tamanho do *buffer* de entrada e de saída*:*** estes dois parâmetros definem quantos *flits* cada porta é capaz de armazenar nos *buffers* de entrada e de saída em cada roteador. Parâmetro medido em *flits*.
- **Tamanho do *buffer* temporário:** este parâmetro define quantos *flits* o *buffer* temporário, relacionado à geração de carga, consegue armazenar. Parâmetro medido em *flits*.
- **Árbitro:** escolhe qual tipo de árbitro será utilizado na simulação. O árbitro terá a função de estabelecer a ordem de acesso às portas dos roteadores conforme a prioridade de execução em cada ciclo. Na versão atual do simulador o árbitro implementa uma estratégia *Round Robin*. Vale ressaltar que o simulador está preparado para a implementação e simulação de outros tipos de árbitros.
- **Algoritmo de roteamento:** opção responsável por determinar qual algoritmo de roteamento será utilizado na simulação. Na versão atual do simulador o tem disponível o algoritmo de roteamento XY. Vale ressaltar que o simulador está preparado para a implementação e simulação de outros tipos de algoritmos de roteamento.
- **Custo de roteamento:** determina quantos ciclos levam para que o algoritmo de roteamento seja calculado. Este parâmetro é importante para incluir o custo do cálculo de algoritmos de roteamento mais complexos. Além disso, esse parâmetro permite analisar casos onde algumas ações levam mais de um ciclo para acontecer. Ao escolher o valor 0 para este atributo, o algoritmo de roteamento é calculado instantaneamente, fazendo com que o custo do

cálculo do algoritmo de roteamento seja desconsiderado na simulação. Este parâmetro permite a configuração de cenários mais realistas, onde ações como o cálculo do roteamento podem levar mais de um ciclo para serem executadas. Por exemplo, na NoC HERMES, apresentada em (MORAES, 2004), o algoritmo de roteamento leva pelo menos 10 ciclos para ser executado. Parâmetro medido em ciclos.

- **Custo de transmissão:** de maneira similar ao custo de roteamento, este parâmetro ajuda na análise de casos com operações mais custosas. O valor escolhido neste parâmetro é referente a quantidade de ciclos que leva para um *flit* ser retirado do *buffer* de saída da porta de um roteador e ser inserido no *buffer* de entrada do roteador destino. Parâmetro medido em ciclos.
- **Tamanho dos pacotes:** determina a quantidade de *flits* de cada pacote. Parâmetro medido em *flits*.
- **Gerar tráfego:** ao ativar essa opção, o simulador irá gerar um tráfego sintético para a simulação.
- **Tipo de tráfego:** utilizando tráfego sintético, esta opção determina qual será o critério para escolher o roteador destino de um pacote, partindo de um roteador origem. Os padrões de carga implementados são: manual, onde o usuário indica um roteador origem e outro destino para o envio de pacotes; direcionado, onde o usuário indica vários pares de roteadores para serem origem e destino e o simulador escolhe aleatoriamente estes pares; aleatório pelo raio, onde, baseado em um roteador origem, o simulador irá escolher um roteador destino dentro de um círculo determinado pelo raio, que é um parâmetro configurável.
- **Raio de mensagens:** este é um parâmetro exclusivo para o tipo de tráfego chamado de “Aleatório pelo raio”. Neste tráfego, quando um roteador é escolhido para ser a origem de um pacote, o roteador destino será um roteador que está dentro de um círculo determinado pelo *raio de mensagens*. O tamanho do raio é o valor escolhido nesse parâmetro.

- **Ciclos para a geração:** durante a geração de tráfego sintético, este parâmetro determina de quantos em quantos ciclos serão gerados novos pacotes. Parâmetro medido em ciclos.
- **Pacotes por geração:** também relacionado ao tráfego sintético, quando acontecer um ciclo de geração de pacotes, a quantidade de pacotes que serão gerados é escolhida através deste atributo. Parâmetro medido em pacotes.
- **Tamanho da simulação:** quando a simulação atingir uma quantidade de ciclos de simulação igual ao valor escolhido neste parâmetro, a simulação é encerrada. Parâmetro medido em ciclos.
- **Imprimir *log* CSV:** ao ativar esta opção, o simulador irá produzir um arquivo CSV contendo um *log* do estado dos *buffers* de todos os roteadores ao longo da execução.
- **Ciclos para imprimir CSV:** este parâmetro determina de quantos em quantos ciclos serão armazenadas informações no arquivo CSV. Por exemplo, ao escolher o valor 1, serão armazenados os estados de todos os roteadores para todos os ciclos da simulação. Ao utilizar valores maiores, o custo para armazenar as informações é reduzido e o tamanho do arquivo gerado será menor. Porém, a granularidade de observação é mais grossa, podendo omitir detalhes da execução. Parâmetro medido em ciclos.

5.5 Métricas de Simulação

O simulador conta com métricas que auxiliam o usuário a observar comportamentos na simulação. Existem métricas que podem ser observadas tanto durante quanto ao término da simulação. As definições e características dos indicadores implementados são:

- **Contabilização de pacotes e *flits*:** é possível verificar a quantidade de pacotes e *flits* que foram criados e recebidos. A diferença entre os pacotes/*flits* que foram criados e recebidos determina quantos ainda estão trafegando pela rede, ou seja, ainda não chegaram no destino.

- **Tempo de execução**: Tempo em segundos, desde o início da simulação até o recebimento da última mensagem.
- **Contagem de saltos**: são apresentados os valores mínimos, médios e máximos para o número de saltos determinados na rota dos pacotes. Um salto é contabilizado quando um *flit* sai de um roteador e avança para outro.
- **Contagem de atraso**: Este parâmetro mede os números mínimo, médio e máximo de ciclos desde a inserção do *flit* na NoC até chegar no destino. Existem diversos fatores que podem fazer com que aconteça um atraso, como o tempo para cálculo do algoritmo de roteamento, tempo de transmissão, a porta destino pode já estar em uso e o *buffer* destino pode estar cheio.
- **Coloração dos roteadores:** durante cada ciclo, os roteadores recebem a cor referente à sua taxa de ocupação.

A coloração dos roteadores é uma etapa realizada após processar os pacotes, mediante um cálculo realizado para determinar qual das portas do roteador está mais ocupada, com exceção das portas do PE. A cada ciclo, o roteador busca em todas suas portas, incluindo de entrada e de saída, qual está mais ocupada, calcula qual é o percentual de ocupação dessa porta em relação à sua capacidade total da porta e o roteador recebe uma coloração conforme a porcentagem de utilização da porta mais ocupada no momento. O simulador oferece uma saída visual da utilização do roteador, colorindo de acordo com a classificação apresentada na Tabela **5**.

Tabela 5: Representação de cor conforme ocupação do *buffer* da porta ocupada mais ocupada

Cor	Ocupação
Branco	0% da capacidade do *buffer* da porta ocupada
Azul	1~24% da capacidade do *buffer* da porta ocupada
Verde	25~49% da capacidade do *buffer* da porta ocupada
Amarelo	50~74% da capacidade do *buffer* da porta ocupada
Vermelho	75~99% da capacidade do *buffer* da porta ocupada
Preto	100% da capacidade do *buffer* da porta ocupada

Fonte: do autor

Por fim, o simulador permite ao usuário utilizar uma opção para gerar um *log* contendo o estado de todos os *buffers*, de todos os roteadores, ao longo de toda a execução. A Tabela 6 ilustra a estrutura do *log* gerado. A primeira linha contém o cabeçalho da tabela, indicando o número do ciclo e todas as portas de entrada e de saída de todos os roteadores utilizados na simulação. A partir da segunda linha, são apresentadas separadamente a quantidade de *flits* que estavam nos *buffers* de entrada e de saída de todos os roteadores, a cada ciclo de execução da simulação. Por exemplo, a linha 2 indica que no ciclo 1 o *buffer* de entrada da porta Norte do roteador 0 tinha 1 *flit*, o *buffer* de entrada da porta Leste do roteador 0 tinha 2 *flits,* o *buffer* de saída da porta Norte do roteador 0 estava vazio, o *buffer* de saída da porta Leste do roteador 0 tinha 2 *flits* e o *buffer* de entrada da porta Norte do roteador 1 tinha 1 *flit*.

Tabela 6: Exemplo simplificado de *log* de simulação

Ciclo	Roteador 0 Entrada Norte	Roteador 0 Entrada Leste	...	Roteador 0 Saída Norte	Roteador 0 Saída Leste	...	Roteador 1 Entrada Norte	...
1	1	2	...	0	2	...	1	...
2	0	1	...	3	2	...	0	...
...	...	...	...	...	...	...	...	...

Fonte: do autor

Essa métrica é muito importante para quem deseja observar comportamentos específicos em qualquer parte da simulação. Além disso, ter todos os dados brutos da simulação permite ao usuário manipular estas informações para diversos tipos de análises, como gerar gráficos de análise da utilização de uma ou mais portas de entrada, de saída, da utilização do roteador como um todo.

6 AVALIAÇÃO DO SIMULADOR

Esse capítulo tem como objetivo apresentar a avaliação do simulador, feita através da criação de cenários de teste e da análise dos resultados obtidos nestas simulações. Foram criados casos de teste com o intuito de verificar o funcionamento do simulador desenvolvido. Além disso, os testes ajudaram a avaliar a expressividade do simulador em termos do que pode ser modelado e a capacidade de análise através das métricas e indicadores obtidos na simulação.

Os testes foram divididos em três grupos. O primeiro grupo tem por objetivo verificar o impacto na latência que os *flits* apresentam até chegar no seu destino, utilizando diferentes volumes de geração de carga e rotas com e sem interseções. Já o segundo grupo tem por objetivo explorar os dados da simulação que podem ser obtidos através do *log*, apresentando alguns dos resultados que podem ser extraídos com base nos dados brutos disponíveis. Por fim, o terceiro grupo mostra o impacto do uso da interface e da geração do *log* com frequências de impressão diferentes no tempo de simulação.

Para todos os testes, os cenários utilizam alguns parâmetros em comum. A configuração padrão é definida conforme os valores apresentados na Tabela 7.

