PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

MONITORAÇÃO E ROTEAMENTO ADAPTATIVO PARA FLUXOS QOS EM NOCS

LEONEL PABLO TEDESCO

TESE APRESENTADA COMO REQUISITO PARCIAL À OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIA DA COMPUTAÇÃO NA PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL

ORIENTADOR: PROF. DR. FERNANDO GEHM MORAES

PORTO ALEGRE
2010

Dados Internacionais de Catalogação na Publicação (CIP)

T256m Tedesco, Leonel Pablo
Monitoração e roteamento adaptativo para fluxos QoS em NoCs / Leonel Pablo Tedesco. – Porto Alegre, 2010.
132 p.

Tese (Doutorado) – Fac. de Informática, PUCRS.
Orientador: Prof. Dr. Fernando Gehm Moraes.

1. Informática. 2. Arquitetura de Redes. 3. Roteamento – Redes de Computadores. I. Moraes, Fernando Gehm. II. Título.

CDD 004.6

Ficha Catalográfica elaborada pelo
Setor de Tratamento da Informação da BC-PUCRS

Pontifícia Universidade Católica do Rio Grande do Sul
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TERMO DE APRESENTAÇÃO DE TESE DE DOUTORADO

Tese intitulada "Monitoração e Roteamento Adaptativo para Fluxos QoS em NoCs", apresentada por Leonel Pablo Tedesco, como parte dos requisitos para obtenção do grau de Doutor em Ciência da Computação, Sistemas Embarcados e Sistemas Digitais , aprovada em 12/05/2010 pela Comissão Examinadora:

Prof. Dr. Fernando Gehm Moraes – Orientador — PPGCC/PUCRS

Prof. Dr. Ney Laert Vilar Calazan - — PPGCC/PUCRS

Prof. Dr. Cesar Albenes Zeferino – — UNIVALI

Prof. Dr. Fabien Clermidy – — CEA-LETI

Homologada em 29/06/10..., conforme Ata No. 011.. pela Comissão Coordenadora.

Prof. Dr. Fernando Gehm Moraes
Coordenador.

PUCRS
Campus Central
Av. Ipiranga, 6681 - P. 32 - sala 507 - CEP: 90619-900
Fone: (51) 3320-3611 - Fax (51) 3320-3621
E-mail: ppgcc@pucrs.br
www.pucrs.br/facin/pos

AGRADECIMENTOS

Primeiramente, gostaria de agradecer aos meus pais Mauro Heitor Tedesco e Marina Izabel Tedesco pelo apoio recebido em todas as etapas de minha vida acadêmica. Espero que o enorme esforço realizado por vocês ao longo destes anos possa ter resultado no que hoje é motivo de orgulho. Eu amo vocês! Ao meu inseparável e querido irmão Leonardo Heitor Tedesco, que é e sempre será para mim uma referência de ser humano. Te amo, Mano!

À minha esposa Camila Pires Carvalho Tedesco, pela tua compreensão e carinho. Foram muitos dias chegando tarde em casa e alguns fins de semana sem tempo... Mas nem por isso deixaste de me apoiar a seguir confiante nesta caminhada. Me deste o maior presente da minha vida, que é a nossa filhinha Lívia. Nossa família continua crescendo, e com certeza, com muita união e amor superaremos juntos os desafios do futuro. Te amo, minha Vida!

Ao meu orientador e amigo Fernando Gehm Moraes. Foram até aqui quase sete anos de convivência. Aprendi e continuarei sempre aprendendo contigo, afinal, nossa relação pessoal e acadêmica permanece. Obrigado, Professor!

Ao CNPQ (Conselho Nacional de Desenvolvimento Científico e Tecnológico) pelo apoio financeiro prestado durante a realização do doutorado que originou esta Tese.

Aos meus grandes amigos Edson Moreno e Luciano Ost. Foram muitas as discussões a respeito do trabalho desenvolvido e o aprendizado adquirido, que vou levar para sempre comigo. Também não posso esquecer-me dos excelentes momentos passados extra-laboratório, onde se incluem incontáveis jogos e churrascos, desde os tempos de mestrado. Obrigado, Amigos!

Ao professor Ney Calazans, que ao longo destes anos prestou valiosa contribuição em todos os trabalhos desenvolvidos, acompanhando-me desde o mestrado. Aos professores César Marcon e Cesar Zeferino pelas contribuições nesta Tese. Ao pesquisador Fabien Clermidy, por ter me orientado durante o período que passei no CEA-LETI.

Aos colegas Rafael Soares, Alexandre Amory, Ewerson Carvalho, Everton Carara, Julian Pontes, Guilherme Guindani, Marcelo Mandelli, Thiago Raupp e Eduardo Wachter pela ótima convivência e amizade. Aos bolsistas Thiago Rosa e Guilherme Medeiros pelo auxílio nas ferramentas de apoio ao desenvolvimento dos métodos aqui apresentados.

Finalmente, agradeço a Deus por ter dado saúde, paz de espírito e harmonia a mim e aos meus familiares, para que eu pudesse atingir os meus objetivos.

MONITORAÇÃO E ROTEAMENTO ADAPTATIVO PARA FLUXOS QOS EM NOCS

RESUMO

O crescente número de aplicações executando em MPSoCs emergentes pode ser caracterizado pela sua alta demanda de computação e comunicação nas diferentes parte do chip. Os elementos de processamento que executam estas aplicações trazem uma natureza dinâmica e imprevisível para o tráfego em chip, devido à variabilidade nas taxas de injeção de dados que eles podem gerar. As redes em chip (NoC – do inglês *Network-on-Chip*) são as estruturas de comunicação a serem utilizadas em tais sistemas, devido ao seu desempenho, confiabilidade e escalabilidade. Para lidar com o comportamento dinâmico do tráfego de aplicações, vários métodos de adaptação são propostos em nível de sistema (em tempo de execução) e em nível de arquitetura (em tempo de projeto). Esta Tese aborda o uso de técnicas de adaptação em NoCs em nível de sistema e de arquitetura: *dimensionamento de buffer* e *roteamento adaptativo*. A primeira técnica introduz um *buffer* de desacoplamento (*D-buffer*) no IP destino. Este *buffer* recebe dados da NoC com jitter, enquanto que o IP destino consome dados deste *buffer* na taxa da aplicação, sem jitter. Dois problemas devem ser resolvidos para a implementação de *D-buffers*: (*i*) qual tamanho este *buffer* deve possuir? (*ii*) quanto tempo deve ser esperado antes do início do consumo de dados (*threshold*)? Propõe-se aqui um método geral para definir o tamanho e *threshold* de *D-buffers*, considerando a influência do empacotamento, arbitragem, roteamento e concorrência entre fluxos. A segunda técnica é um algoritmo de roteamento adaptativo para NoCs, onde o caminho entre o IPs origem e destino pode ser modificado devido a eventos de congestionamento. A maior parte das propostas do estado da arte possui uma visão limitada de congestionamento, considerando que cada roteador da NoC toma decisões baseado no estado de seus vizinhos. Esta decisão local pode rotear pacotes a outras regiões congestionadas, o que pode tornar o algoritmo ineficiente. Este trabalho apresenta um novo método onde a análise de congestionamento considera informações de todos os roteadores no caminho entre a origem e destino. Este método é composto por um protocolo para estabelecimento de sessões QoS, seguido de monitoração distribuída e re-roteamento para regiões não congestionadas. Resultados experimentais demonstram o impacto de fluxos multimídia com tamanhos de pacotes fixo e variável (a partir de *traces* reais de tráfego) no dimensionamento de *buffers*, e o percentual de violações de prazos em função do tamanho do *D-buffer*. Em termos de roteamento adaptativo, os resultados obtidos apresentam a influência de diferentes níveis de localidade de tráfego na latência de pacotes, ocupação da NoC e reatividade do roteamento adaptativo a eventos de congestionamento.

Palavras chave: redes intra chip, dimensionamento de *buffer*, congestionamento, roteamento na origem, qualidade de serviço, roteamento adaptativo, monitoração.

MONITORING AND ADAPTIVE ROUTING FOR QOS FLOWS IN NOCS

ABSTRACT

The growing number of applications running on emerging MPSoCs can be characterized by their high demand of computation and communication in different parts of the chip. The processing elements that execute these applications bring a dynamic and unpredictable nature to the on-chip traffic, due to the variability on data injection rates that they can generate. Networks on chip (NoCs) are the communication infrastructure to be used in such systems, due to their performance, reliability and scalability. To deal with the dynamic behavior of the application traffic, several methods are proposed at the system level (at runtime) and at the architecture level (at design time). The subject of this Thesis is the use of techniques for adaptability in NoCs at both system and architecture levels: *buffer sizing* and *adaptive routing*. The first technique introduces a decoupling buffer (*D-buffer*) on the target IP. This buffer receives data from the NoC with jitter, while the target IP consumes data from this buffer at the application rate, without jitter. Two problems must be solved to implement *D-buffers*: (*i*) which size must the buffer have? (*ii*) how much time should pass before data consumption starts (threshold)? A general method to define *D-buffer* size and threshold, considering the influence of packaging, arbitration, routing and concurrency between flows is presented. The second technique is an adaptive routing algorithm for NoCs, where the path between source and target IPs may be modified due to congestion events. The major part of the state of art proposals have a limited view of congestion, since each NoC router takes decisions based on the status of a few neighbors. Such local decisions may route packets to other congested regions, making the algorithm inefficient. This work presents a new method where congestion analysis considers information of all routers in the source-to-target path. This method relies on a protocol for QoS session establishment, followed by distributed monitoring and re-route to non-congested regions. Experimental results demonstrate the impact on multimedia flows with fixed and variable packet sizes (from real traffic traces) in the buffer sizing, and the percentage of deadline violations as a function of the *D-buffer* size. In terms of adaptive routing, the obtained results present the influence of different levels of traffic locality on packets latency, NoC occupation and adaptive routing reactivity to congestion events.

Keywords: networks on chip, buffer sizing, congestion, source routing, quality of service, adaptive routing, monitoring.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

APCG	*Application Communication Graph*
BARP	*Balanced Adaptive Routing Protocol*
BE	*Best Effort*
BOM	*Begin Of Message*
CARS	*Contention Aware Routing Selection*
CBR	*Constant Bit Rate*
CC	*Channel Controller*
CCBE	*Congestion Controlled Best Effort*
CI	Circuito Integrado
CMP	*Chip Multiprocessor*
CS	*Congestion Status*
DyAD	*Dynamic Adaptive Deterministic*
DyXY	*Dynamic XY*
EOM	*End Of Message*
FDM	*Frequency Division Multiplexing*
FPGA	*Field Programmable Gate Array*
GPC	Gerador de Pacotes de Controle
GT	*Guaranteed Throughput*
IP	*Intellectual Property*
IMA	*Inter Messages Arrival*
ITRS	*International Technology Roadmap of Semiconductors*
KR	*Known Rate*
LTE	*Long Term Evolution*
LUT	*Look Up Table*
Mbps	*Megabits per second*
MC	Métrica de Congestionamento
MMP	*Markov Modulated Process*
MPC	*Model Predictive Control*
MPSoC	*Multiprocessor System on a Chip*
MSA	*Monitoring Service Access*
NF	*Negative First*
NI	*Network Interface*
NL	*North Last*
NoC	*Network-on-Chip*
OFDMA	*Orthogonal Frequency Division Multiple Access*
PE	*Probe* Externo
PI	*Probe* Interno
PLR	*Path Load Routing*
PT	*Packet Type*
PTS	*Path To Source*
PTT	*Path To Target*
QoS	*Quality of Service*
RA	*Request/Acknoledge*
RCT	*Router Congestion Table*
SCT	*Source Congestion Table*
SoC	*System on a Chip*
SR	*Source Routing*
SREQ	*Session Request*
TCT	*Target Congestion Table*
TTI	*Time Transmission Interval*
UR	*Unknown Rate*
VC	*Virtual Channel*
VCB	*Virtual Channel Buffer*
WF	*West First*

SUMÁRIO

1. INTRODUÇÃO

Uma característica observada nos últimos anos em relação a dispositivos eletrônicos diz respeito ao aumento da capacidade de armazenamento em um único chip. Este fato confirma a previsão da conhecida “Lei de Moore” [MOO65], onde se afirma a duplicação da capacidade de integração de transistores a cada dois anos. A Figura 1.1 ilustra a concretização deste crescimento para processadores Intel. Ainda, dados da ITRS (*International Technology Roadmap for Semiconductors*) apontam que, para um futuro próximo, os CIs (Circuitos Integrados) irão operar com freqüências em torno de 4.7GHz e dimensão de aproximadamente 50nm, contendo um número de transistores na ordem de bilhão. Esta estimativa é confirmada no artigo apresentado em [ITR09]. Segundo o artigo apresentado em [INT10], o processador *Tukwila*, com dois bilhões de transistores, começará a ser produzido em 2010. Entretanto, a evolução da tecnologia está superando a própria lei de Moore, levando ao novo termo *“more than Moore”* [IEE10]. Confirmando esta tendência de evolução tecnológica, a fornecedora de dispositivos lógicos programáveis *Xilinx* apresenta em [XIL10] o lançamento de uma família de plataformas com tecnologia de 28nm.