Tabela 7: Parâmetros da simulação

Tamanho da Rede	3x3
Buffer de Entrada/Saída	8 *Flits*
Buffer Temporário	10 *Flits*
Custo de Roteamento/Transmissão	0 Ciclos
Tamanho da Simulação	10.000 Ciclos (1.000 *warmup*)
Árbitro	Round-Robin
Algoritmo de Roteamento	XY

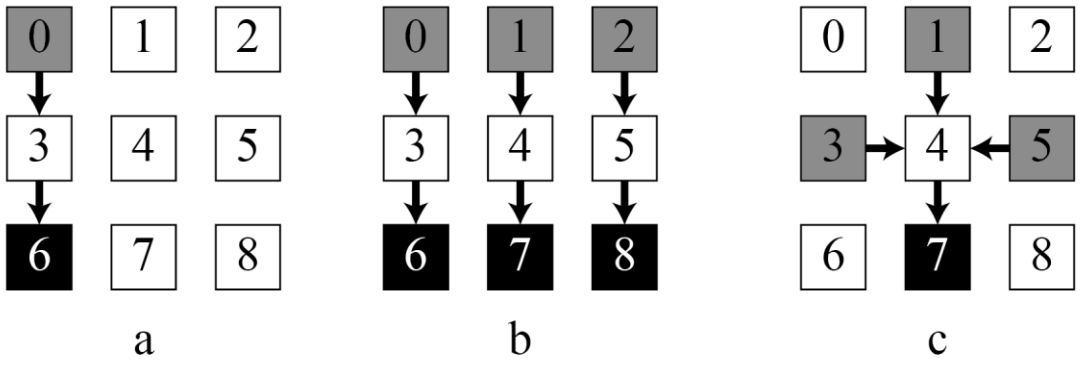
Fonte: do autor

Nos experimentos realizados, os primeiros 1.000 ciclos foram desconsiderados para efeito de análise.

6.1 Grupo 1: Avaliação da latência e contenção na rede

O primeiro conjunto de testes visa avaliar os efeitos em NoCs simples ao utilizar diferentes padrões de carga, utilizando rotas com e sem congestionamento e cargas de diferentes intensidades. Cada um dos tipos de rotas tem por objetivo avaliar a latência em cenários específicos. A rota sem congestionamento mostra os valores de latência para cenários afetados apenas pelos atrasos da própria geração de carga. Já ao simular rotas com congestionamento, além da latência ser afetada pela geração de carga, ela também será impactada pelos caminhos congestionados que farão com que alguns pacotes precisem aguardar a liberação de portas. Da mesma maneira que as rotas, as cargas de diferentes intensidades também visam mostrar como a sobrecarga no uso dos roteadores tem impacto na latência, visto que, conforme a intensidade da carga aumenta, também aumenta a chance de que certos pacotes precisem aguardar mais pela liberação de portas. Com base na configuração padrão apresentada na Tabela 7, foram derivados três cenários diferentes, variando a quantidade de pares de origem/destino e a intensidade da carga. Para uma melhor compreensão sobre os cenários criados, a Figura 29 ilustra a rede onde os Cenários são testados, com a identificação dos roteadores. Os três cenários são:

Figura 29: Cenários com tráfego direcionado: um par (a), três pares sem interseção (b) e três pares com interseção (c)



Fonte: do autor

- Cenário 1 – Tráfego direcionado, utilizando apenas 1 par de origem/destino (par 0-6). Ilustração na Figura 29 (a);

- Cenário 2 – Tráfego direcionado, utilizando 3 pares de origem/destino sem interseções (pares 0-6, 1-7 e 2-8). Figura 29 (b);

- Cenário 3 – Tráfego direcionado, utilizando 3 pares de origem/destino com interseções (pares 1-7, 3-7 e 5-7). Figura 29 (c).

O Cenário 1 representa um cenário com apenas uma rota. O Cenário 2 apresenta uma simulação onde existem 3 rotas para os pacotes, porém estas rotas não têm interseção. Por fim, o Cenário 3 também apresenta um caso com 3 rotas, mas as três rotas possuem uma interseção.

Para cada um dos cenários, foram testadas três intensidades de carga. Os padrões de carga buscam simular cenários onde a rede está tendo uma utilização baixa, moderada ou intensa. As informações sobre as cargas são apresentadas na Tabela 8.

Tabela 8: Informações sobre as cargas

Carga	Quantidade de *flits* por pacote	Quantidade de ciclos para geração de um pacote
A	3	3
B	3	2
C	3	1

Fonte: do autor

Para cada simulação foram analisados os indicadores de saída mostrados no painel do simulador. Os dados analisados foram os valores mínimos, médios e máximos do número de saltos e da latência dos flits, além da quantidade total de flits que chegaram ao destino. A Tabela 9 apresenta os resultados da simulação dos cenários 1, 2 e 3, utilizando as cargas de trabalho A, B e C.

Tabela 9: Resultados dos valores de Saltos e Latência

	Saltos			Latência			*Flits* entregues	*Flits* gerados	*Flits* não gerados
Cenário	Min	Méd	Max	Min	Méd	Max			
Cenário 1, Carga A	2	2	2	3	4	5	9995	9999	0
Cenário 2, Carga A	2	2	2	3	4	5	9995	9999	0
Cenário 3, Carga A	2	2	2	3	4	5	9995	9999	0
Cenário 1, Carga B	2	2	2	3	9	9	9996	14999	4986
Cenário 2, Carga B	2	2	2	3	5	9	14992	14999	0
Cenário 3, Carga B	2	2	2	3	49	88	9996	14999	4935
Cenário 1, Carga C	2	2	2	3	9	9	9996	29999	19986
Cenário 2, Carga C	2	2	2	3	8	9	28185	29999	1785
Cenário 3, Carga C	2	2	2	3	49,9	88	9996	29999	19929

Fonte: do autor

É possível observar que o número mínimo, médio e máximo de saltos se mantém o mesmo em todos os casos. Isso acontece devido ao comportamento determinístico do algoritmo de roteamento XY, que utiliza sempre o mesmo caminho para um determinado par de nodos origem e destino. Além disso, nos cenários 1, 2 e 3 a distância entre origem e destino de todos os pacotes é a mesma, fazendo com que pacotes passem sempre por 2 nodos da rede, a partir do nodo de origem. No domínio da latência, os valores mínimos se mantêm os mesmos para todos os casos. Este valor corresponde à chegada do primeiro *flit* do primeiro pacote, que percorre todo o caminho livre, gerando o menor valor de latência possível, 2 ciclos de transmissão entre roteadores mais um ciclo para o roteador final enviar o *flit* para o PE. Os valores de latência média e máxima obtidos utilizando a carga de 1 pacote a cada ciclo são os menores possíveis, em todos os cenários. Estes valores são 3 ciclos de latência mínima, 4 de média (*flits* do tipo *body*, que são gerados junto com os *headers*, mas só podem trafegar depois que o *header* passa pelos roteadores e garante o acesso as portas de saída) e 5 de máxima (*flits* do tipo *tail*, que só passam depois dos *headers* e *bodies* passarem). Vale ressaltar que, devido à carga A, de 1 pacote a cada 3 ciclos, ser uma carga de baixa intensidade, não é gerada saturação na geração de pacotes. Dessa forma, nenhum pacote deixa de ser gerado por conta de o *buffer* temporário estar cheio.

Utilizando as cargas B e C, o Cenário 1 apresenta um descarte de pacotes na geração por conta da utilização dos roteadores, sendo aproximadamente 33% de descarte na carga B e 66% na carga C. Como existe apenas um par de origem/destino, todos os pacotes são gerados no mesmo roteador, fazendo com que aconteça essa sobrecarga da própria geração de carga. Dessa forma, o Cenário 1 apresenta latências

intermediárias, mas com quantidades de descartes alto. Já o Cenário 2, além de apresentar valores de latência similares ao Cenário 1, tem valores de descartes bem menores do que o Cenário 1. Isso acontece porque o Cenário 2 possui 3 pares de origem/destino, sem contenção. Como a geração de carga é distribuída aleatoriamente entre esses 3 pares, a contenção que é criada pela geração dos pacotes é menor. Como existe menos contenção, o *buffer* temporário dos roteadores fica ocupado por menos tempo e, por consequência, mais pacotes podem ser gerados. Por fim, o Cenário 3 apresenta os maiores valores de latência e uma quantidade de descarte de pacotes similar ao Cenário 1. Embora esse cenário também tenha 3 pares de origem/destino, as rotas destes pares contêm intersecções, ou seja, além da contenção que existe na própria geração de carga, também existe a contenção gerada pelos pacotes que precisam acessar as portas dos roteadores intermediários que já estão em uso. O congestionamento gerado pela intersecção faz com que *flits* precisem aguardar a liberação de portas. Por consequência, os *flits* levam mais ciclos para chegar ao destino, fazendo com que a latência seja maior. Da mesma maneira que a latência, a contenção gerada pela intersecção também faz com que os *flits* que aguardam a liberação de portas ocupem os *buffers* por mais tempo. Portanto, em determinados casos os *flits* inseridos na rede pela geração de carga não conseguem sair do *buffer* temporário, fazendo com que ele fique ocupado por mais tempo, o que leva ao descarte de pacotes na geração de carga.

As colunas de contabilização de *flits* apresentam informações importantes, que permitem observar se o padrão da geração de carga está saturando a NoC. A coluna dos *flits* que deveriam ser gerados trata de uma estimava. Como é de conhecimento o tamanho da simulação e qual é o padrão da geração de carga, é possível estimar quantos pacotes deveriam ser gerados e, por consequência, os *flits* gerados também, visto que todos os pacotes têm o mesmo tamanho. Por exemplo, no primeiro padrão de carga, é gerado 1 pacote a cada 3 ciclos. Como a simulação dura 10.000 ciclos, tem-se a geração de 3333 pacotes. Da mesma maneira, cada pacote contém 3 *flits*, logo, 9999 *flits* deveriam ser gerados. Pode-se observar na Tabela 9 que o número de *flits* entregues não coincide com o número de *flits* gerados, mesmo nos casos onde o número de *flits* não gerados foi 0. As pequenas diferenças entre eles, dois pacotes, é causada pela interrupção da simulação ao se atingir 10.000 ciclos. Como são gerados pacotes a cada ciclos, os últimos pacotes não tiveram tempo de chegar no destino antes da simulação acabar. Como observado na coluna "*Flits* não gerados", principalmente nos cenários 1 e

3 com as cargas B e C, a geração de carga nestes casos foi muito elevada, indicando que a taxa de geração de pacotes é maior do que a capacidade de processamento da NoC. Este indicador torna-se importante para identificar pontos de saturação na simulação.