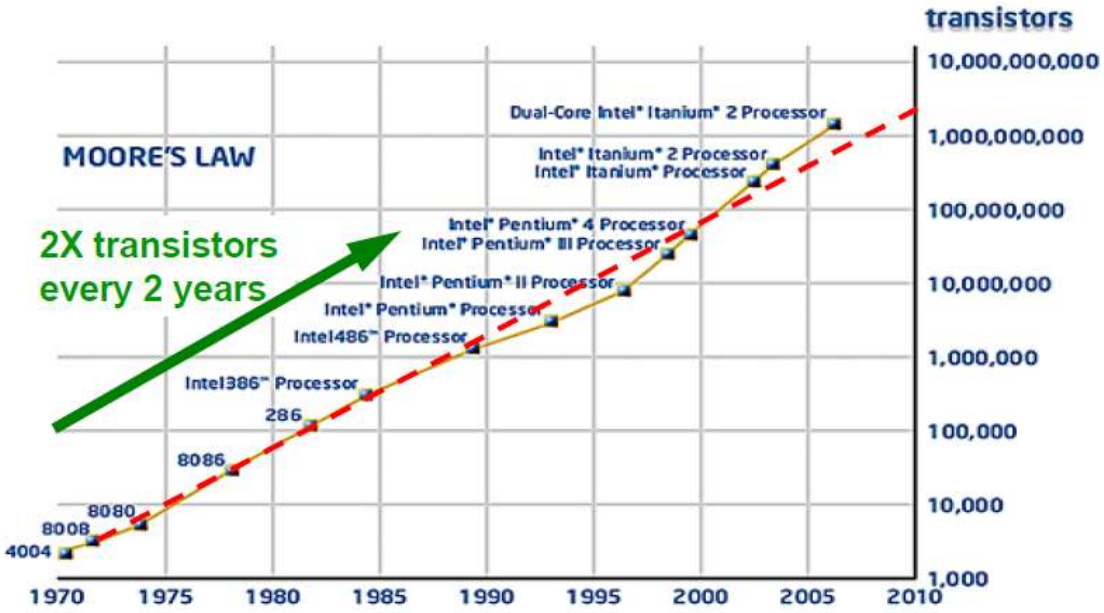


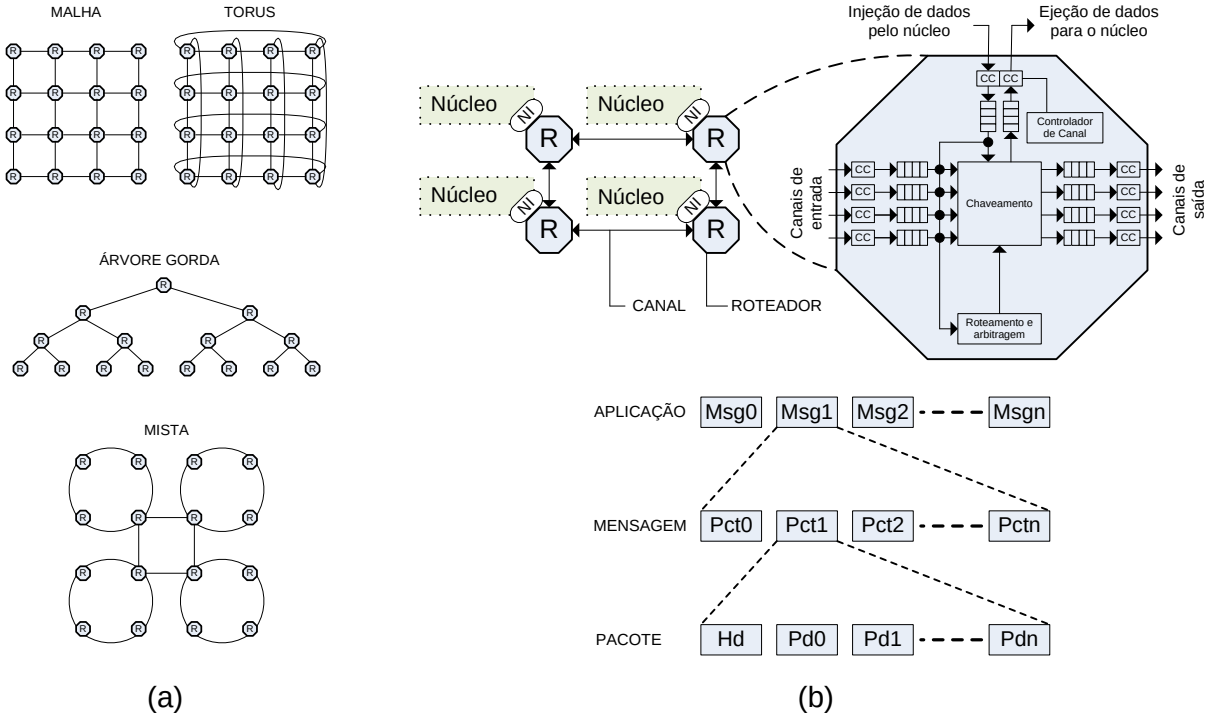
Figura 1.1 – Aumento da capacidade em processadores Intel [GHA09].

Tal avanço na tecnologia de fabricação de circuitos integrados proporcionou o advento do conceito de SoC (*System-On-Chip*) [MAR01], o qual designa um sistema computacional completo em um única pastilha de silício, podendo conter processadores, estruturas de comunicação e elementos de armazenamento. Devido à alta complexidade inerente ao projeto de um SoC, a sua construção baseia-se no reuso de componentes pré-projetados e pré-validados que são denominados *núcleos de propriedade intelectual*, ou

simplesmente núcleos IP (*Intellectual Property*), possibilitando, desta forma, o lançamento de um sistema embarcado dentro de sua janela de *time-to-market*. Com a especificidade e elevada quantidade de aplicações dos sistemas embarcados atuais, é conveniente a construção dos mesmos utilizando o conceito de MPSoC (*Multi Processor System On Chip*). Um MPSoC constitui uma arquitetura composta por elementos de processamento (PEs) que devem atender a requisitos de uma dada aplicação ou classe de aplicações [JER05]. Estes elementos comunicam-se entre si em alta velocidade, o que faz com que um MPSoC possua requisitos rígidos de comunicação que podem não ser atendidos por estruturas baseadas em fios dedicados, barramentos simples ou barramentos hierárquicos.

Um conceito alternativo de comunicação em MPSoCs em contraposição aos fios dedicados e barramentos é o de redes intra-chip (NoCs, do inglês *Networks-on-Chip*) [PAS08]. As NoCs vêm sendo estudadas como uma promissora alternativa para interconexão de núcleos IP em sistemas embarcados, utilizando para isso conceitos da área de redes de computadores e comunicação de dados. Dentre as vantagens oferecidas por esta estrutura de comunicação, relacionam-se: (*i*) paralelismo de comunicação entre pares distintos de núcleos; (*ii*) compartilhamento de fios, pelo fato da largura de banda ser escalável, comportando fluxos concorrentes [BOL04][DAL01][GOO02][GUE00]; (*iii*) possibilita priorizar fluxos com requisitos de QoS (*Quality of Service*); (*iv*) possibilita a implementação de mecanismos de confiabilidade e gerência do consumo de energia [BEN01][BOL04]; (*v*) reusabilidade.

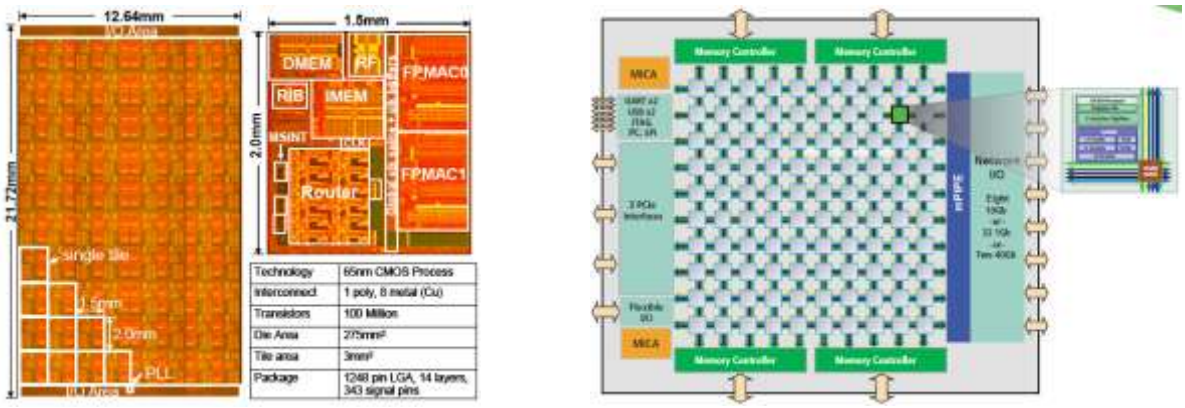
Em uma arquitetura de comunicação baseada em NoC, os núcleos IP são conectados aos *roteadores* através de *NIs* (*Interfaces de Rede*). A forma na qual os roteadores são interconectados entre si define a *topologia* da rede, como a topologia malha apresentada na Figura 1.2(a). A Figura 1.2(b) detalha os principais componentes de uma NoC. As aplicações executando nos núcleos trocam *mensagens* entre si, mensagens estas que são segmentadas em *pacotes* (Figura 1.2(b)). Os pacotes por sua vez são divididos em *flits* (*flow control units*), sendo estes *flits* header (Hd - que pode conter, por exemplo, o destino do pacote e dados de controle) ou payload (Pd – que contém dados úteis à aplicação). Os pacotes devem possuir um método para indicar o seu final. Três métodos podem ser empregados: o número de *flits* é indicado no header, e o roteador faz o controle; *flit* de término de pacote; sinal no enlace para indicar este término. No contexto deste trabalho este método é abstraído. No interior dos roteadores, *controladores de canal* (CC) coordenam as entradas e saídas dos *flits*. *Buffers* armazenam temporariamente dados ainda não encaminhados. O módulo de *chaveamento* conecta uma porta de entrada a uma porta de saída. O módulo de *roteamento e arbitragem* executa o algoritmo que define as conexões entre entradas e saídas. Objetiva-se com a utilização do NoCs o aumento do grau de paralelismo e escalabilidade das comunicações, em comparação com as arquiteturas anteriormente citadas. Por outro lado, a complexidade de projeto, a alta latência média dos pacotes e a considerável área de silício ocupada, constituem as principais desvantagens e desafios no projeto de arquitetura de NoCs.



(a) (b)

Figura 1.2 – (a) Exemplos de topologias empregadas em NoCs; (b) Componentes de uma NoC.

MPSoCs já vêm sendo construídos com arquitetura de comunicação baseada em NoC, sendo dois exemplos ilustrados na Figura 1.3. Um dos projetos atualmente desenvolvidos pela Intel [INT10a] é um MPSoC com 80 núcleos, interligados por uma NoC com topologia malha 8x10 (Figura 1.3(a)). Cada núcleo possui duas unidades para cálculos em ponto flutuante. Este chip também tem como objetivo obter baixo consumo de energia, oferecendo suporte a técnicas como *sleep transitors*, relógio mesócrono e *clock gating*. Os processadores da família *TILE-Gx*, ilustrados na Figura 1.3(b), utilizam de 16 a 100 núcleos [TIL10]. Cada núcleo inclui unidades para suporte a DSP e SIMD, habilitando no mesmo dispositivo tanto instruções multmídia, quanto de propósito geral. Algumas das aplicações alvo desta arquitetura estão relacionadas a redes (processamento de pacotes), *wireless* (controle em estações de base), multimídia (processamento de vídeo) e *cloud computing* (serviços web).



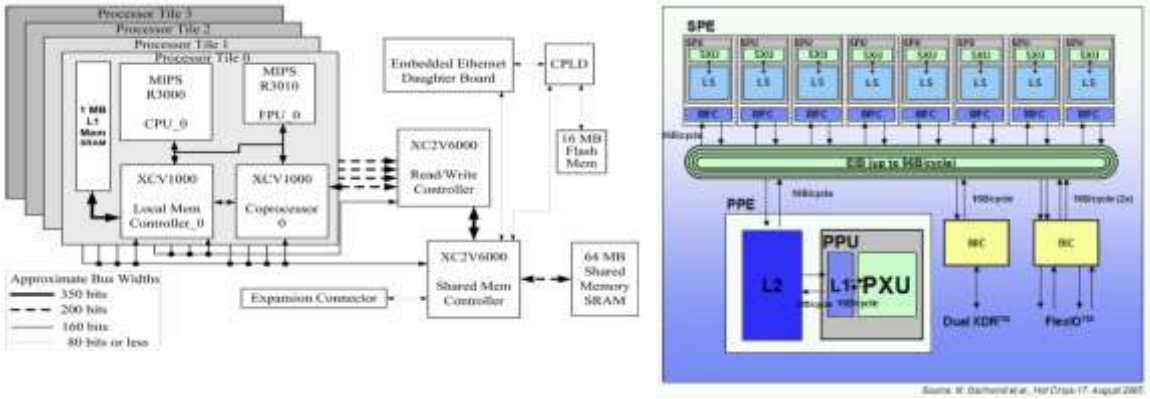
(a) projeto *Teraflops Research Chip*, da Intel [INT10a]

(b) processador da família *TILE-Gx*, desenvolvido pela Tilera [TIL10]

Figura 1.3 – Exemplos de MPSoCs que utilizam comunicação baseada em NoCs.

1.1 ADAPTABILIDADE DA COMUNICAÇÃO EM MPSOCS

Esta Seção apresenta métodos e trabalhos relacionados à adaptabilidade da comunicação em MPSoCs baseados em NoCs, para posterior posicionamento da presente Tese. Como mostrado nas propostas de processadores com múltiplos núcleos, as movimentações de dados originadas pelas aplicações-alvo em MPSoCs podem incluir transferências de dados entre núcleos e memórias, além de transmissão de streams (rajadas de dados) de aplicações multimídia. A Figura 1.4 ilustra exemplos de arquiteturas de MPSoCs, sendo possível observar a existência de diferentes tipos de núcleos de processamento, memórias, interfaces e controladores. É desejável que a estrutura de comunicação tenha condições de tratar a heterogeneidade de padrões de tráfego apresentada pelas diversas aplicações que executam em MPSoCs. De forma genérica, pode-se classificar o tráfego em duas categorias [LOG04][SOT06]: *estático* e *dinâmico*. No tráfego *estático*, uma dada aplicação gera dados a intervalos periódicos de tempo, como ocorre, por exemplo, em um decodificador MPEG. O tráfego *dinâmico* é produzido por uma aplicação que gera dados em momentos aleatórios, ocorrendo, por exemplo, na comunicação entre módulos de processamento e memórias cache, em migração de tarefas ou em transmissão de dados de sinalização.



(a) *FAST* [DAV05] (b) *Cell Processor* [IBM10]

Figura 1.4 – Exemplos de MPSoCs e seus componentes.

A literatura na área de NoCs apresenta diversas propostas de adaptação da rede para atender aos requisitos de aplicações. Estas soluções normalmente objetivam o atendimento de requisitos de níveis de QoS nas comunicações entre os núcleos (latência mínima, *jitter*, *throughput* e *deadlines*), gerenciamento de área de silício utilizada e energia consumida pela rede. Segundo [CAR09], a adaptabilidade em NoCs pode ocorrer em dois níveis: *sistema* e *arquitetura* (Figura 1.5). A adaptação em nível de *sistema* ocorre em tempo de execução, onde parâmetros funcionais são alterados de acordo com as características de tráfego da aplicação em curso, que pode por sua vez apresentar características de dinamicidade. A adaptação em nível de *arquitetura* ocorre em tempo de projeto, onde parâmetros estruturais são dimensionados de acordo com a descrição da aplicação-alvo. A seguir são apresentadas soluções de adaptabilidade aplicadas em arquiteturas de comunicação baseadas em NoCs, e trabalhos relacionados.

Figura 1.5 – Adaptação em nível de sistema e de arquitetura.

A aplicação de um MPSoC é tipicamente divida em tarefas, sendo essas atribuídas a processadores. Este processo de atribuição é chamado de *mapeamento*. Um grafo é uma possível abstração utilizada para a especificação da aplicação, cujos vértices constituem as tarefas e as arestas especificam características de suas comunicações. O mapeamento de tarefas pode ser *estático* ou *dinâmico*. O mapeamento estático é realizado em tempo de projeto, objetivando principalmente a redução do consumo de energia e área. Os autores [HU03], [MUR05] e [MAR05] propõem algoritmos que analisam as propriedades das aplicações descritas em grafos e escolhem melhor o posicionamento das tarefas na rede. O mapeamento dinâmico é definido em tempo de execução, em função de parâmetros tais como a carga do sistema, estado de congestionamento na interface de comunicação dos roteadores. [CAR09a] investiga o desempenho de heurísticas para mapeamento dinâmico de tarefas em MPSoCs heterogêneos, com o objetivo de minimizar congestionamentos na NoC. As tarefas são mapeadas sob demanda, de acordo com as requisições de comunicação e com a ocupação de canais.

Outra técnica de adaptação é a *migração de tarefas*, onde é realizada a transferência de tarefas entre os elementos de processamento. É necessário, entretanto, que a execução da tarefa no processador original seja interrompida, e o salvamento de contexto para que o mesmo seja retomado quando a tarefa reiniciar no processador substituto. [BAR07] propõe um esquema híbrido de migração de tarefas em MPSoCs baseados em NoCs, onde o código da aplicação pode ser migrado do nodo original onde a tarefa executa, ou a partir da memória global do sistema, sendo esta escolha realizada em tempo de execução, com base na distância entre os nodos envolvidos.

O *controle da inserção de pacotes* regula a quantidade de dados presente na rede, de modo a evitar congestionamentos futuros. Em [OGR08] é proposto um controle de fluxo

baseado na predição de ocorrência de congestionamento na NoC. Cada roteador avalia a sua capacidade de atendimento dos pacotes que estão chegando através da medição em tempo de execução da ocupação média de sua fila de armazenamento e do histórico de taxa de chegada dos pacotes. Estes dados são então utilizados para avaliar a possibilidade de ocorrência de congestionamento. Por fim a probabilidade de congestionamento é enviada para os outros roteadores da rede de modo a regular a sua inserção de pacotes. O trabalho desenvolvido em [MAN06] traz como enfoque a análise do tempo em que comunicações acontecem em conjunto com a demanda de ocupação de *buffers* em roteadores em um pior caso. Restrições de latência das aplicações e estimativas de descarte de pacotes são os parâmetros utilizados. Eventos de comunicação são mapeados na rede em tempo de projeto e o envio de pacotes é atrasado na fonte, com o objetivo de evitar concorrência entre diferentes fluxos.

A utilização de conexões com *freqüência de transmissão variável* é uma abordagem adotada para redução do consumo de energia em NoCs. [LEE09] propõe uma estratégia onde a freqüência de transmissão é regulada de acordo com a taxa de transmissão de dados. É utilizada ainda a técnica de *clock boosting*, onde diferentes freqüências são utilizadas para a transmissão de *flits* de *header* e *payload*.

A execução de *algoritmos de roteamento adaptativos* define os caminhos a serem percorridos pelos pacotes, considerando o estado de congestionamento da rede. O objetivo é desviar o fluxo que está sendo transmitido de regiões conhecidas como *hotspots*, que são pontos na rede onde ocorre maior dificuldade no encaminhamento dos dados. A decisão do caminho a ser tomado pode considerar a ocupação de um ou mais roteadores, o que define o escopo da observabilidade do tráfego: local ou global. [HU04] adota roteamento distribuído, sendo considerado em cada *hop* a ocupação de *buffers* em roteadores vizinhos para decidir se vai configurar o roteador corrente para atuar de maneira adaptativa ou determinística. Por outro lado, [LOF08] avalia roteadores em regiões críticas, que enviam o seu estado de congestionamento aos geradores de tráfego, de modo a orientar as suas decisões de roteamento. [ALF08] propõe ainda uma abordagem que considera a largura de banda disponível no caminho restante entre o roteador atual e o destino.

A *configuração da topologia* é adotada em [DUM09] e [OGR06]. [DUM09] incorpora ao processo de geração de topologia resultados de cálculos matemáticos, onde é estimado o grau de contenção de dados nos canais da rede. O método baseia-se no modelo de comunicações normalmente encontrado em NoCs, permitindo que sejam geradas topologias otimizadas para aplicações específicas, constituídas somente com os recursos que serão realmente utilizados. Embora o método seja executado em tempo de projeto, ele procura prever eventos dinâmicos na rede. [OGR06] propõe a geração de topologias híbridas, através da inserção de fios para ligação de pares origem-destino não diretamente conectados pela rede. A partir de uma topologia malha regular, poucas conexões são seletivamente adicionadas com o objetivo de reduzir a latência dos pacotes. Este trabalho também propõe um algoritmo de roteamento centralizado, que procura explorar o potencial dos caminhos oferecidos pelos fios adicionados, de forma a atingir o nível de desempenho especificado em tempo de projeto.

A *multiplexação de um canal físico em canais virtuais* é uma técnica onde é compartilhado o uso de um canal físico por múltiplos fluxos. Sem canais virtuais, os pacotes que ganham o roteamento acabam bloqueando a porta de saída correspondente para fluxos concorrentes que queiram a mesma porta. Com canais virtuais, fatias de tempo para utilização do canal físico são atribuídas aos fluxos. A estratégia mais comumente utilizada para gerenciamento da multiplexação de largura de banda é a definição do número de canais virtuais a serem utilizados. Se forem definidos dois canais virtuais, 50% de tempo de uso do canal físico será atribuída para cada fluxo concorrente, por exemplo. É possível ainda a associação dos fluxos das aplicações no nível de serviço. Neste caso, é possível a priorização de pacotes associados a um determinado canal virtual, preemptando-se pacotes de outros fluxos. Na estrutura QNoC [BOL04] o tráfego é dividido em classes de serviços, que possuem prioridades diferentes entre si. Desta forma, sempre que o pacote com fluxo de maior prioridade solicita roteamento, os pacotes de classes menos prioritárias são preemptados. Na NoC *AEthereal* [GOO05] uma aplicação pode ser associada a um nível de serviço GT (*Guaranteed Throughput*) ou BE (*Best Effort*). No nível GT, os pacotes possuem maior prioridade, com taxa de transmissão garantida pela técnica *circuit-switching*. [MEL07] define prioridades dinâmicas, sendo que em cada roteador por onde passa um pacote, é avaliado se a sua taxa de saída está em conformidade com um valor contratado no estabelecimento da sessão. Caso não esteja, a prioridade muda de valor.