6.2 Grupo 2: Avaliação das taxas de ocupação e saturação

Para observar como a utilização dos *buffers* varia em função do tamanho da rede, distância entre nodos origem e destino, e intensidade da carga foram montados cenários utilizando NoCs de tamanho 5x5, 10x10 e 15x15. Foram utilizados três raios de mensagens, com o objetivo de criar cargas que simulam mensagens trocadas apenas entre vizinhos (raio pequeno), mensagens trocadas entre vizinhos ou até roteadores de distância intermediária (raio médio) e mensagens que podem ser trocadas por qualquer par de roteadores na rede (raio grande). Para cada tamanho e cada raio, ainda são utilizadas cargas de duas intensidades, proporcionais ao tamanho da rede. As quantidades de pacotes criados a cada ciclo de geração foram iguais a 1/10 do tamanho da rede (carga baixa) e 1/5 da rede (carga alta). A configuração padrão utilizada foi similar a apresentada na Tabela 7, com exceção do tamanho da rede, onde foram testados 3 tamanhos de rede, e o tipo de tráfego, sendo utilizado o tráfego aleatório pelo raio, com geração de carga a cada 1 ciclo.

Seguindo esse padrão, ao utilizar uma rede com 25 roteadores (5x5) por exemplo, são gerados 5 pacotes por ciclo (com carga baixa) e 17 pacotes por ciclo (com carga alta). As configurações dos cenários, como o tamanho da rede, raios e cargas são detalhados na Tabela 10.

Tabela 10: Casos de teste - Grupo 2

	Tamanho do raio			Pacotes gerados por ciclo	
Cenário	Pequeno	Médio	Grande	Carga Baixa	Carga Alta
5x5	1	4	8	3	5
10x10	1	8	16	10	20
15x15	1	11	22	23	45

Fonte: do autor

Para cada cenário, foram feitas 5 iterações, onde os resultados se mantiveram os mesmos. Estes testes foram realizados para observar a utilização dos *buffers* de entrada através da taxa de ocupação e taxa de saturação. A taxa de ocupação mede qual foi a utilização dos *buffers* para cada roteador, em média, ao longo da execução. Já a taxa de saturação mostra como foram os casos de maior utilização de *buffers*, em média, ao longo da execução. É importante ressaltar que os dados utilizados para gerar os mapas de calor foram extraídos dos *logs* gerados pelo simulador.

A taxa de ocupação é calculada através da fórmula:

$$Taxa\ de\ Ocupação\ = \frac{\sum flits\ nas\ portas\ de\ entrada\ N, S, L\ e\ O}{Ciclos\ de\ simulação\ x\ Tamanho\ do\ buffer\ de\ entrada\ x\ 4}$$

É importante ressaltar que as análises das taxas de ocupação desconsideram as informações do *log* relacionadas *buffer* de entrada do PE, pois o interesse de avaliação é no roteamento. Por esse motivo, o tamanho das portas é multiplicado por 4 (referente às portas N, S, L, O) no denominador da taxa de ocupação, e não por 5.

Já a taxa de saturação é calculada pela equação:

$$Taxa\ de\ Saturação = \frac{\sum Maior\ quantidade\ de\ flits\ nas\ portas\ de\ entrada}{Tamanho\ do\ buffer\ de\ entrada\ x\ Ciclos\ de\ simulação}$$

Para um melhor compreendimento sobre a fórmula, a Tabela 11 apresenta um exemplo hipotético de informações sobre os dados da quantidade de *flits* nos *buffers* das portas de entrada de um roteador R ao longo de uma simulação de 5 ciclos. O exemplo adota *buffers* com capacidade para 8 *flits*. Segundo os dados da tabela e a fórmula apresentada, a taxa de ocupação para este roteador apresenta o valor 0,13125.

Tabela 11: Exemplo para análise das taxas de ocupação e saturação de um roteador

	Roteador R			
Ciclo	Porta N	Porta L	Porta S	Porta O
1	0	0	2	2
2	0	1	3	2
3	1	0	2	1
4	2	1	1	0
5	3	0	0	0

Fonte: do autor

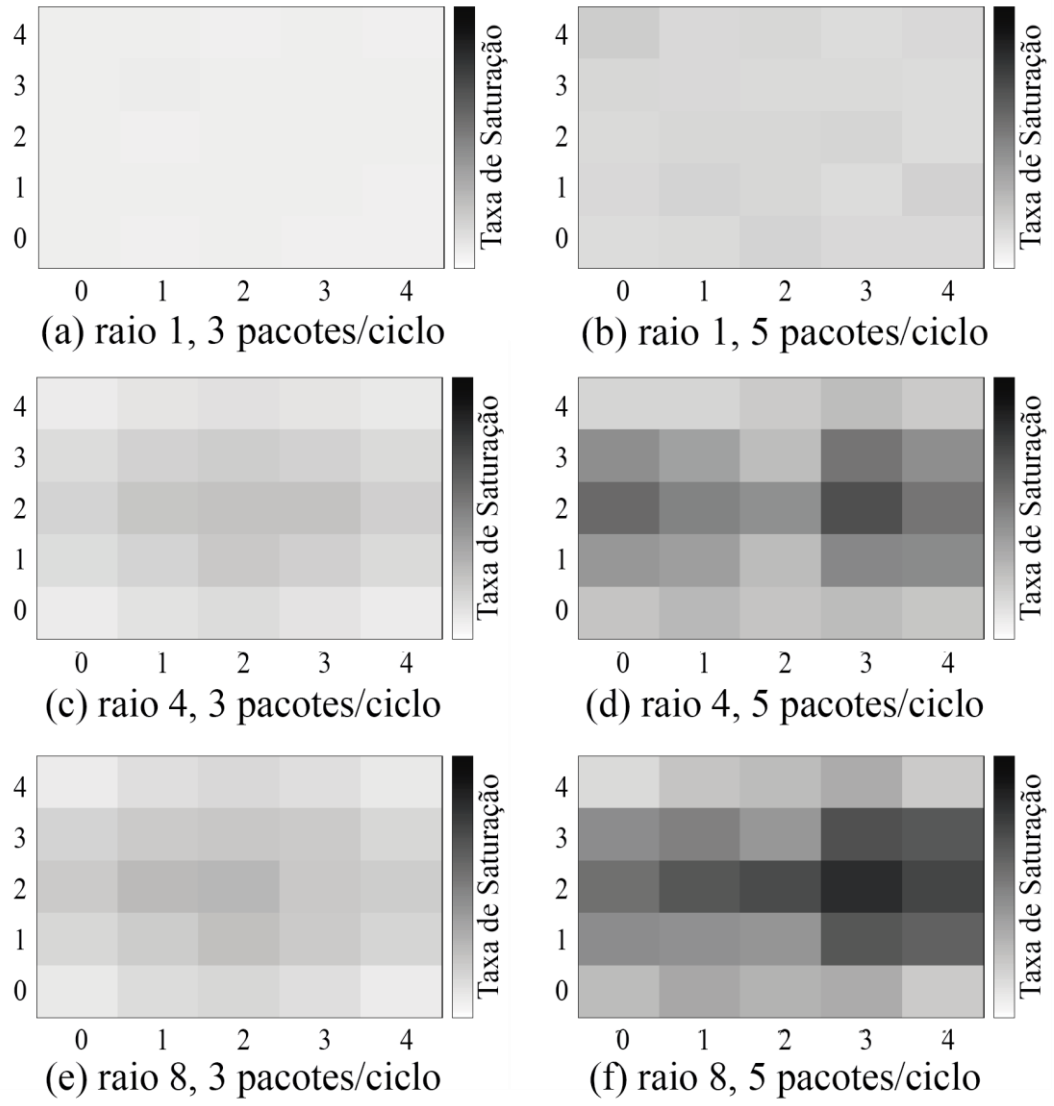
Utilizando o mesmo exemplo da Tabela 11, aplicando a fórmula da taxa de saturação, a taxa de saturação para este roteador é 0,3.

Para ilustrar a importância da taxa de ocupação e da taxa de saturação, foram gerados gráficos de calor (*heatmaps*). Estes mapas são utilizados para mostrar a intensidade da ocorrência de alguma situação. Os mapas de calor das taxas de saturação e das taxas de ocupação dos *buffers* de entrada de cada um dos cenários são mostrados da Figura 30 até a Figura 38. Os dados utilizados para a elaboração dos mapas de calor foram extraídos dos *logs* gerados pelas simulações. Por se tratarem de mapas de calor, as figuras mostram a intensidade de um comportamento através de uma coloração. Os valores exibidos apresentam as taxas de ocupação e saturação média de cada porta ao longo da execução. Nos eixos x e y são apresentados, respectivamente, as coordenadas X e Y de cada roteador da simulação. Dessa forma, cada roteador da NoC corresponde a um retângulo do mapa de calor. A escala é correspondente ao tipo de mapa, ou seja, os retângulos mais escuros indicam roteadores que tiveram uma ocupação ou saturação maior, enquanto os retângulos mais claros mostram roteadores que sofreram menos saturação ou ocupação.

A Figura 30 mostra o mapa de calor da taxa de saturação da NoC 5x5, utilizando os raios 1, 4 e 5, e as cargas baixa (pacotes gerados por 1/10 da rede a cada ciclo) e alta

(pacotes gerados por 1/5 da rede a cada ciclo). Nos primeiros dois casos, por conta de utilizar um raio baixo, onde as mensagens serão trocadas apenas com roteadores vizinhos, o uso da rede é homogêneo, apresentando baixa utilização para todos os roteadores. Conforme o raio é aumentado, são observadas concentrações de uso na rede, principalmente em tráfegos mais intensos, como pode ser observado na Figura 30 (d) e Figura 30 (f).