Uma alternativa considerada mais simples em comparação à multiplexação de canais físicos é a *replicação de canais físicos*. O objetivo é atender a requisitos *hard-real-time* de aplicações, onde a latência e o *jitter* (variação instantânea da latência) devem ser mínimos. Em [BOU06] é proposta uma NoC com alta vazão baseada em uma topologia de árvore gorda. Os roteadores possuem os canais físicos replicados de maneira que existam mais portas de saída do que de entrada. Tendo como estratégia a eliminação da contenção, a latência é reduzida e a vazão é maximizada. Não existindo contenção, elimina-se também a necessidade de *buffers* internos. [SET06] propõe a redução do consumo de área em NoCs reduzindo o número de roteadores. O objetivo é atingido através da replicação da porta local, tornando possível conectar mais IPs utilizando menos roteadores. [CAR08] mostra os benefícios da utilização da replicação de canais, em comparação com estratégias baseadas em canais virtuais e chaveamento de circuitos.

A *configuração de tamanhos de buffer* define a profundidade de *buffers* dos roteadores dos roteadores e das interfaces externas da rede. Esta técnica normalmente recebe uma rede com os núcleos da aplicação já posicionados. A partir desta rede de base podem ser realizadas verificações da utilização de *buffers*. Tal avaliação indica em que pontos os *buffers* devem ser redimensionados. [VAR04] verifica a auto-similaridade de um tráfego de vídeo para dimensionamento de *buffers* de uma NoC que interliga módulos de um decodificador MPEG. [HU06] propõe um algoritmo que otimiza a alocação de *buffers* em tempo de projeto, de acordo com as características do tráfego da aplicação. [NIC06] utiliza uma estrutura de *buffers* unificada, o que habilita a cada roteador ter um número distinto de canais virtuais em cada porta de entrada.

A utilização de enlaces *bi-direcionais* é proposta em [LAN09]. Nesta estratégia, cada enlace de comunicação pode ser dinamicamente auto-configurado para transmitir *flits* em qualquer direção, obtendo-se com isso, melhor utilização de recursos em chip. Em relação a roteadores tradicionais, uma lógica de controle adicional é inserida em cada roteador, de modo a definir se o enlace de dados correspondente vai ser transmissor ou receptor de *flits* em um dado instante.

Neste contexto de adaptabilidade, o foco da presente Tese reside na otimização do desempenho em fluxos de dados entre pares de núcleos que se comunicam através de uma NoC, com requisitos de qualidade de serviço. Tal otimização utiliza um sistema de monitoração de tráfego em roteadores pertencentes ao caminho de um fluxo com requisitos de QoS, que oferece suporte a um algoritmo de roteamento adaptativo na origem. O objetivo geral é tratar em tempo de execução a ocorrência de congestionamento, gerado por comunicações que tipicamente ocorrem em MPSoCs. Fluxos QoS são caracterizados pela transmissão de dados de aplicações típicas dos sistemas embarcados atuais, como multimídia, fluxo de símbolos (padrões de protocolos de telecomunicações) acessos a memórias e sinalização. A comunicação desempenhada por estas aplicações normalmente possuem restrições em parâmetros como latência, vazão, *jitter* e prazos de entrega. Ainda, a configuração de *buffers* de interfaces de redes externas é abordada, onde é realizado o desacoplamento comunicação-computação.

1.2 VISÃO GERAL DO TRABALHO DESENVOLVIDO

Para atender a adaptabilidade, tanto em tempo de projeto quanto em tempo de execução é proposto um sistema com monitoração de tráfego, roteamento adaptativo na origem e *buffers* de desacoplamento (*D-buffers*) dimensionados de acordo com padrões de produção de dados pela rede e consumo pela aplicação. A Figura 1.6 apresenta uma visão geral do trabalho desenvolvido, sendo ilustradas as etapas no projeto de NoCs que serão abordadas: geração de tráfego, definição de caminho, desacoplamento comunicação-computação, coleta e propagação de informações de congestionamento, detecção de congestionamento, envio de pacote com informações de congestionamento e definição de nova rota.

As etapas 1 e 2 mostram a geração de tráfego de um IP origem para um IP destino. O objetivo é estruturar os dados da aplicação no formato de pacotes, de maneira a viabilizar a sua transmissão no meio de comunicação. Inicialmente os dados da aplicação são quebrados em mensagens, sendo posteriormente formatados em pacotes. Todo pacote é formado por *flits* de *header* e *payload*. O *header* (*Hd* na figura) contém informações para execução do protocolo de comunicação, além de indicar o destino dos dados através do campo PPT (*path to target*). De fato, o campo PTT define a distribuição espacial do tráfego, onde são especificadas as portas de saída em cada roteador pertencente ao caminho do fluxo. O *payload* (Pd) contém informações específicas da aplicação ao qual o fluxo pertence. A taxa de injeção de dados é outro importante parâmetro na etapa de geração de tráfego, onde se especifica a freqüência na qual se inserem dados na rede. Taxas típicas de aplicações são utilizadas como referência, trazendo aos experimentos maior possibilidade de se avaliar a utilização de recursos em

cenários cujo comportamento seja similar a situações reais. Quatro classes de pacotes são transmitidas sobre a arquitetura-alvo proposta. Pacotes do tipo SREQ iniciam uma sessão QoS, configurando os roteadores intermediários para o armazenamento de informações sobre o fluxo. Pacotes do tipo DATA são aqueles que carregam informações da aplicação. Ainda, a funcionalidade deste pacote é estendida para conter dados de gerados pelo processo de monitoração da rede. Pacotes do tipo ALARM informam quais roteadores estão congestionados, além de habilitarem a transmissão de mensagens. Pacotes do tipo CLEAN limpam registros de fluxos QoS em roteadores após a transmissão de mensagens ou devido a falhas de inicialização de caminhos.

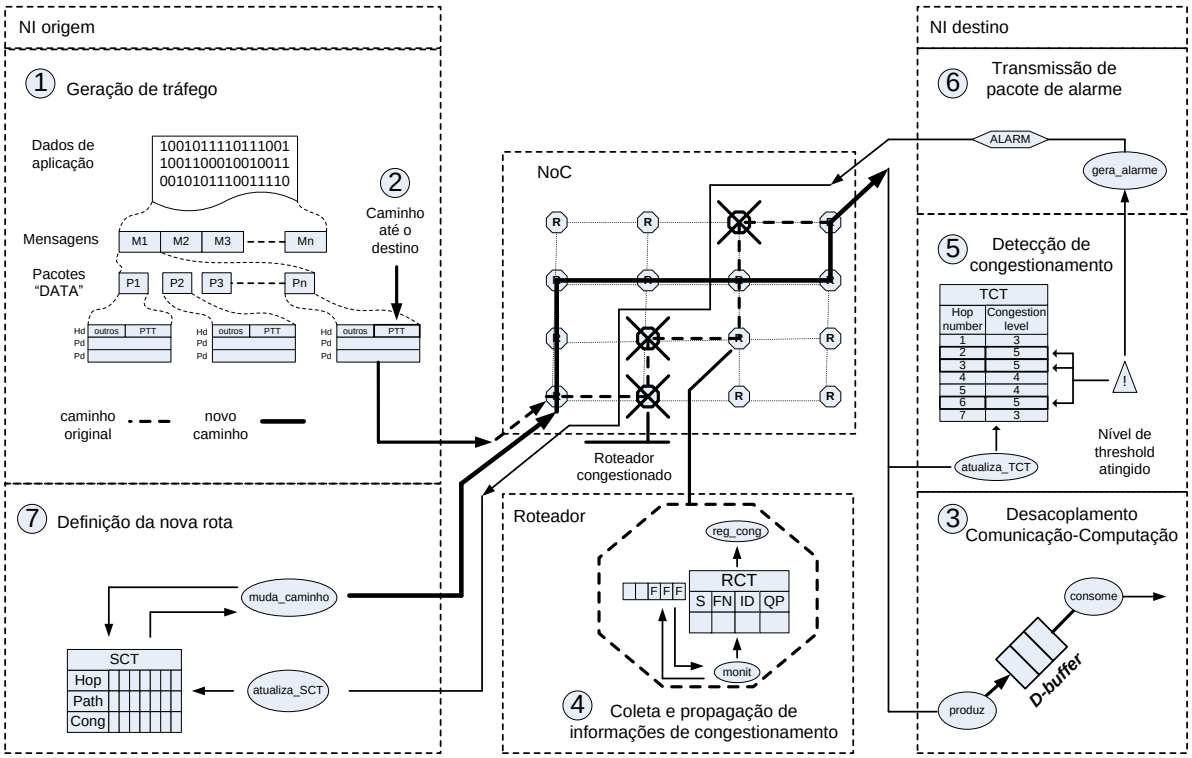


Figura 1.6 – Visão geral da proposta de trabalho.

Em 3 é ilustrado o processo de desacoplamento comunicação-computação, em que os pacotes que chegam na taxa de transmissão da rede são armazenados em um *buffer* (tratado aqui como *D-buffer*) para serem posteriormente consumidos pelo núcleo IP receptor na taxa da aplicação. Portanto, o *D-buffer* tem como função adaptar os eventos que ocorreram na rede à demanda de processamento pelo núcleo IP. O empacotamento de dados na origem, o processamento de roteamento e a concorrência de outros fluxos que utilizam a NoC provocam a descaracterização do fluxo original dos dados. Neste trabalho é mostrado como é dimensionado o *D-buffer* pertencente às NIs de módulos IP, de modo que sejam evitadas ocorrências de falta de espaço para inserção de novos dados que chegam, assim como ocorrências de esvaziamento precoce do *D-buffer*, o que faz como que a aplicação não consiga consumir dados no momento em que desejar.

A parte 4 da figura ilustra o processo monitoração de tráfego na rede. Nesta etapa é observado o estado de utilização de recursos da rede. Ocupação de *buffers* e taxa de

encaminhamento de pacotes são algumas das métricas que podem ser observadas. A Tabela RCT (*Router Congestion Table*) armazena informações monitoradas para cada fluxo que utiliza o roteador. A mesma tabela é utilizada no momento em que a informação deve ser enviada ao destino, informando o seu grau de congestionamento.

A etapa 5 ilustra a detecção de um valor de *threshold* (limite) atingido, indicando a necessidade de haver mudança na rota dos pacotes das próximas mensagens, visto que congestionamento em um dado ponto do caminho foi detectado. A tabela TCT (*Target Congestion Table*) armazena para cada fluxo QoS o nível de congestionamento encontrado em cada roteador de seu caminho. Em 6 é ilustrada a transmissão de um pacote ALARM, o qual notifica congestionamento, informando quais roteadores estão com dificuldades para encaminhamento de pacotes.

Em 7 é mostrado no IP origem do tráfego a redefinição da rota de um pacote gerado na etapa 1. Tal redefinição toma como referência o dado contido no pacote ALARM recebido. Uma vez que os dados são extraídos do pacote, eles são armazenados na tabela SCT (*Source Congestion Table*), a qual é consultada pela função que gera o novo caminho. Assim sendo, o próximo pacote é transmitido em um novo caminho, que evita *hot-spots* gerados pelos tráfegos concorrentes.

1.3 OBJETIVOS

Este trabalho possui como objetivos estratégicos:

- Dominar os conceitos de geração de tráfego para NoCs, inicialmente estabelecidos em [TED05], no que se refere à utilização de *benchmarks* de aplicações reais;
- Compreender os problemas decorrentes de congestionamento de tráfego gerados por aplicações que executam em MPSoCs;
- Dominar e propor métodos para monitoração de tráfego em NoCs;
- Dominar e propor métodos para controle de congestionamento em NoCs através da aplicação de algoritmos de roteamento adaptativos.

Assumem-se como objetivos específicos:

- Desenvolver um método de geração de tráfego genérico, de modo a considerar aspectos particulares de aplicações que tipicamente executam sobre MPSoCs;
- Desenvolver um método para dimensionamento de *buffers* de Interfaces de Rede (NIs) de núcleos IP, de modo a minimizar os efeitos de *jitter* produzidos por eventos que ocorrem na rede;
- Desenvolver um algoritmo de roteamento adaptativo para a NoC Hermes [MOR04], para aumento do desempenho de aplicações que apresentam tráfego dinâmico em conjunto com tráfego estático;

- Propor um método de monitoração de tráfego realizado nos roteadores na rede e nas interfaces externas, de modo a orientar a adaptabilidade do algoritmo de roteamento a ser proposto;
- Estudar *benchmarks* de aplicações típicas de MPSoCs.

Busca-se na presente Tese provar que o controle de congestionamento pode ser realizado através de roteamento adaptativo na origem com monitoração do tráfego ao longo do caminho origem-destino.

1.4 CONTRIBUIÇÕES ORIGINAIS DA TESE

O presente trabalho apresenta contribuições nas áreas de geração de tráfego, desenvolvimento de interfaces para módulos de recepção de tráfego, monitoração e implementação de algoritmos de roteamento adaptativos.

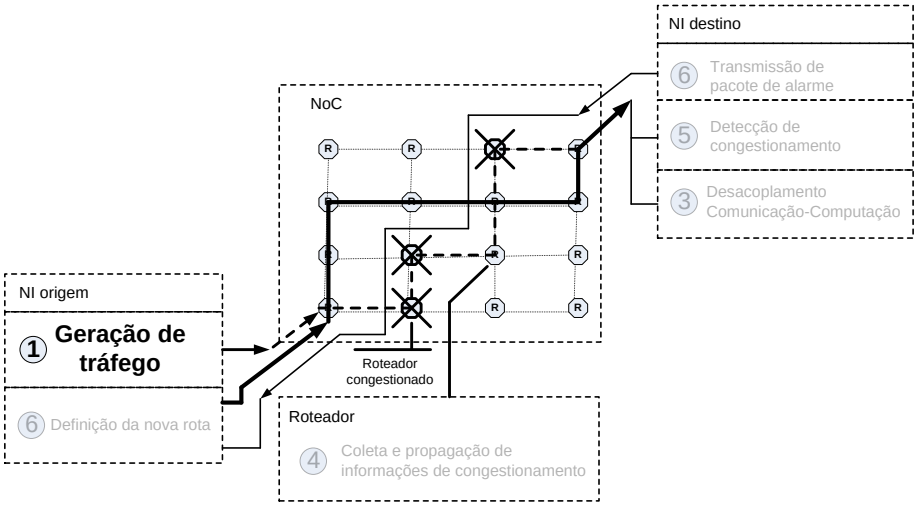
As contribuições deste trabalho no que se refere à geração de tráfego compreendem um método para modelagem de tráfego para aplicações com tráfego estático e dinâmico e a integração de *benchmarks* de aplicações típicas de MPSoCs ao método de geração de tráfego proposto. Em relação ao desenvolvimento de interfaces de rede para NoCs é proposto o desenvolvimento de um método para dimensionamento de *buffers* pertencentes a interfaces de rede de módulos receptores de tráfego QoS, considerando eventos que ocorrem durante a operação da NoC.

Em relação à monitoração de tráfego é proposto o desenvolvimento de monitores que consideram a análise de tráfego que ocorre tanto nos roteadores quanto nas interfaces externas que interligam módulos IP à rede em tempo de execução das aplicações. Finalmente, na parte de algoritmos de roteamento é proposto o desenvolvimento de um algoritmo de roteamento adaptativo que considera situações de tráfego que ocorrem no caminho de fluxos QoS. O método tem por objetivo reagir de forma rápida aos eventos de congestionamento, adotando caminhos alternativos durante a execução das aplicações. Esta abordagem tomará decisões não somente com base em informações de roteadores vizinhos, como acontece em [HU04][DAN06][SAB07], mas também no estado dos roteadores ao longo do caminho de fluxos QoS. Desta forma objetiva-se uma visão global do estado da rede, de modo que o tráfego alvo percorrer caminhos menos congestionados. Resultados obtidos durante a execução deste trabalho apontam a chegada de pacotes com latência mínima após a detecção de pontos de congestionamento produzidos por tráfegos altamente localizados. Uma melhor utilização da rede também foi obtida, através da utilização de caminhos alternativos.

1.5 ORGANIZAÇÃO DO DOCUMENTO

Cada Capítulo corresponde a uma etapa mostrada na Figura 1.6. O Capítulo 2 é relacionado à etapa 1 e 2 da Figura 1.6, e trata do método de geração de tráfego proposto. O método proposto para dimensionamento de *buffers* das interfaces de rede externas é assunto do Capítulo 3, sendo relacionado à etapa 3 da Figura 1.6. A monitoração de tráfego é assunto do Capítulo 4, sendo relacionado às partes 4, 5 e 6 da Figura 1.6, sendo ainda detalhada a arquitetura do roteador proposto e a sua validação. O Capítulo 5 apresenta o método proposto para roteamento adaptativo na fonte adotado para fluxos QoS, mostrado na etapa 7. O Capítulo 6 apresenta experimentos envolvendo a geração de tráfego, o dimensionamento e operação de *D-buffers* e a avaliação de parâmetros de QoS com tráfegos sintéticos e reais. O Capítulo 7 apresenta as considerações finais, a lista das publicações relacionadas aos temas deste trabalho, e direções para trabalhos futuros.

2. BENCHMARKS PARA GERAÇÃO DE TRÁFEGO



Um *benchmark* constitui uma estrutura de referência para a caracterização de um projeto de um sistema computacional, considerando aspectos como modelagem da aplicação para avaliação de desempenho e teste de confiabilidade pós-fabricação. Além de revelar os pontos fracos e fortes de uma arquitetura alvo, *benchmarks* também permitem a comparação entre diferentes sistemas. A especificação e/ou utilização de *benchmarks* de aplicações em NoCs é um tema bastante discutido, até porque a própria área de redes de comunicação intra-chip é recente. Em [GRE07] [SAL05] [OGR08] [DAL06] são discutidos problemas de pesquisa para NoCs, onde são abordadas questões ainda não resolvidas, relacionadas à metodologia/tecnologia de projeto, análise arquitetural, estratégias de teste e ferramentas de CAD dedicadas. Os pontos levantados nestas análises mostram que é fundamental a disponibilização de *benchmarks* para o desenvolvimento de estruturas de comunicação em sistemas embarcados.

Como mencionado no início deste trabalho, as NoCs são consideradas a arquitetura de interconexão a ser empregada na implementação de sistemas embarcados com múltiplos elementos de processamento. Conseqüentemente, *benchmarks* tradicionais projetados para um único processador não são aplicáveis. Além disso, espera-se que a natureza das aplicações executadas sobre sistemas baseados em NoCs seja mais variada e heterogênea do que aplicações típicas para computadores multiprocessados [GRE07]. Portanto, existe a necessidade de *benchmarks* especialmente desenvolvidos para o domínio de NoCs, juntamente com métricas e metodologias de medição bem definidos. Existem basicamente duas categorias de *benchmarks*: aplicações e modelos sintéticos.

Os *benchmarks baseados em aplicações* têm suas características de tráfego geradas a partir da implementação de uma aplicação. Aplicações proporcionam uma avaliação precisa do sistema em relação à computação (elementos de processamento) e a comunicação (estrutura de interconexão). Neste caso é possível determinar com precisão o nível de adequação do sistema computacional à aplicação executada. No entanto, essa

abordagem apresenta como desvantagem escalabilidade e portabilidade pobres, visto que são projetadas para sistemas específicos (e.g. processadores e protocolos). O tempo de simulação e o desenvolvimento tendem a ser longos quando comparados a *benchmarks* sintéticos. Além disso, muitas companhias e instituições não têm interesse de contribuir com códigos fonte para novos casos de teste [SAL05]. Ferramentas para simulação de benchmarks baseados em aplicações, como E3S [DIC08] e SoCLib [SOC08] são utilizadas [HU06][SCH07]. Outra proposta é a utilização de modelos do Matlab [MAT08], visto que ele oferece suporte à interação com programas escritos em SystemC [BOL05].

Os *benchmarks sintéticos* por sua vez, empregam algum nível de abstração de onde são extraídas informações que caracterizam o tráfego [GRE07][SAL05]. O emprego deste nível de abstração é realizado através da utilização de geradores de tráfego. A geração de tráfego é a etapa dentro do fluxo de projeto de NoCs onde são especificados os estímulos que testam o funcionamento da rede. As informações dos *benchmarks* são extraídas, e, a partir destes dados, é realizada a geração de tráfego, sendo configurados parâmetros em duas dimensões: *modelagem espacial*, que especifica a relação posicional entre cada iniciador e destinatário de tráfego; *modelagem de taxas de injeção*, que especifica a quantidade de dados injetada em um determinado espaço de tempo.

A geração de tráfego baseado em *benchmarks* sintéticos pode adotar em um ambiente de teste os seguintes tipos de referências: *funções de probabilidade*, sendo que os destinos de tráfego e as taxas de injeção são definidos de acordo com variáveis aleatórias; *grafos*, onde os destinos de tráfego e as taxas de injeção são específicos de uma aplicação; *traces*, onde os destinos de tráfego, as taxas de injeção e os momentos dos eventos de comunicação são específicos de uma aplicação.

Durante o processo de extração de informações para geração de tráfego, somente o comportamento externo dos elementos de processamento (acoplados aos roteadores através de NIs) é relevante. Tal comportamento normalmente é descrito pela taxa de injeção de dados, desconsiderando o valor ou o significado dos dados transmitidos. A abstração da funcionalidade dos elementos de processamento favorece a troca de dados de teste entre companhias e instituições. Em comparação a *benchmarks* baseados em aplicação, a utilização de geradores de tráfego é mais flexível (pelo fato de geradores serem normalmente parametrizáveis), de simples implementação e baixo custo, além de possibilitar menor tempo de simulação em comparação com o desenvolvimento de aplicações.

Neste capítulo são revisados inicialmente a modelagem de taxas de injeção e distribuição espacial. A utilização de traces e grafos é abordada na seqüência. O Capítulo apresenta ainda uma revisão bibliográfica sobre geração de tráfego para NoCs. Ao final é apresentada aspectos fundamentais da aplicação de telecomunicações 3GPP-LTE, cuja execução em uma NoC foi estudada durante o período de estágio de doutorado sanduíche no laboratório CEA-LETI [LET10].

2.1 MODELAGEM DE TAXAS DE INJEÇÃO DE DADOS

A especificação de *taxas de injeção de dados* corresponde à quantidade de informação transmitida por um IP fonte em um dado período de tempo. Neste tipo de modelagem podem ser utilizadas funções matemáticas, grafos ou mesmo *traces* coletados a partir de aplicações reais para descrever parâmetros de tráfego. Devido à maior taxa de transmissão dos canais de uma NoC comparado a taxas de IP individuais, é recomendado o encapsulamento de dados a serem transmitidos em pacotes, em um processo denominado empacotamento (*packaging*). Os dados da aplicação são agrupados em um *buffer* para transmissão de dados. Após a quantidade de dados agrupados ser correspondente ao tamanho do pacote em número de *flits*, os dados são gerados em rajada na taxa de transmissão da rede.

É possível dividir a geração de tráfego baseado em taxas de injeção em duas classes: (i) modelos KR (*Known Rate*), onde as taxas médias de geração de pacotes são conhecidas *a priori*; (ii) UR (*Unknown Rate*), onde a taxa de injeção de dados é desconhecida. A utilização do modelo CBR (*Constant Bit Rate* – taxa de injeção constante) e modelos probabilísticos (como normal e exponencial) auxilia na construção de tráfego pertencente à classe KR. Os processos Bernoulli, ON-OFF e auto-similar são parte da classe UR. A vantagem ao se utilizar a abordagem analítica é a possibilidade de se realizar simulações com uma quantidade reduzida de pacotes, os quais capturam as principais propriedades de uma aplicação, assumindo que o modelo é suficientemente preciso ao manter as mesmas características da aplicação modelada [VAR04]. Por este motivo, a maior parte dos grupos de pesquisa utiliza estas abordagens no processo de geração de tráfego.

2.1.1 MODELOS KR (KNOWN RATE)

A especificação dos modelos KR é baseada em um prévio conhecimento das taxas nas quais os pacotes serão transmitidos. Os pacotes podem ser gerados em uma taxa fixa utilizando o modelo de tráfego CBR (*Constant Bit Rate*) ou com uma taxa variável, utilizando funções de probabilidade, como por exemplo, normal e exponencial.