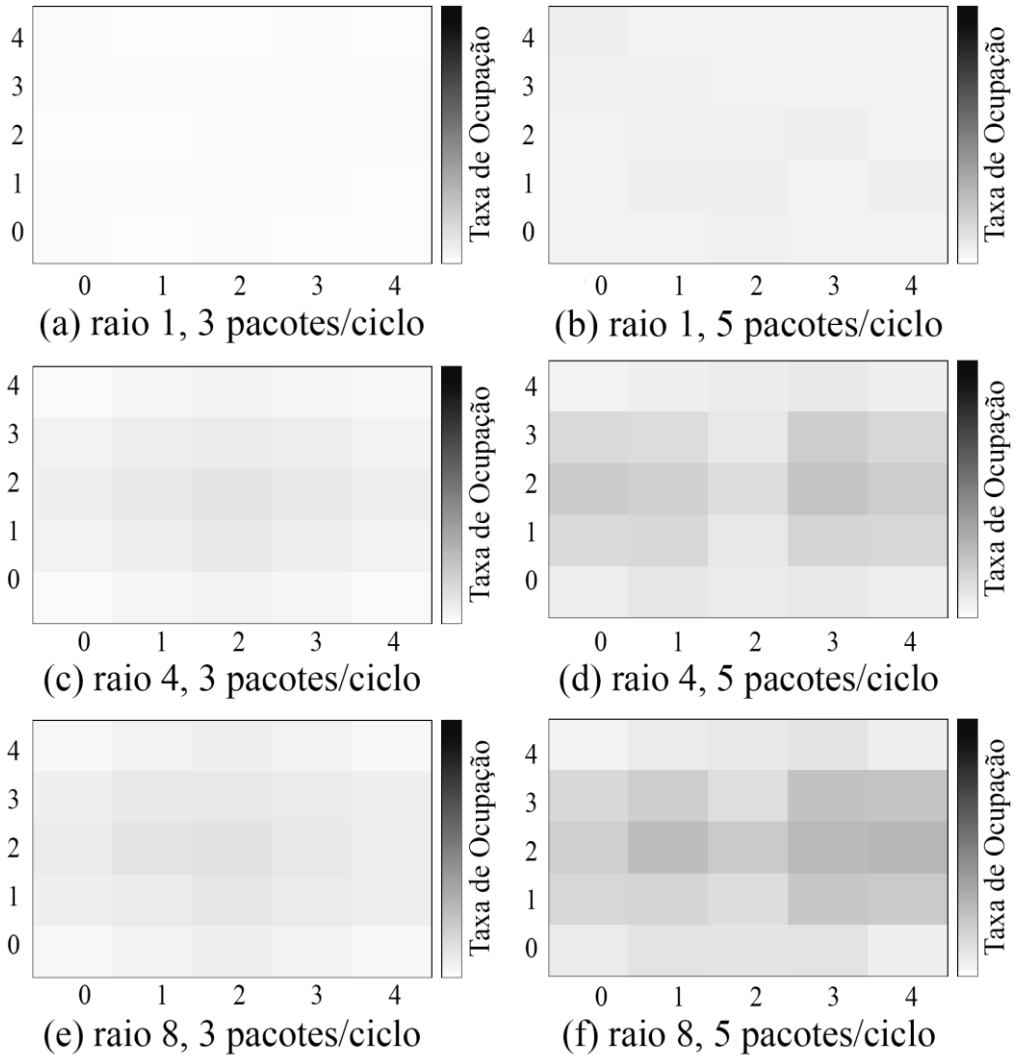
Figura 30: Mapa de calor da taxa de saturação: Rede 5x5.



Fonte: do autor

A Figura 31 apresenta o mapa de calor da taxa de ocupação da mesma rede citada acima. No geral, os comportamentos dos casos são bem semelhantes aos observados na Figura 30.

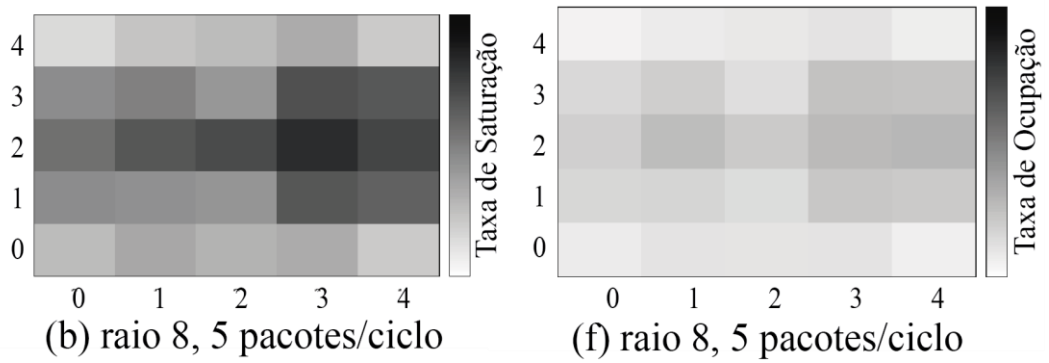
Figura 31: Mapa de calor da taxa de ocupação: Rede 5x5.



Fonte: do autor

Embora os comportamentos descritos aconteçam de maneira bem similar tanto nos mapas de calor da taxa de ocupação quanto nos das taxas de saturação, os gráficos podem oferecer informações que se complementam. Para ilustrar a importância da utilização destes dos dois tipos de análises, a Figura 32 (a) destaca o mapa de calor da taxa de saturação e taxa de ocupação dos *buffers* de entrada do cenário 5x5, raio 8, 17 pacotes por geração.

Figura 32: Cenário 5x5, raio 8, 5 pacotes: (a) Mapa de calor da taxa de saturação; (b) Mapa de calor da taxa de ocupação.

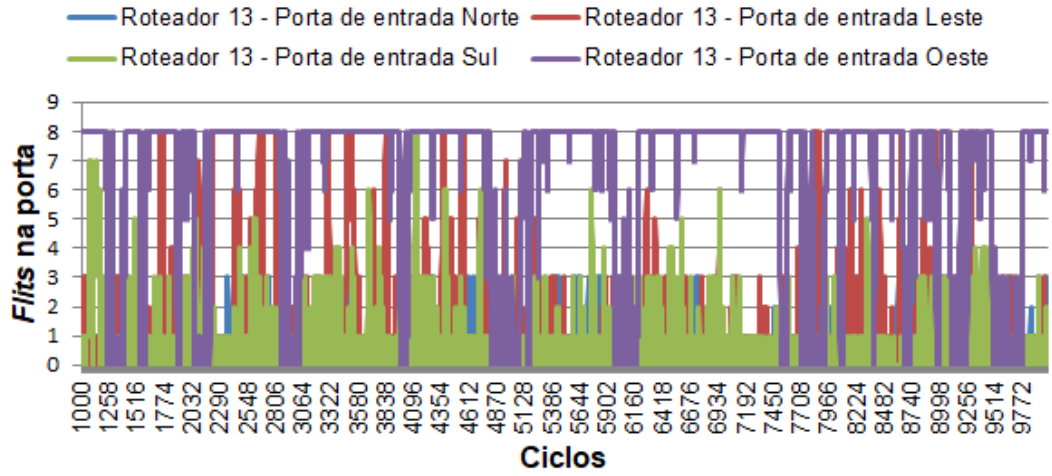


Fonte: do autor

Na Figura 32 (a), observa-se que os roteadores mais à direita da rede tiveram mais casos de saturação nas portas de saída. Porém, as saturações ocorridas nestes roteadores podem ter sido pontuais, decorrente do uso elevado em apenas uma ou em poucas portas do roteador. Ao se observar a taxa de ocupação dos roteadores na Figura 32 (b), é possível observar que, embora a maioria dos roteadores da parte direita da rede apresentassem casos de saturação, nenhum mostrou uma taxa de ocupação média alta (vide Figura 32 (a)).

Para reforçar esta constatação, foram analisados os estados de cada porta do roteador 13, dado pela coordenada (3,2), ao longo da execução. Dessa forma, pode-se determinar qual porta teve a maior sobrecarga. Esta análise é apresentada na Figura 33. É possível observar que a porta Oeste foi a que teve a maior utilização, estando completamente ocupada na maior parte do tempo.

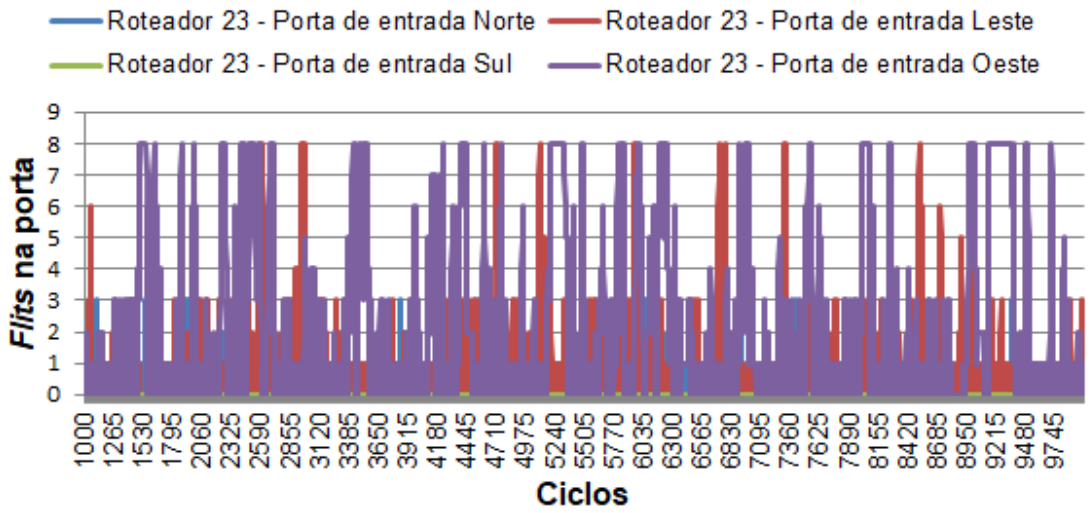
Figura 33: Roteador 13 da Rede 5x5 – Utilização das portas de entrada



Fonte: do autor

Ainda neste cenário, também foi analisado o roteador 23, dado pela coordenada (3,0) por apresentar uma utilização baixa/média tanto no mapa de calor da taxa de ocupação, quanto no de saturação. A utilização individual das portas desse roteador é mostrada na Figura 34.