No modelo CBR, pacotes são gerados em uma taxa fixa ***r***, sendo tanto os tamanhos quanto o intervalo entre dois pacotes consecutivos constantes durante o período de transmissão dos dados. A Figura 2.1 exemplifica o resultado de um processo de geração de tráfego utilizando o modelo CBR.

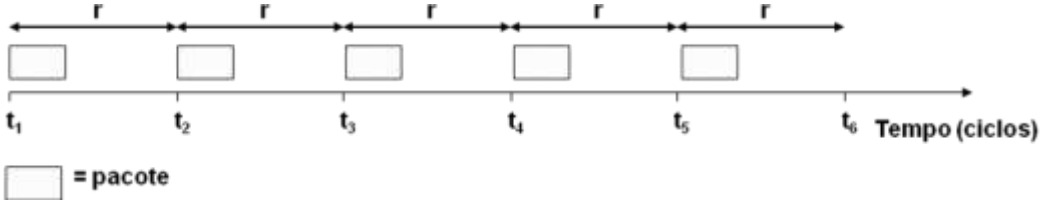


Figura 2.1 – Geração de tráfego utilizando o modelo CBR.

A geração de tráfego quando da utilização do modelo CBR envolve a definição dos valores de dois parâmetros fundamentais. O primeiro deles é o intervalo entre dois pacotes consecutivos, denominado *idle*. Uma vez o valor de *idle* definido, é necessário então calcular o momento em que o pacote deve ser injetado na rede, denominado *pkt_ts*. O parâmetro *pkt_ts* é calculado para cada pacote a ser gerado. Abaixo as equações necessárias para a definição de *idle* (Equação 2.1) e *pkt_ts* (Equação 2.2) [TED05].

$$idle = \left(\frac{chr}{ipr} - 1 \right) * pcksize * ncyclesflit \qquad \text{Equação 2.1}$$

$$pkt\_ts = prvtmp + (pcksize * ncyckesflit) + idle \qquad \text{Equação 2.2}$$

Os parâmetros de entrada para o cálculo de *idle* são: (*i*) *chr*, taxa de transmissão do canal, ou a sua largura de banda; (*ii*) *ipr*, taxa de transmissão do módulo IP; (*iii*) *pcksize*, tamanho do pacote da aplicação, em quantidade de *flits*; (*iv*) *ncyclesflit*, o número de ciclos necessários para transmissão de um *flit*, entre roteadores adjacentes. O valor de *ncyclesflit* vai depender da estratégia de controle de fluxo adotada pelos roteadores que compõem a rede. Se for utilizado o método baseado em créditos, o valor de *ncyclesflit* será 1. Se for utilizado o modo *handshake*, o valor de *ncyclesflit* será 2.

A definição do parâmetro de *pkt_ts* necessita da adoção de uma referência temporal, oferecida por uma variável global que armazena o número de ciclos de relógio ocorridos desde o início da simulação. Ao primeiro pacote atribui-se o valor zero para o seu *pkt_ts*. Para os pacotes seguintes, os valores correspondentes de *pkt_ts* são calculados de acordo com a Equação 2.2, onde, além do tamanho do pacote, do número de ciclos para transmissão de um *flit* e o período entre pacotes, é também considerado o momento ideal de injeção do último pacote enviado, especificado na variável *prvtmp*.

Aplicações como voz digital não-compactada (amostras de 8 bits transmitidas a 64 Kbps), áudio e vídeo não-compactado são exemplos típicos de tráfegos com taxa de injeção constante. Alguns autores sugerem a utilização deste modelo tanto como ruído quanto para caracterização de aplicações [BOL04][HU04][TED05]. A facilidade de implementação constitui a vantagem deste modelo de tráfego, considerando o reduzido número de entradas para o gerador de tráfego definir *pkt_ts*. A principal desvantagem deste modelo é o fato de que aplicações reais normalmente variarem suas taxas de injeção de dados.

A geração de taxas variáveis de injeção de dados nos modelos KR baseia-se em versões discretas de funções probabilísticas contínuas, como normal e exponencial. A Figura 2.2 ilustra um exemplo do resultado de uma geração de tráfego com taxas variáveis, onde $r_1, r_2, r_3, \ldots r_n$ especificam diferentes taxas de injeção.

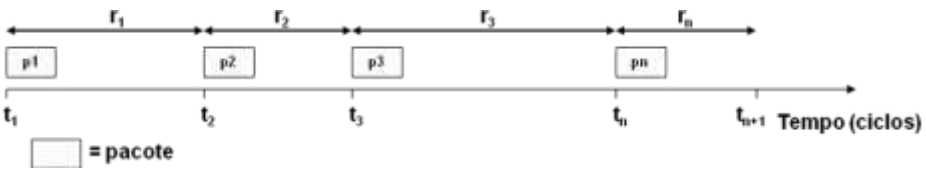


Figura 2.2 – Geração de tráfego com taxas variáveis.

A Figura 2.3 ilustra o processo de geração de tráfego para a classe KR. Dados como entrada alguma distribuição de probabilidade (como exemplo as mostradas em (a)) e a quantidade de pacotes e um conjunto de taxas de transmissão, o processo monta uma tabela de transmissão de pacotes como em (b), indicando o número de pacotes para cada taxa definida. A tabela gerada corresponde à discretização da função probabilística associada. Em (c) um gerador de números aleatórios seleciona a taxa a ser utilizada quando da transmissão de cada pacote e produz seu momento de inserção na rede, de acordo com a utilização das Equações 2.1 e 2.2. Finalmente, o processo preenche uma tabela (em (d)) com o momento de transmissão e o conteúdo de cada pacote a ser enviado. Os processos (b) para (c) e (c) para (d) são repetidos até que todos os pacotes sejam inseridos na tabela (d). A tabela em (d) representa o modelo de tráfego a ser utilizado em simulação ou em emulação.

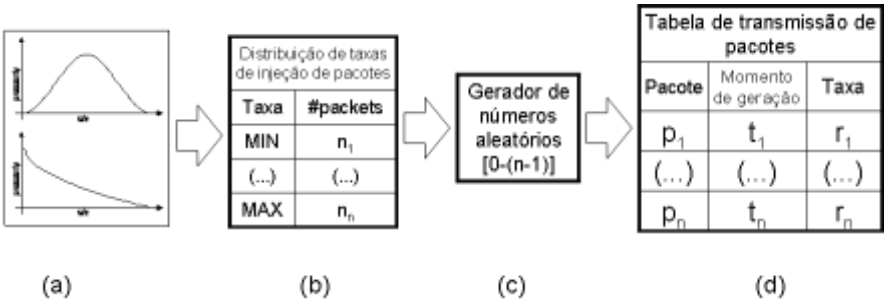


Figura 2.3 – Processo de geração de tráfego para a classe KR.

A taxa global de injeção de dados para os modelos KR são diretamente obtidas pela equação que descreve a distribuição em conjunto com outros parâmetros (por exemplo, desvio padrão). Freqüentemente as entradas para definir o modelo de tráfego são a função de probabilidade, a taxa média de injeção global e um conjunto discreto de taxas.

2.1.2 MODELOS UR (UNKNOWN RATE)

Em contraste com o que acontece na geração de tráfego KR, na classe de modelos de tráfego UR, a geração de pacotes ocorre sob uma estrutura ON-OFF. A estrutura de um processo ON-OFF compreende períodos alternados de atividade de geração de tráfego (em rajadas) e inatividade, como mostrado na Figura 2.4.

Figura 2.4 – Estrutura de um processo ON-OFF.

Considera-se que uma mensagem é constituída por pacotes de tamanho fixo. Durante períodos de atividade, a fonte de tráfego produz pacotes em intervalos regulares

(rajada), até que a mensagem seja completamente transmitida. Em períodos inativos não há geração de pacotes, podendo o IP gerador de tráfego estar processando dados recebidos ou esperando pela chegada de novos dados. O tamanho das rajadas e a duração do período de inatividade podem variar de acordo com valores capturados de *traces*, ou mesmo a partir da computação de funções de probabilidade. Assim sendo, o projetista não sabe a taxa global em que os dados são transmitidos.

Aplicações multimídia, como áudio e vídeo compactados, representam um ambiente de trabalho típico de MPSoCs, podendo ser modelados segundo a classe UR. Trabalhos como [VAR04] e [HU04] descrevem algoritmos de modelagem de tráfego para *streams* de vídeo. Streams são fluxos de dados compostos por mensagens (também conhecidos como *frames*), gerados a intervalos constantes caracterizando um tráfego ON-OFF. Períodos ON correspondem à atividade da geração de tráfego, enquanto que períodos OFF correspondem à inatividade. O tamanho de cada período ON neste caso pode variar, de acordo com a execução de algoritmos de compressão, por exemplo. A taxa durante o período ON é normalmente constante.

A Figura 2.5 e o algoritmo da Figura 2.6 (extraídos de [TED08]) ilustram o processo de geração de tráfego UR (empacotamento) para um *stream* composto em seu início por 2 mensagens com diferentes tamanhos (Figura 2.5(a)). A cada 1,5µs uma nova mensagem deve ser enviada. Assume-se para este exemplo que mensagens são compostas por palavras de 16 bits, com tamanho de *flit* igual ao tamanho da palavra. No exemplo, *Mensagem0* contém 10 *flits* e *Mensagem1* contém 15 *flits*.

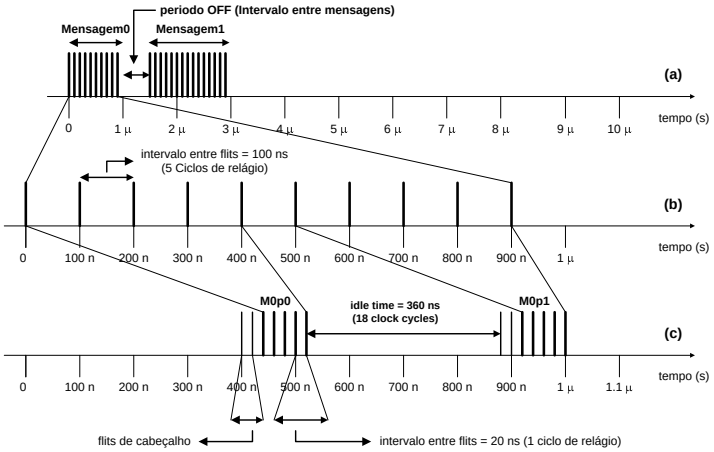


Figura 2.5 – Processo de empacotamento para um *stream* [TED08].

Durante cada período ON, o IP gera pacotes a uma taxa constante. Na figura, a taxa de geração de dados para o *Mensagem0* e *Mensagem1* é de 160 Mbps (10 *flits* de 16 bits transmitidos em 1µs para *Mensagem0* e 15 *flits* de 16 bits transmitidos em 1.5µs, para *frame1*). A Figura 2.5(b) mostra uma visão detalhada do *Mensagem0*. Um *flit* é produzido a cada 100ns. Para uma freqüência de relógio de 50 MHz (período de 20ns), tanto a geração quanto o consumo de *flits* deve ocorrer a cada 5 ciclos de relógio. A Figura 2.5(c) mostra o *Mensagem0* segmentado em dois pacotes, com informações de *header* inclusas.

O algoritmo da Figura 2.6 descreve o processo de geração de tráfego para o modelo ON-OFF, considerando o processo de empacotamento (Figura 2.5(c)). Na linha 1 é feita a atribuição do momento de injeção da primeira *mensagem* na rede (*msg_start_time*) ao parâmetro *msg_ts* (momento de envio da mensagem atual). Este parâmetro baseia-se no contador global, descrito anteriormente. A próxima linha inicia o laço para transmissão de frames, sendo executado *nmsgs* vezes.

O trecho de código entre as linhas 4 e 13 é executado se o tamanho do pacote é fixo (definido pelo valor da variável booleana *packet_size_fixed* – linha 3). O número de pacotes do frame (*npacks*) é obtido pelo resultado do quociente entre o tamanho da mensagem atual (obtido pela execução da função *msgsize* para uma mensagem *i*) e o tamanho do pacote (variável *pcksize*). O laço entre as linhas 5 e 13 injeta um novo pacote, com momento de injeção *pkt_ts* e tamanho *pcksize* em uma FIFO, além de atualizar o valor de *prv_tmp*, para compor o momento injeção do próximo pacote. Esta FIFO é utilizada somente para geração de tráfego. Outro processo lê esta FIFO, injetando o pacote na NoC. Observa-se na linhas 9 e 20 a execução da Equação 2.2, onde é calculado o momento de injeção de um pacote, considerando o momento de injeção do pacote anterior (*prv_tmp*), o tamanho do pacote anterior (*prv_pcksize*), o número de ciclos de relógio para transmitir um *flit* (*ncyclesflit*) e o intervalo ocioso entre pacotes (*idle*). Se a rede está congestionada, o momento de injeção pode ser diferente de *pkt_ts*, aumentando a latência e o *jitter*.

```
1:  msg_ts ← msg_start_time;
2:  for i=0 to nmsgs-1 do    // Número de mensagens
3:    if packet_size_fixed==true then
4:      npacks ← msgsize(i)/pcksize;
5:      for j=0 to npacks-1 do
6:        if j==0
7:          pkt_ts ← msg_ts;
8:        else
9:          pkt_ts ← prv_tmp + prv_pcksize*ncyclesflit + idle;
10:       end if
11:       generate(pkt_ts,pcksize)
12        prv_tmp     ← pkt_ts;
13:     end for
14:   else if packets_per_msg_fixed==true then
15:     pcksize ← msgsize(i)/pckspmsg
16:     for j=0 to pckspmsg-1 do
17:       if j==0
18:         pkt_ts ← msg_ts;
19:       else
20:         pkt_ts ← prv_tmp + prv_pcksize*ncyclesflit + idle;
21:       generate(pkt_ts,pcksize)
22:       prv_tmp  ← pkt_ts;
23:     end for
24:   end if
25:   msg_ts ← msg_ts + IMA(i);
26: end for
```

Figura 2.6 – Processo de geração de tráfego para o modelo ON-OFF [TED08].

O trecho de código compreendido entre as linhas 15 até 23 é executado quando o número de pacotes por mensagem é fixo (quando a variável *packets_per_msg_fixed* tem valor *verdadeiro*). Observa-se, desta vez na linha 20, a execução da Equação 2.1 para obtenção do momento de injeção do pacote na rede a exemplo do trecho comentado no parágrafo anterior. Na linha 25 é computado o momento de injeção para o primeiro pacote da próxima mensagem, de acordo com a função *IMA* (*Inter Messages Arrival*), cujo valor informa o intervalo (em ciclos de relógio) entre o início de uma mensagem *i* e sua

sucessora. Para aplicações de vídeo, por exemplo, o valor retornado pela função *IMA* é constante, considerando que um frame de vídeo é transmitido em uma mensagem.

No exemplo ilustrado na Figura 2.5, *chr*=800Mbps, *ipr*=160Mbps, *pcksize*=5 e *ncyclesflit*=1, resultando em um tempo ocioso entre pacotes (*idle*) de 20 ciclos de relógio. Entretanto, a Figura 2.5 mostra um intervalo ocioso de 18 ciclos de relógio, devido à influência dos *flits* de cabeçalho. O processo de injeção de dados lê o momento de injeção de cada pacote, subtraindo o número de ciclos necessário para inserir as informações de cabeçalho (no exemplo, 2 ciclos de relógio).

Como dito acima, a função *framesize* pode empregar *frames* cujo tamanho pode ser fixo, de acordo com funções de probabilidade, ou de acordo com *traces* de aplicações reais. Quando da utilização de funções de probabilidade, podem ser empregados os processos Pareto ON-OFF e Markov On-OFF. Considerando que uma variável aleatória define a duração de períodos de atividade e inatividade em cada utilização da equação relacionada sem referir a nenhum valor anteriormente computado, conclui-se que a utilização de modelos ON-OFF não permite controlar a taxa de injeção global na rede. Entretanto, estes modelos permitem controlar a taxa de injeção para cada pacote/rajada. Esta característica faz a distribuição ON-OFF ser adequada para modelar tráfego de ruído.

De acordo com [PAN05], um modelo ON-OFF que utiliza a distribuição Pareto serve para caracterizar aplicações como MPEG-2 e tráfego internet. As equações Equação 2.3 e Equação 2.4 descrevem as funções do processo Pareto ON-OFF para gerar tráfego.

$$t_{ON} = (1\text{-}r)^{-1/\alpha_{On}} \qquad \text{Equação 2.3}$$

$$t_{OFF} = (1\text{-}r)^{-1/\alpha_{Off}} \qquad \text{Equação 2.4}$$

Nestas equações, *r* é um valor dentro do intervalo [0:1], escolhido aleatoriamente, permitindo a geração dinâmica dos tamanhos dos períodos ON e OFF, sendo utilizados pelas funções *msgsize* e *IMA* no algoritmo da Figura 2.6. O parâmetro α é uma constante de formatação ON-OFF, definindo pelo usuário para adaptar a geração de tráfego para a característica da aplicação (para geração de tráfego de vídeo e aplicações de rede, [PAN05] recomenda utilizar α_{On}=1.9 e α_{Off}=1.25).

Uma alternativa ao processo Pareto ON-OFF é a utilização do modelo Markov ON-OFF. Neste modelo, também conhecido como Processo Modulado Markov (MMP - *Markov Modulated Process*), o estado corrente da fonte de tráfego em uma Cadeia de Markov especifica a geração de dados a uma taxa *r*. A função que descreve os períodos ON e OFF é uma exponencial.

Outro processo utilizado para modelagem de tráfego é o auto-similar. Um processo estocástico é dito auto-similar se ele mantém a mesma estrutura sobre diferentes escalas (comportamento conhecido com *fractal*). A auto-similaridade em aplicações de redes tem sido pesquisada nos últimos anos na área de comunicação de dados. Leland et al. em [LEL94] observaram o fenômeno da auto-similaridade nas taxas de transmissão de pacotes

em redes Ethernet. Já Beran et al. [BER95] e Varatkar e Marculescu [VAR04] observaram esta propriedade nas taxas de transmissão de macroblocos componentes de *frames* em aplicações multimídia.

Métodos de geração de tráfego auto-similar são utilizados por vários grupos de pesquisa da área de redes de comunicação de dados e NoCs. Cada método pode ser diferenciado de outro em termos de precisão, complexidade e tempo de execução. Alguns exemplos de métodos são os *Chaotic Maps* [HEG05], processo de superposição *Markov-Modulated Poisson* (proposto em [AND98] e utilizado em [MIN04]) e superposição de Pareto [LOG04].

2.2 MODELAGEM DE DISTRIBUIÇÃO ESPACIAL DE TRÁFEGO

A modelagem de tráfego *espacial* define iniciadores e destinos de tráfego, ou seja, qual ponto da rede se comunica com qual outro ponto. O projetista pode especificar uma conexão ponto-a-ponto (com um par origem-destino específico) ou um padrão de tráfego (origens e destinos escolhidos de acordo com uma função específica). Devido às similaridades que ocorrem entre arquiteturas paralelas e MPSoCs, padrões normalmente encontrados em aplicações paralelas podem ser utilizados para definir distribuições espaciais de tráfego em NoCs.

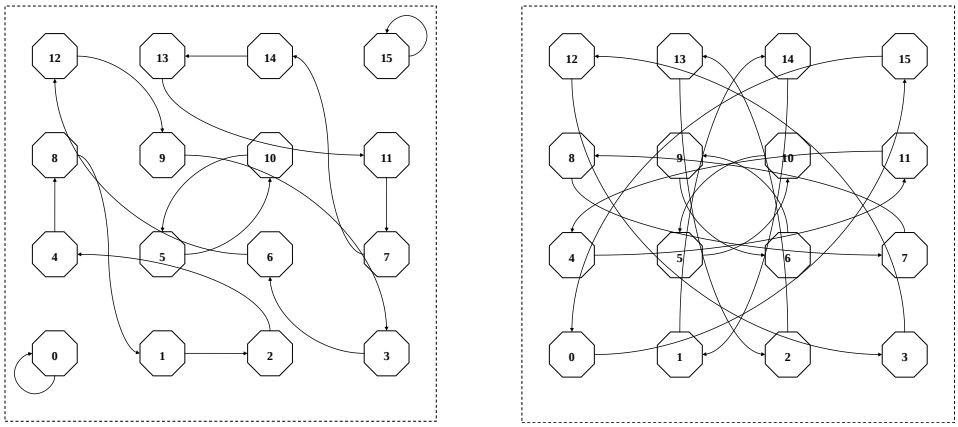
Um padrão de tráfego pode ser caracterizado de acordo com o grau de *localidade espacial*, e também em termos de *localidade temporal*. Uma dada aplicação exibe localidade espacial quando há maior probabilidade de comunicação entre módulos vizinhos. Como resultado, as mensagens acabam consumindo menos recursos da rede, também reduzindo a contenção [DUA03]. Uma aplicação apresenta localidade temporal quando há uma afinidade de comunicação entre um subconjunto de nodos. Neste caso, a probabilidade do envio de mensagens para nodos que foram recentemente escolhidos como alvos de mensagens anteriores é maior do que para outros nodos. Deve ser enfatizado que nodos que exibem afinidade de comunicação não precisam estar próximos uns dos outros na rede.

As distribuições *uniforme* e *não-uniforme* caracterizam-se pelo grau de localidade espacial de uma dada aplicação. Na distribuição uniforme, todos os nodos possuem a mesma probabilidade de serem alvos de tráfego. Neste caso há um baixo grau de localidade espacial. A utilização da distribuição uniforme é comumente utilizada por grupos de pesquisa da área de NoCs. Alguns trabalhos relacionados que utilizam este padrão são [BOL04][GEN05][SAN05]. No padrão de tráfego não-uniforme, a probabilidade um nodo transmissor enviar pacotes para os seus vizinhos é maior do que para outros nodos. Portanto, a natureza desta distribuição produz um maior grau de localidade espacial em comparação com o padrão uniforme, sendo adequada para caracterização de aplicações reais. Quando é utilizado o padrão não-uniforme, o projetista deve conhecer o grau de localidade de uma aplicação em estudo. Por exemplo, o grau de localidade 0,5 indica a probabilidade de 50% dos dados gerados por uma fonte serem transmitidos para nodos vizinhos.

Em termos de localidade temporal, a Tabela 2.1 apresenta algumas das principais distribuições criadas para a realização de teste em sistemas distribuídos. Nota-se que nestas distribuições, há apenas um nodo alvo para cada iniciador. Por exemplo, a relação posicional entre origens e destinos para o cálculo da Transformada Rápida de Fourier (FFT) e para aplicações de ordenação tomam como referência o padrão *perfect shuffle*. [HU04] e [TED05] por sua vez utilizam o padrão complemento. A Figura 2.7 ilustra a utilização destas duas distribuições espaciais em uma NoC malha 4x4.

Tabela 2.1 – Padrões de tráfego [DUA03].

Padrão de tráfego	Descrição
Bit reversal	O nodo com coordenadas binárias a_{n-1}, a_{n-2}, ..., a_1, a_0 comunica-se com o nodo a_0, a_1, ..., a_{n-2}, a_{n-1}.
Perfect shuffle	O nodo com coordenadas binárias a_{n-1}, a_{n-2}, ..., a_1, a_0 comunica-se com o nodo a_{n-2}, a_{n-3}, ..., a_0, a_{n-1} (rotação à esquerda de 1 bit).
Butterfly	O nodo com coordenadas binárias a_{n-1}, a_{n-2}, ..., a_1, a_0 comunica-se com o nodo a_0, a_{n-2}, ..., a_1, a_{n-1} (troca os bits mais e menos significativos).
Matrix transpose	O nodo com coordenadas binárias a_{n-1}, a_{n-2}, ..., a_1, a_0 comunica-se com o nodo $a_{\frac{n}{2}-1}$,...,a_0, a_{n-1}, …, $a_{\frac{n}{2}}$.
Complement	O nodo com coordenadas binárias a_{n-1}, a_{n-2}, ..., a_1, a_0 comunica-se com o nodo $\overline{a_{n-1}}$, $\overline{a_{n-2}}$,..., $\overline{a_1}$, $\overline{a_0}$.