Figura 34: Roteador 23 da Rede 5x5 – Utilização das portas de entrada



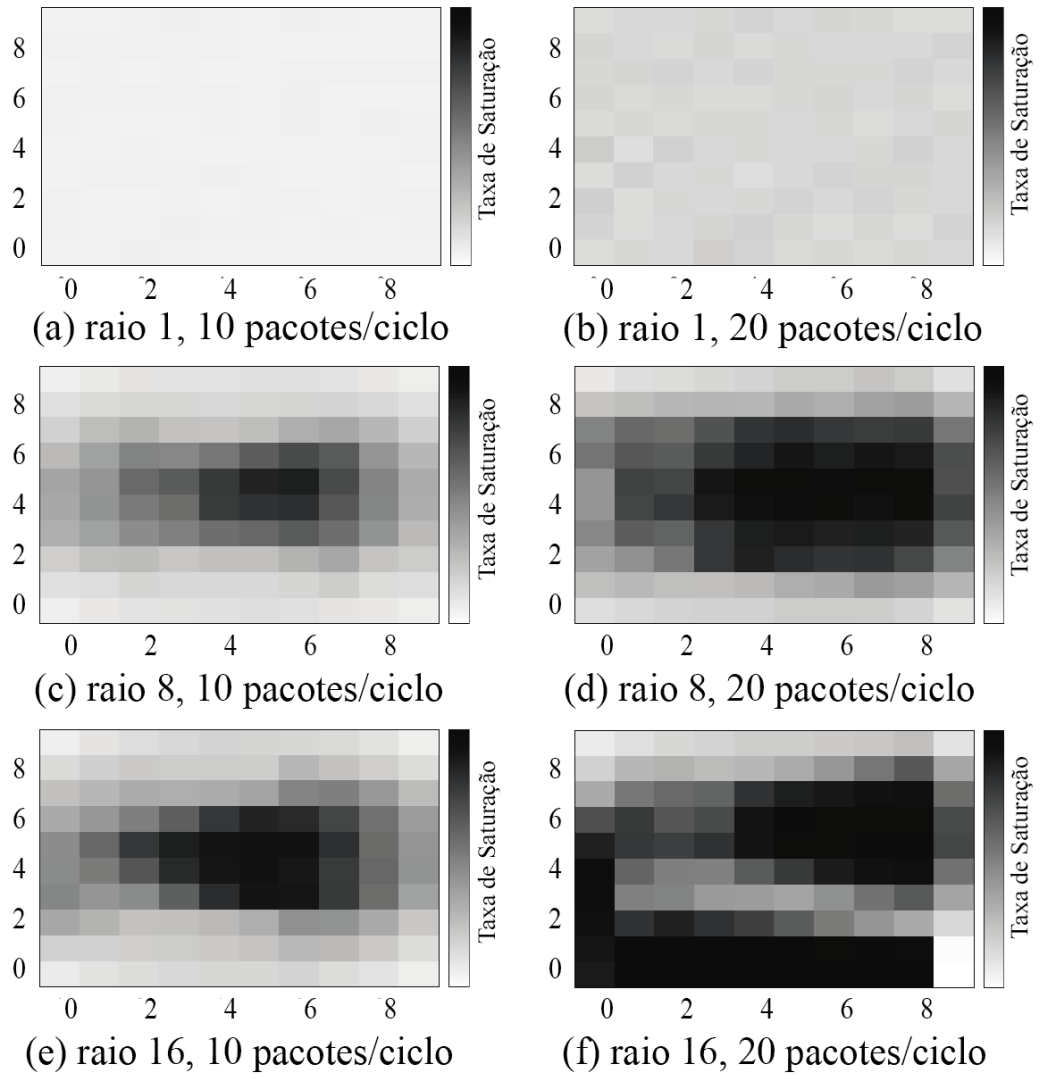
Fonte: do autor

Diferente da análise do roteador 13, onde uma porta mostrou alta atualização durante vários ciclos consecutivos, as portas Leste e Oeste demostraram alta utilização, mas em ciclos mais espalhados. Além de não mostrar utilização máxima contínua, outro fator que contribuiu para que este roteador demonstrasse uma taxa de ocupação menor

quando comparada à do roteador 4 é a utilização das demais portas: as portas Norte e Sul demonstraram utilização baixa, fazendo com que a média das portas do roteador, em geral, baixasse.

A Figura 35 apresenta o mapa de calor da taxa de saturação de uma NoC 10x10, utilizando os raios 1, 8 e 16, com cargas baixa e alta. O padrão da utilização da rede é similar ao encontrado nas análises da NoC anterior, 5x5. A principal diferença está na concentração da utilização: é possível observar que, com o aumento do tamanho da rede, a utilização passa a ficar mais concentrada, ou seja, um seleto grupo de roteadores apresenta a utilização mais intensa do que a maioria dos roteadores da rede. O comportamento é observado nos mapas de calor através da coloração mais intensa.

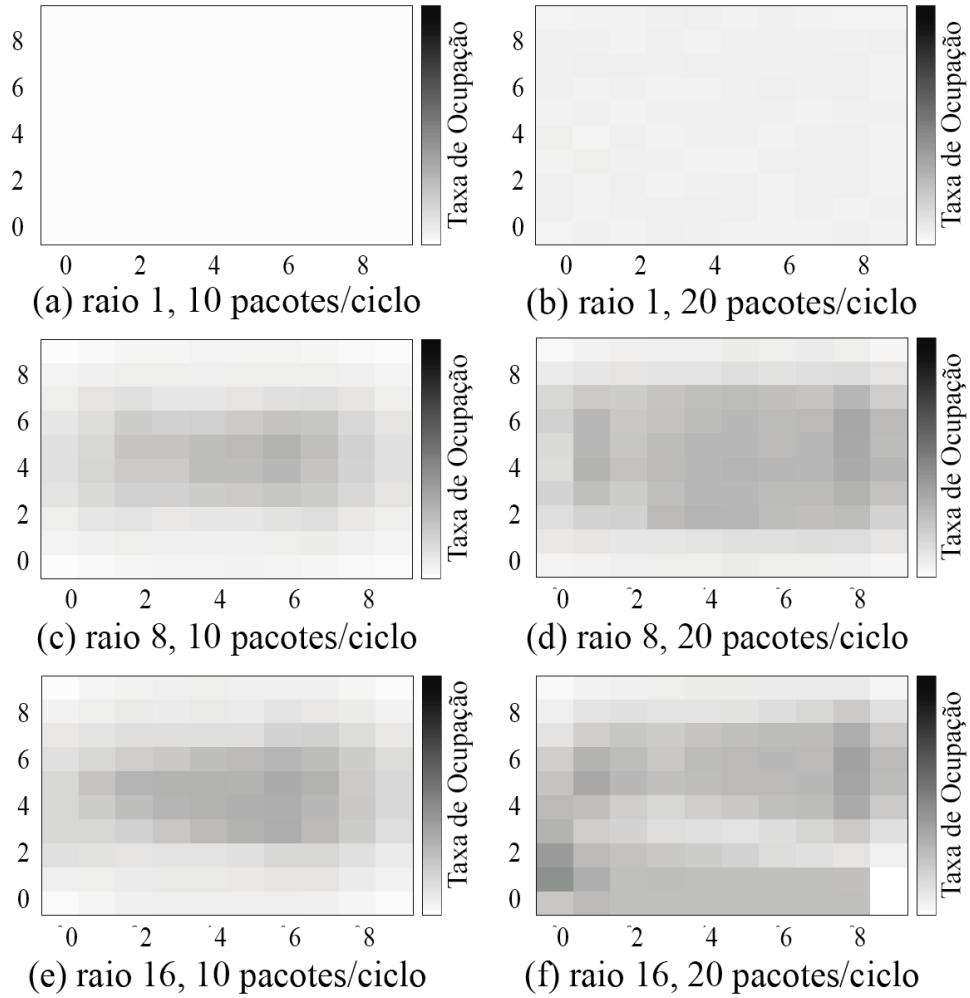
Figura 35: Mapa de calor da taxa de saturação: Rede 10x10.



Fonte: do autor

A Figura 36 ilustra o mapa de calor da taxa de ocupação da rede 10x10, utilizando os mesmos valores para os raios e a geração de carga mostrados na descrição anterior. O comportamento geral é similar ao que foi observado na rede 5x5: a taxa de ocupação apresenta, em um âmbito geral, um comportamento bem parecido com o da taxa de saturação. Pode-se destacar que a maior fonte de contenção ocorre nos roteadores mais ao centro da rede, como espero com o uso do algoritmo XY.

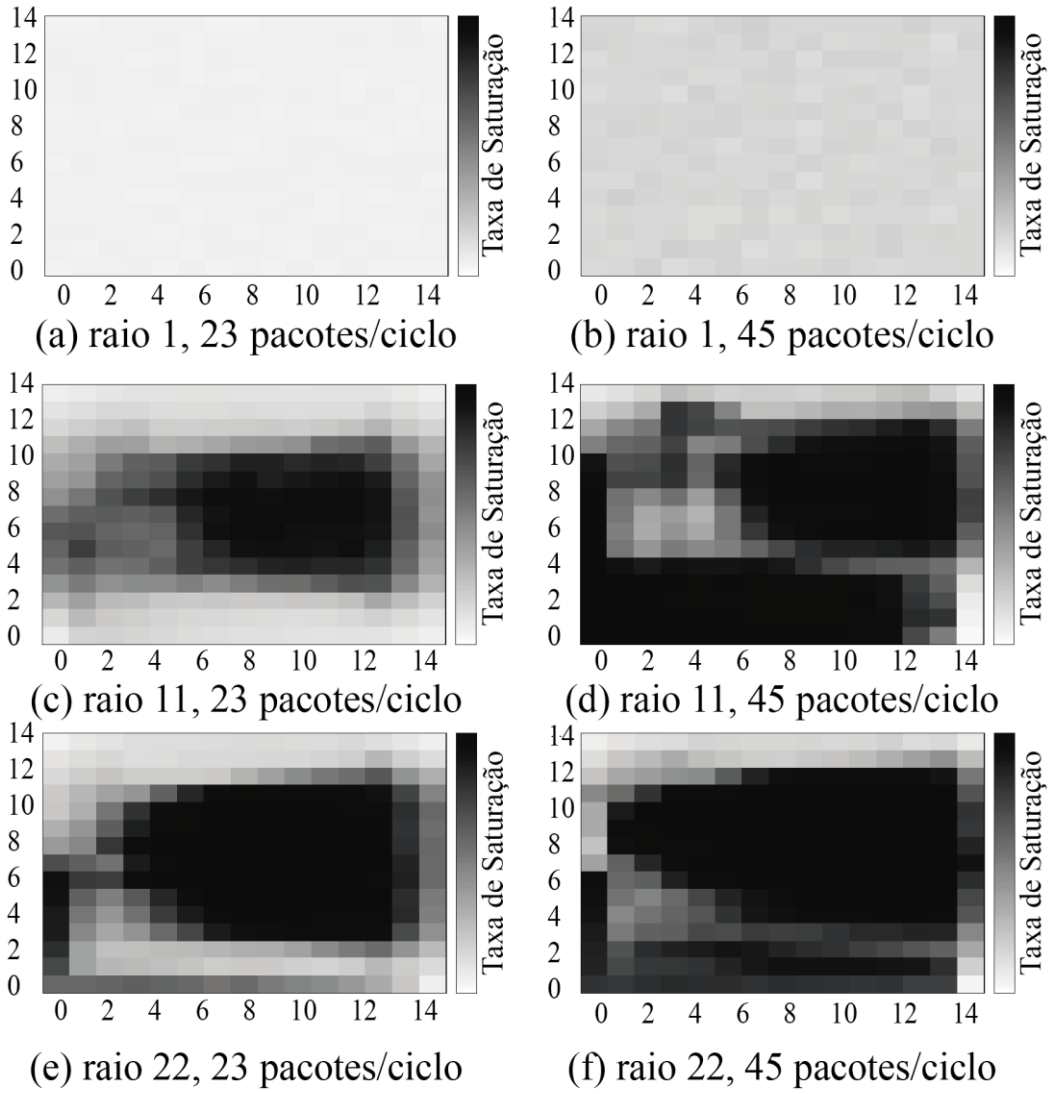
Figura 36: Mapa de calor da taxa de ocupação: Rede 10x10.



Fonte: do autor

O mapa de calor da taxa de saturação da simulação com a rede 15x15 é apresentada na Figura 37. Neste cenário são utilizados os raios 1, 11 e 22, além das cargas baixa e alta. O comportamento da utilização é similar ao comportamento da rede 10x10, com o detalhe da concentração dos pontos da saturação em determinadas áreas, por conta do aumento da rede.

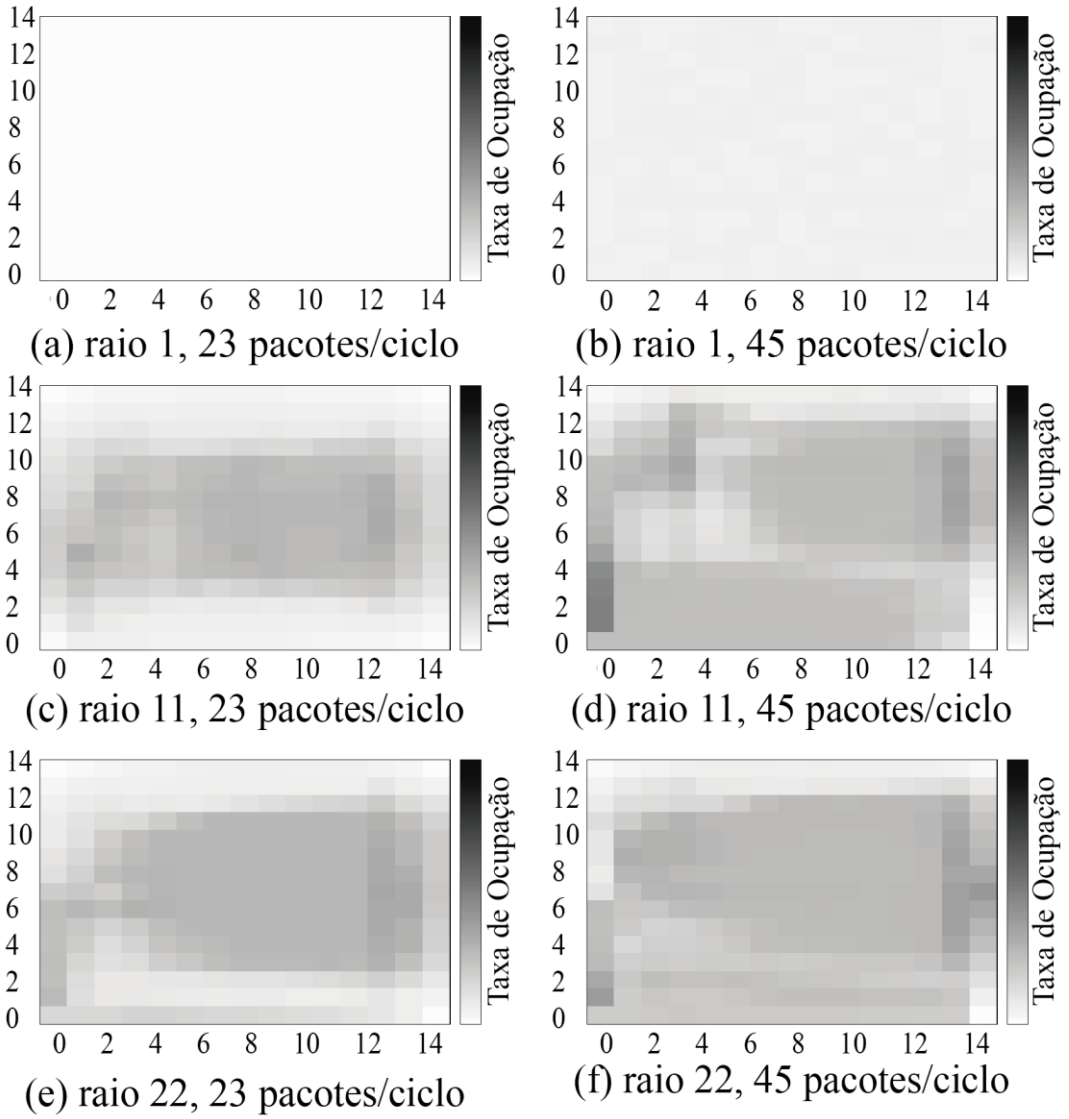
Figura 37: Mapa de calor da taxa de saturação: Rede 15x15.



Fonte: do autor

A Figura 38 ilustra o mapa de calor da taxa de ocupação da rede 15x15, para os mesmos raios e cargas utilizados na taxa de saturação. Na análise desta rede, também é importante a observação dos dois tipos de mapas de calor para observar roteadores que tiveram uso intenso. Na Figura 37 (f) observa-se uma utilização intensa quase em toda a rede, principalmente no centro e no canto inferior esquerdo. Já na Figura 38 (f) é possível observar que a utilização, no geral, foi média.

Figura 38: Mapa de calor da taxa de ocupação: Rede 15x15.



Fonte: do autor

De maneira geral, é possível observar nas simulações que a utilização dos roteadores aumenta com o tamanho da rede, apresentando uma coloração mais intensa. Isso se deve ao fato de que, ao se utilizar um raio maior, mensagens são trocadas entre roteadores distantes. Por estes roteadores estarem distantes uns dos outros, os *flits* acabam passando por mais roteadores intermediários, aumentando o uso da rede como um todo. Por existirem mais *flits* trafegando pela rede, a utilização da rede aumenta.

6.3 Grupo 3: Escalabilidade do Simulador

Foram realizados testes com o objetivo de mostrar o impacto no tempo de simulação ao se alterar certos parâmetros. Foram variados o tamanho da rede, a utilização da interface gráfica durante a simulação, a habilitação da geração de *log* e a frequência com que informações são registrados no *log*.

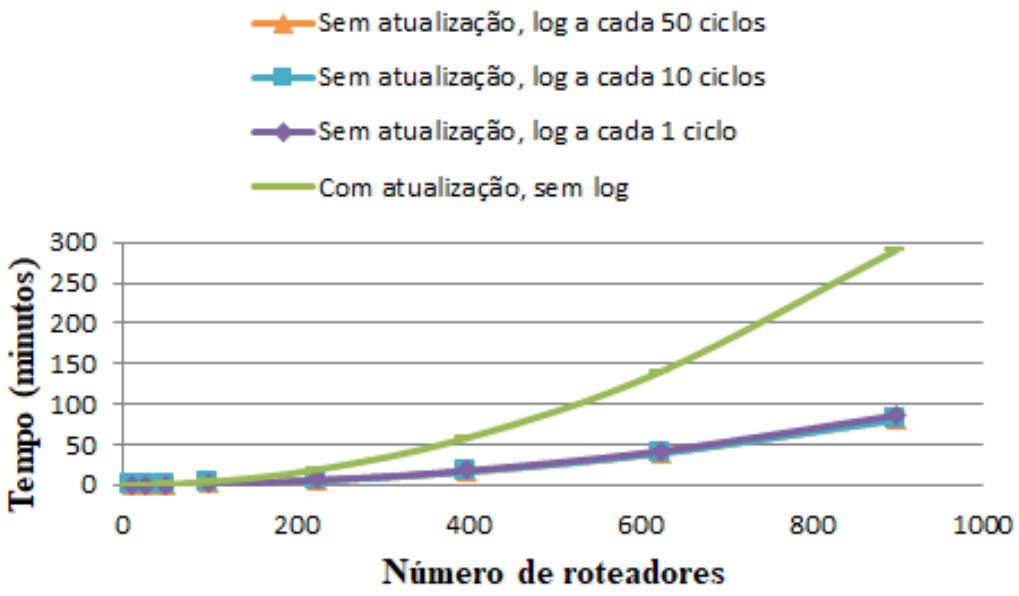
O computador utilizado para realizar as simulações é equipado com um processador Intel Core i5 4440, com 4 *cores* e 4 *threads* funcionando a 3.1GHz, com 8 GB de RAM (1600MHz DDR3). O sistema operacional utilizado foi o *Windows* 10 e a versão do NetLogo utilizada foi a 6.0.4.

Os cenários utilizam alguns parâmetros em comum com a configuração padrão dos cenários apresentados na Tabela 7. O único parâmetro diferente é o tipo de tráfego, sendo utilizado o tráfego aleatório pelo raio, com geração de carga a cada 1 ciclo. Os testes para observar o desempenho do simulador utilizam tráfego do tipo aleatório pelo raio, sendo utilizado o valor 10 para o raio, e a geração de carga é de 10 pacotes a cada ciclo.

Com base na configuração padrão apresentada acima, foram utilizados cenários com base em 8 tamanhos de rede, 9 roteadores (3x3), 25 roteadores (5x5), 49 roteadores (7x7), 100 roteadores (10x10), 225 roteadores (15x15), 400 roteadores (20x20), 625 roteadores (25x25) e 900 roteadores (30x30). Para cada tamanho de rede, foram testados quatro tipos diferentes de utilização do simulador: atualização da interface gráfica ligada, sem geração de *logs*; sem atualização da interface gráfica, imprimindo informações no *log* a cada 1 ciclo; sem atualização da interface gráfica, imprimindo informações no *log* a cada 10 ciclos; sem atualização da interface gráfica, imprimindo informações no *log* a cada 50 ciclos.

As informações obtidas através destas simulações são apresentadas na Figura 39.

Figura 39: Tempo de execução da simulação em função do tamanho da NoC e do *log*



Fonte: do autor

A Figura 39 mostra o aumento no tempo de execução da simulação em função do tamanho da NoC e do log. É possível constatar o parâmetro que tem maior impacto no tempo de simulação é o uso da interface gráfica. A utilização da interface gráfica esperada é parar ver comportamentos pontuais durante a simulação. Para a observação de comportamentos mais complexos, que requerem análises mais detalhadas, o *log* se mostra uma opção mais viável. É possível obter melhor desempenho diminuindo a frequência com que o simulador coleta informações sobre a simulação. Porém, vale ressaltar que, ao se diminuir os ciclos que são mostrados no *log*, o *log* irá contar com menos informações, tornando a análise menos precisa. Por fim, da mesma forma que a interface gráfica, o aumento no tamanho da rede tem um alto impacto no tempo de simulação. Ao simular redes maiores, o volume de informações que o simulador precisa manipular é maior, como os roteadores, seus *buffers* e mais informações de carga.

7 DISCUSSÃO

Este trabalho apresentou as etapas de construção de um simulador de NoCs modelado como um sistema multiagente. O simulador tem como objetivo ser uma ferramenta de simulação para NoCs modelada com um alto grau de abstração. Esta abstração provê independência do hardware, o que possibilita que a grande maioria das NoCs sejam implementadas e testadas. A motivação para a modelagem deste simulador como um sistema multiagente vem da comparação de características de um roteador e um agente, sendo possível constatar que os roteadores podem ser tratados como agentes, que desempenham funções importantes em seu ambiente e que compõem um sistema complexo, onde devem trabalhar em conjunto para atingir um objetivo maior. Dessa forma, a modelagem de NoCs através de um SMA, é viável.

Para compreender melhor as áreas que abrangem este estudo, os Capítulos 2 e 3 definiram os principais conceitos da área de NoCs e SMA. A parte de NoCs foi importante para entender os conceitos e funcionamento das partes que compõem o simulador, enquanto o estudo sobre SMA foi importante para entender como estes sistemas são formados, além de apresentar como os agentes são classificados e como se comportam.

O Capítulo 4 mostrou os simuladores de NoC. Na análise destes simuladores, foi possível observar que cada simulador apresenta características diferentes, que podem variar conforme a sua utilização. Foram apresentados em mais detalhes os simuladores que possuem mais características em comum com o aqui proposto, como o teste de algoritmos de roteamento e possuir interface gráfica. Dentre os simuladores mais parecidos, encontrou-se espaço para o desenvolvimento de um novo visando aprimorar alguns detalhes, como apresentar comportamentos da simulação em tempo de execução e também após o término, possibilidade de configurar diversos parâmetros de entrada e obter vários tipos de métricas.

O Capítulo 5 apresentou o desenvolvimento do simulador em mais detalhes. A descrição do simulador começa com os detalhes da comparação das semelhanças entre agentes e roteadores, que motivou o desenvolvimento e análise desse simulador como um sistema multiagente. Depois, é apresentada uma visão geral do simulador, onde são mostrados detalhes da estrutura interna dos roteadores que são simulados na ferramenta. O Capítulo também mostra quais são os parâmetros de entrada, métricas obtidas na simulação, além de descrever detalhes de funcionamentos internos dos componentes do simulador.

Para testar as capacidades da ferramenta, além de atingir os objetivos específicos do estudo, o Capítulo 6 apresentou os resultados obtidos no teste de alguns cenários criados. A primeira parte dos testes mostrou como a variação de alguns parâmetros de entrada e as contenções podem afetar os valores de número de saltos e a latência que dos *flits* da simulação. A segunda parte dos testes mostrou como diferentes configurações de NoC afetam a utilização dos roteadores. Por fim, a terceira parte dos testes apresentou o impacto de alguns atributos no tempo de execução do simulador.

Ao desenvolver um simulador de NoCs, busca-se auxiliar os usuários a projetar NoCs que atendam a suas demandas, observando o impacto da mudança dos parâmetros através de simulações. O impacto é observado através dos indicadores de saída que o simulador disponibiliza em tempo de execução e ao final da simulação. Além disso, o uso de do código aberto no desenvolvimento facilita a sua utilização, tanto para estudos acadêmicos, quanto para implementações de NoCs reais para a indústria, além de favorecer possíveis extensões do simulador.

7.1 Contribuição Acadêmica

Este estudo gerou contribuições, na forma de artigos, ao longo de seu desenvolvimento.

O primeiro artigo publicado foi no evento Workshop - Escola de Sistemas de Agentes, seus ambientes e aplicações WESAAC (LIMA, 2018a). A publicação contém informações da primeira versão do simulador, e o trabalho foi apresentado a comunidade de agentes, de forma a validar a viabilidade da construção deste simulador.

O segundo artigo foi publicado no evento Workshop on Circuits and Systems Design (LIMA, 2018b). De maneira similar ao evento anterior, esse artigo apresentou o

desenvolvimento da versão inicial do simulador, porém com foco maior na análise das NoCs, por se tratar de um evento de microeletrônica.

Por fim, o terceiro artigo publicado foi no evento *IEEE International Conference on Electronics Circuits and Systems* (LIMA, 2018c). Nessa publicação, foram comparados diferentes algoritmos de roteamento, XY e WF, mas ainda na versão anterior do simulador.

As três publicações apresentaram o protótipo do simulador. É importante de ressaltar que diversos comportamentos mudaram das publicações para o simulador descrito nesse documento. A estrutura interna do roteador da versão anterior era diferente, considerando menos comportamentos. Além disso, foram adicionadas novos parâmetros de entrada e novas métricas de simulação.

7.2 Trabalhos Futuros

Como trabalhos futuros, deseja-se explorar a capacidade dos sistemas multiagente para criar novos mecanismos para NoCs, como novos algoritmos de roteamento ou novos árbitros. Um algoritmo que utiliza técnicas de coordenação de agentes poderia distribuir melhor a utilização da rede, evitando que determinadas rotas fiquem sobrecarregadas enquanto outras estão livres. Uma nova implementação de árbitro também pode ajudar em distribuir melhor a utilização dos roteadores internamente, o que também irá diminuir sobrecargas. Por fim, também deseja-se implementar novos padrões de carga, visando criar tráfegos que podem se assemelhar mais a aplicações reais de NoCs. Todas estas implementações podem expandir a utilização do simulador.

Além das implementações, deseja-se realizar mais testes no simulador. A exploração de outros cenários, por exemplo considerando outros custos no roteamento e na transmissão de pacotes, pode ajudar a simular NoCs mais realistas. Nesse contexto, também é interessante recriar NoCs conhecidas na literatura. Isso pode permitir uma avaliação do quanto a simulação se aproxima de resultados conhecidos na literatura.

Por fim, por conta de o simulador utilizar uma licença de código livre, os usuários podem implementar novas funcionalidades. Dessa forma, o número de

parâmetros da entrada que são configurados para realizar uma simulação pode aumentar significativamente. Nesse sentido, pensa-se na criação de arquivos de configuração, ou *presets*. Esses arquivos carregam vários parâmetros pré-definidos, facilitando o uso do simulador. Além disso, com o uso destes arquivos será possível carregar antecipadamente parâmetros mais gerais, deixando para modificar na interface parâmetros mais pontuais.

REFERÊNCIAS

ACHBALLAH, A. B., SAOUD, S. B. A Survey of Network-On-Chip Tools, International Journal of Advanced Computer Science and Applications, vol. 4 no. 9, 2013.

ALALAKI, M. S., AGYEMAN, M. O. "A Study of Recent Contribution on Simulation Tools for Network-on-Chip", International Journal of Cumpter Systems, vol. 11 no.4, 2017.

AL-BADI et. al, "A parameterized NoC simulator using OMNet++," Int. Conference on Ultra Modern Telecommunications & Workshops, St. Petersburg, 2009, pp. 1-7.

ALI et al., "Using the NS-2 Network Simulator for Evaluating Network on Chips (NoC)", in International Conference on Emerging Technologies, 2006, pp. 506 - 512.

AMANDI, A.A. Programação de Agentes Orientada a Objetos. Porto Alegre: CPGCC da UFRGS, 1997. Tese de Doutorado.

ATLAS. ATLAS website. Disponível em: < https://corfu.pucrs.br/redmine/projects/atlas/wiki >. Acesso em: jan. 2018.

BEN et al, "HNOCS: Modular open-source simulator for Heterogeneous NoCs," in Embedded Computer Systems (SAMOS), 2012 International Conference on, Samos, 2013.

BERTOZZI, D., BENINI, L. "A network-on-chip architecture for gigascale systems-on-chip". IEEE Circuits and Systems Magazine, 2004.

BRIAO, E. W. Métodos de Exploração de Espaço de Projeto em Tempo de Execução em Sistemas Embarcados de Tempo Real Soft Baseados em Redes-em-Chip. Tese de doutorado (Ciência da Computação). UFRGS. Porto Alegre. 2008.

CARARA, E. A. Uma exploração arquitetural de redes intra-chip com topologia malha e modo de chaveamento Wormhole. Trabalho de Conclusão de Curso, 2004.

CARDOZO, E.; MAGALHÃES, M.F. Redes de computadores: modelo OSI, 2012. Disponível em: < ftp://ftp.dca.fee.unicamp.br/pub/docs/eleri/apostilas/osi.pdf >. Acesso em: ago. 2017.

CARLSON et al, "Sniper: Exploring the Level of Abstraction for Scalable and Accurate Parallel Multi-Core Simulation," Intel Labs Europe, 2015.

CATANIA et. al, "Noxim: An Open, Extensible and Cycle-accurate Network on Chip Simulator", in Application-specific Systems, Architectures and Processors (ASAP), 2015 IEEE 26th International Conference on, Toronto, 2015.

CHAIN. Silistix website. Disponível em: < www.silistix.com >. Acesso em: jan. 2018.

CHAN, J., PARAMESWARAN, S., "NoCGEN: A Template Based Reuse Methodology for Networks on Chip Architecture," in IEEE 17th International Conference on VLSI Design, 2004, pp. 717 - 720.

CHANG et al, Surviving the SOC revolution: a guide to platform-based design". Kluwer Academic Publishers, Norwell, MA. pg. 1-27. 1999.

CHARNIAK, E.; MCDERMOTT, D. A Bayesian model of plan recognition. Massachussets: Addison-Wesley. 1985.

CHAWADE et. al, "Design XY Routing Algorithm for Network-on-Chip Architecture", International Journal of Computer Application, vol. 43, 2012.

CHEN et al., 16.1 A 340mV-to-0.9V 20.2Tb/s source-synchronous hybrid packet/circuit-switched 16×16 network-on-chip in 22nm tri-gate CMOS, in Solid-State Circuits Conference Digest of Technical Papers (ISSCC), IEEE International, 2014.

CHIOU, D. "MEMOCODE 2011 Hardware/Software CoDesign Contest: NoC simulator," ACM/IEEE Int. Conference on Formal Methods and Models for Codesign (MEMPCODE2011), 2011, pp. 73-76.

COTA et. al, "Reliability, Availability and Serviceability of Networks-on-Chip". Springer US. 2012.

EVAIN, et al., "μSpider: a CAD Tool for Efficient NoC Design," in Proceedings of the Norchip Conference, 2004, pp. 218 - 221.

EZZ-ELDIN, R.; EL-MOURSY, M.A,; HAMED, H.F.A. Analysis and Design of Networks-on-Chip Under High Process Variation. p34-41, 2015.

FERBER, J. Multi-agent systems: An introduction to distributed artificial intelligence 1st Edition. Boston, MA, USA. 1999.

FLEXNOC. Arteris website. Disponível em: < www.arteris.com/flex_noc.php >. Acesso em: jan. 2018.

FROZZA, R. Ambiente para Desenvolvimento de Sistemas Multiagentes Reativos. Dissertação (Mestrado em Ciência da Computação). Universidade Federal do Rio Grande do Sul. Porto Alegre. 1997.

GENESERETH, M. R; KETCHAPEL, S. P. Software agents. Communications of the ACM, 37(7), pg 48-53. 1994

GHOSH et. al, "A Highly Parameterizable Simulator for Performance Analysis of NoC Architectures," Int. Conference on Information Technology, 2014, pp. 311-315.

GRAAF et al, "Embedded software engineering: the state of the practice". IEEE Software, v20, n6, pg 61-69. 2003.

GREGOLIN, Vanderlei. Conceitos Matemáticos em ambiente Logo. São Paulo, UFSC Ar/ CECH/ PPGE, 1994.

GUDERIAN et al, "Fair rate packet arbitration in Network-on-Chip". IEEE International SOC Conference. 2011.

HAHNE, Ellen L., GALLAGER,S Robert G. Round robin scheduling for fair flow control in data communication networks. IEEE Internetional Conference on Communications, June 1986.

HEMANI et al., "Network on a Chip: An architecture for billion transistor era", Norchip-2000, pp. 166-173.

HENNESSY, J. L.; PATTERSON, D. A. Organizacao e Projeto de Computadores - A Interface Hardware Software. 3 edition, 2005.

HENNESSY, J. L.; PATTERSON, D. A. Arquitetura de Computadores: Uma abordagem quantitativa. Elsevier, 4 edition, 2008.

HOSSANI et al, "Gpnocsim - A General Purpose Simulator for Network-On-Chip," in Information and Communication Technology, 2007. ICICT '07. International Conference on, Dhaka, 2008.

HUBNER, J. Um modelo de reorganização de sistemas multiagentes. Tese de doutorado: Escola Politécnica da Universidade de São Paulo, p246. 2003.

INOC. iNoCs website. Disponível em: < www.inocs.com >. Acesso em: jan. 2018.

JIANG et. al, "A Detailed and Flexible Cycle-Accurate Network-on-Chip Simulator". IEEE International Symposium On Performance Analysis of Systems and Software (ISPASS). 2013.

KAHNG et al., "ORION 2.0: A Fast and Accurate NoC Power and Area Model for Early-Stage Design Space Exploration " in Proceedings of Design Automation and Test in Europe, 2009.

KAMALI et. al, "DuCNoC: A High-Throughput FPGA-Based NoC Simulator Using Dual-Clock Lightweight Router Micro-Architecture," EEE Trans. on Computers, vol. 67, no. 2, pp. 208-221, Feb. 1 2018.

KAMALI, H. M., HESSABI, S. "AdapNoC: A fast and flexible FPGA-based NoC simulator", Int. Conf. on Field Programmable Logic and Applications (FPL), 2016, pp. 1-8.

KUMAR, V., KUMAR, S. "Design and Implementation of Router Arbitration in Network on Chip". International Journal of Engineering Research & Technology (IJERT). 2014.

KURZWEIL, R. The Age of Spiritual Machines. Massachusetts: The MIT Press. 1990.

LIMA, G. L. et al, "Simulador de NoCs modelado como um SMA", 12th Workshop-School on Agents, Environments and Applications (WESAAC), 2018a, pp. 167-178.

LIMA et al. "Development of a NoC Simulator in a MultiagentSystem Environment". In: WCAS 2018 - Workshop on Circuits and Systems Design, 2018b.

LIMA et. al. Exploring MAS to a High Level Abstration NoC Simulation Environment. In: IEEE ICECS 2018 - IEEE International Conference on Electronics Circuits and Systems, Bordeaux, 2018c.

LIS et al., "DARSIM: a parallel cycle-level NoC simulator", in IEEE Asian SolidState, Circuits Conference, Saint Malo, 2010.

MACAL, C.; NORTH, M. Introductory tutorial: Agent-based modeling and simulation. In: Proceedings of the 2014 Winter Simulation Conference. Piscataway, NJ, USA: IEEE Press, 2014. (WSC ’14), p. 6–20.

MOORE, G. E. "Cramming More Components onto Integrated Circuits", Electrinocis Magazine, 1965.

MORAES et al, "HERMES: an Infrastructure for Low Area Overhead Packet-switching Networks on Chip". Integration, Volume 38, Issue 1, pg 69-93. 2004.

NIKOUNIA, S. H., MOHAMMADI, S., "Gem5v: a modified gem5 for simulating virtualized systems," The Journal of Supercomputing, vol. 71, no. 4, p. 1484–1504, 2015.

MURALI, S., MICHELI, G. D., "SUNMAP: A Tool for Automatic Topology Selection and Generation for NoCs," in ACM/IEEE Design Automation Conference, 2004.

NAIR, T. R. G.; SOODA, K. Routing Techniques using Cognitive Approach for Realizing a Network On-chip. Disponível em: < https://www.iconceptpress.com/book/networks--emerging-topics-in-computer-science/11000032/1109000228.pdf>. Acesso em: out. 2017.

NS-2. "NS-2 website". Disponível em: <nsnam.isi.edu/nsnam/index.php/Main_Page>. Acesso em: jan. 2018.

NUNES, C. A. K. Uma estratégia para redução de congestionamento em sistemas multiprocessadores baseados em NOC**.** 2015. Dissertação (Mestrado em Ciência da Computação). Centro de Informática, UFPE, Pernambuco.

OMENA, Felipe Marques Alves, "Uma Biblioteca Orientada a Aspectos para Modelagem de simulações Sociais". Dissertação (Engenharia de Computação). Universidade de Pernambuco. 2014.

OPREA, M. Applications of MultiAgent Systems. In: Reins R. (eds) Information Technology, IFRIP International Federation for Information Processing, vol 157. Springer, Boston, MA. Disponível em: <https://link.springer.com/chapter/10.1007/1-4020-8159-6_9>. Acesso em: out. de 2017.

OSI. Open System Interconnection in International Organization for Standardization. 1977.

OXMAN, G., WEISS, S. "An NoC Simulator That Supports Deflection Routing, GPU/CPU Integration, and Co-Simulation," IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, vol. 35, no. 10, pp. 1667-1680, Oct. 2016.

PALERMO, G., SILVANO, C., "PIRATE: A Framework for Power/Performance Exploration of Network-on-Chip Architectures," in 14th International Workshop Power and Timing Modeling, Optimization and Simulation, 2004, pp. 521-531.

PEREIRA, Carlos Eduardo de Quadros, "BioTraffic: um modelo comportamental bio-inspirado para a geometria veicular bidimensional". Mestrado (Modelagem Computacional). FURG. 2015.

POOLE, D. A.; MACKWORTH, A.; GOEBEL, R. Computational Intelligence: A logical approach. Oxford: Oxford University. 1998.

POWER et al, "gem5-gpu: A Heterogeneous CPU-GPU Simulator," IEEE Computer Architecture Letters, vol. 14, no. 1, pp. 34 - 36, 2014.

REIS, L. P. Coordenação em Sistemas Multiagente: Aplicações na Gestão Universitária e Futebol Robótico. 2003. Disponível em: < https://paginas.fe.up.pt/~niadr/PUBLICATIONS/thesis_PhD/PhD_LuisPauloReis.pdf>. Acesso em: out. 2017.

RUSSEL, S.; NORVIG, P. Inteligência Artificial 2. Ed. Rio de Janeiro: Campos. 2004.

SANTANA, M. R. F. Uma abordagem meta-heurística para o mapeamento de tarefas em uma plataforma mpsoc baseada em noc. Tese de PhD, Universidade Federal de Pernambuco, 2014.

SEICULESCU et al., "SunFloor 3D: A Tool for Networks on Chip Topology Synthesis for 3D Systems on Chips," in Proceedings of the conference on Design, Automation and Test in Europe, 2009, pp. 9 - 14

TOPÎRCEANU, A. Networks On Chip, 2013. Disponível em: < https://sites.google.com/site/alexandrutopirceanu/research/networks-on-chips>. Acesso em: ago. 2017.

TSAI et. al, "Networks on Chips: Structure and Design Methodologies". Journal of Electrical and Computer Engineering. 2012.

VANGAL e. al, An 80-tile Sub-100-W TeraFLOPS processor in 65-nm CMOS, IEEE J. Solid-State Circuits, vol. 43, no. 1, pp. 29–41, Jan. 2007.

VIGLUCCI, P. W., CARPENTER, A., "ENoCS: An Interactive Educational Network-on-Chip Simulator," in ASEE Annual Conference & Exposition, New Orleans, 2016.

WANG et al, "DART: A programmable architecture for NoC simulation on FPGAs," IEEE Transactions on Computers, vol. 63, no. 3, pp. 664 - 678, 2014.

WEHNER et al, "MPSoCSim: An extended OVP Simulator for Modeling and Evaluation of Network-on-Chip based heterogeneous MPSoCs," in Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), Samos, 2015.

WILENSKY, 1997, U., NetLogo. Center for Connected Learning and Computer-Based Modeling, Northwestern University. Evanston, IL., 1999 WOOLDRIDGE, M. An Introduction to MultiAgent Systems. pg 225-226. 2009.

ZHUANSUN et al., "Multipath routing algorithm for application-specific wormhole NoCs", in Concurrency and Computation: Practice and Experience, 2017.

ZOLGHADR et. al, "GPU-based NoC simulator," ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMPCODE2011), 2011, pp. 83-88.