(a) (b)

Figura 2.7 – Distribuição espacial em uma NoC malha 4x4 utilizando os padrões (a) *perfect shuffle* e (b) complemento.

Uma alternativa utilizada para geração de tráfego para vários destinos com localidade temporal é a utilização do padrão *hot-spot*. Neste padrão, um subconjunto de nodos da rede é escolhido como destino durante uma determinada porcentagem do tempo

total de geração de dados pelos núcleos IP. Desta forma, é forçada a carga de tráfego em pontos específicos da rede, onde há a maior probabilidade de haver congestionamento.

Como exemplo, a Figura 2.8 apresenta um padrão *hot-spot* em uma rede 4x4, onde são escolhidos como destinos 4 IPs localizados no centro da rede, que receberão 20% do tráfego total gerado na rede. Segundo [ASC08], o padrão de tráfego *hot-spot* é considerado um padrão de tráfego realista, uma vez que, na maior parte das aplicações, a comunicação ocorre freqüentemente com apenas uma parte do número total de núcleos IP (módulos de memória e entrada/saída).

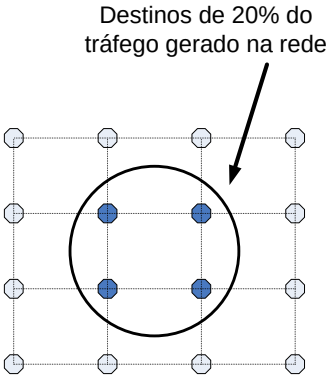


Figura 2.8 – Exemplo de configuração de tráfego com o padrão *hot-spot*.

Nos experimentos conduzidos na presente Tese, será considerado o padrão *hot-spot* como sendo aquele produzido por núcleos vizinhos que comunicam intensamente entre si, de modo utilizar recursos de rede em uma região bastante específica, causando congestionamento para os tráfegos concorrentes.

2.3 MODELAGEM DE TRÁFEGO COM A UTILIZAÇÃO DE GRAFOS DE APLICAÇÕES

Um APCG (Grafo de Caracterização de Aplicação – *APplication Characterization Graph*) representa a aplicação no nível de IPs. Conceitualmente, um APCG é um grafo dirigido <I,C>, onde I é o conjunto de vértices e C é o conjunto de arestas. Cada vértice $ip_{núcleo}$ representa um núcleo IP e cada aresta c_{ij} representa o processo de comunicação do ip_i com o ip_j. (Figura 2.9).

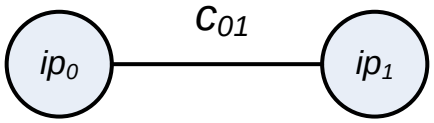


Figura 2.9 – APCG (Grafo de Caracterização de Aplicação).

Todas as arestas têm associado um valor específico da aplicação (e.g. volume de comunicação, taxa de comunicação), o qual especifica restrições de projeto (e.g. largura de banda da comunicação, requisitos de latência). A Figura 2.10 mostra exemplos de grafos de aplicações multimídia, encontrados em [BER05]. Cada aresta é rotulada pela especificação da taxa de comunicação (em MBytes/s) entre os nodos conectados.

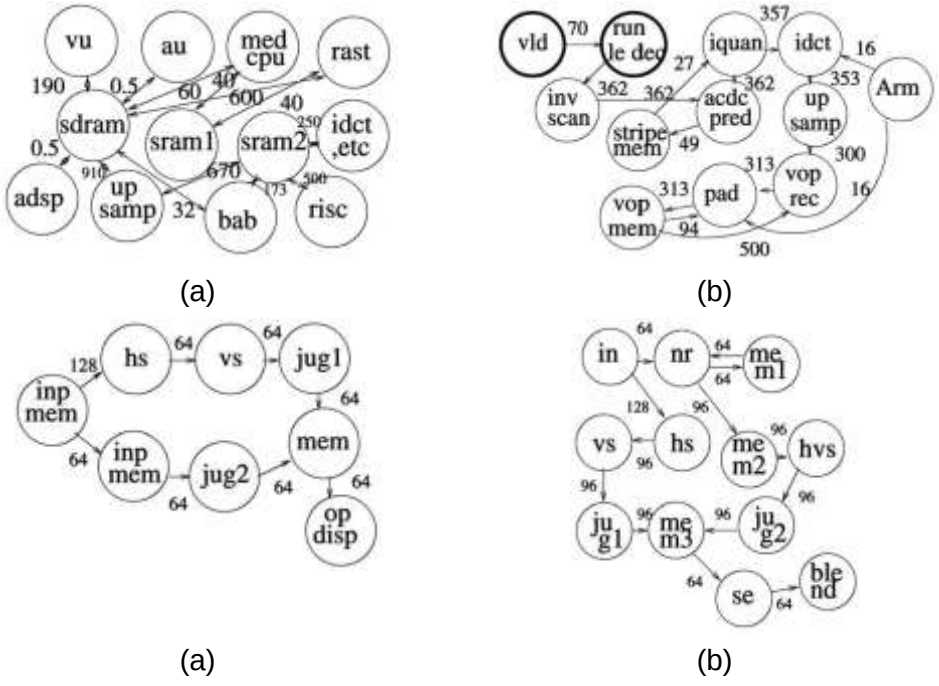


Figura 2.10 – Exemplos de grafos de aplicações multimídia [BER05].

Grafos de aplicações têm como características: (*ii*) *distribuição espacial fixa*, visto que são especificados em tempo de projeto as origens e destinos de tráfego; (*ii*) *localidade de tráfego dependente do mapeamento* dos nodos na rede; (*iii*) *taxa de injeção constante*, durante os períodos de atividade de tráfego da aplicação. Pelo fato de não haver uma referência de tempo na especificação de um grafo, esta abordagem não informa a ordem em que ocorrem os eventos de comunicação.

2.4 GERAÇÃO DE TRÁFEGO COM A UTILIZAÇÃO DE *TRACES*

Traces são descrições reais do comportamento de aplicações, em contraste com a captura de dados calculados a partir de funções de probabilidade e grafos. O método de geração de tráfego baseado em *traces* tem por objetivo caracterizar com exatidão a transmissão de dados pelos núcleos. Com isso, o projetista parametriza a transmissão dos dados com valores coletados a partir de um ambiente com tráfego real, permitindo assim uma representação precisa de um sistema conhecido. Esta abordagem apresenta, entretanto, a desvantagem de possuir um tempo excessivo de simulação, sendo apropriada para emulação em hardware [GEN05][HU04].

Para obter os *traces* é necessária a utilização de algum equipamento de medição acoplado à saída da instância do dispositivo a se modelar. Durante o processo de coleta dos *traces* são armazenados os valores medidos em arquivos que posteriormente são utilizados para conversão em formato de mensagens ou pacotes. A Figura 2.11 ilustra o processo de coleta de *traces* para geração de tráfego. As informações de tráfego normalmente referem-se a bits, sendo coletados fundamentalmente três parâmetros: o momento de transmissão (*timestamp* - *tmp*), o tamanho do dado a ser transmitido (*tam*) e o seu destino (*dst*). Cada linha do arquivo corresponde a um dado, podendo o mesmo ser

convertido em uma mensagem (que deverá passar pelo processo de empacotamento), ou mesmo um pacote.

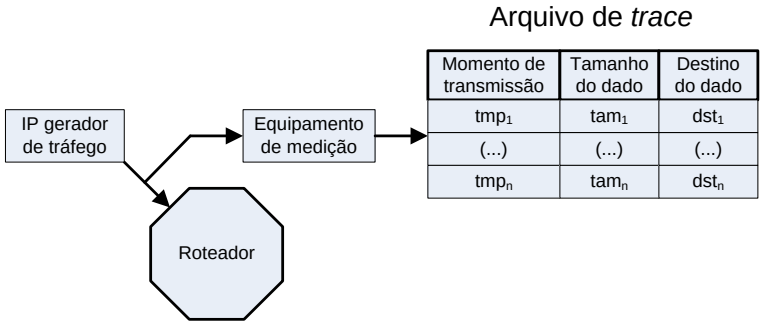


Figura 2.11 – Processo de coleta de *traces* para geração de tráfego.

As principais características de um tráfego gerado por *traces* são as seguintes: (i) *distribuição espacial fixa*, considerando que as origens e destinos de tráfego são pré-definidos de acordo com a aplicação alvo, sendo que normalmente cada IP gerador se comunica com poucos IP receptores de tráfego; (*ii*) localidade variável, visto que é definida pelo mapeamento dos nodos da aplicação na rede; (*iii*) *modelo UR de taxa de injeção de dados*, pelo fato de ser especificada a partir de arquivos de *trace*; (*iv*) ordem definida, especificada pelos valores de *timestamp*.

Em comparação com o método de geração de tráfego baseado em grafos, a transmissão baseada em *traces* possui como vantagem a especificação da ordem em que os eventos devem ocorrer. Isto traz mais fidelidade à simulação, visto que as transmissões e recepções de dados são dependentes entre si. Observando a Figura 2.12, verifica-se que é possível representar o método baseado em *traces* através de um diagrama de tempo, adicionando um eixo temporal em adição aos nodos e às ligações entre nodos. Entretanto, a construção de uma interface de rede que implemente tal método é mais complexa, visto que a geração de dados pode depender ou não da recepção de outros dados.

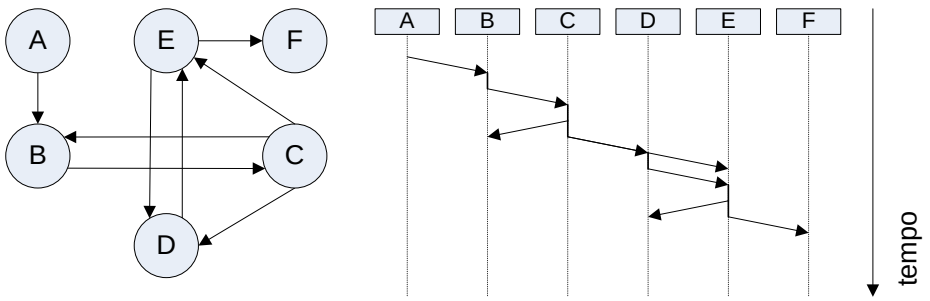


Figura 2.12 – Modelagem de tráfego baseada (a) em grafos; (b) em *traces*.

2.5 TRABALHOS RELACIONADOS À GERAÇÃO DE TRÁFEGO EM NOCS

Esta Seção apresenta trabalhos relacionados à geração de tráfego em NoCs. Os parâmetros de comparação entre os trabalhos estão relacionados à aplicação-alvo, à modelagem de taxas de injeção de dados e da distribuição espacial.

[HU04] compara o algoritmo de roteamento DyAD com três outros (XY, OE-fixo e odd-even), utilizando dois cenários de tráfego. No primeiro, os núcleos foram conectados em uma malha 6x6 gerando pacotes de 5 *flits*, com intervalo de geração variando de acordo com uma distribuição exponencial. Os padrões de tráfego utilizados foram o uniforme e o complemento. No outro cenário simulado, nove núcleos dispostos em uma malha 4x4 foram aleatoriamente escolhidos para gerar tráfego multimídia (dados obtidos a partir de *traces* reais), e os núcleos restantes gerando tráfego a uma taxa constante.

[VAR04] propõem um método para geração de tráfego através da coleta de *traces* de aplicações reais, sendo suas propriedades estatísticas combinadas com um procedimento de geração de tráfego sintético. O diferencial deste trabalho é a captura de propriedades de auto-similaridade em tráfegos multimídia, com o intuito de observar a utilização de recursos em um decodificador MPEG em tempo reduzido, visto que somente propriedades estatísticas do tráfego da aplicação são consideradas. O tráfego experimentado, no entanto, apresenta distribuição espacial específica, considerando apenas comunicação ponto a ponto entre os nodos.

[BOL04] emprega geração de tráfego com taxas constantes. Quatro níveis de serviço são definidos, modelando quatro tipos de tráfego: (i) sinalização, composto por requisições e interrupção e outros sinais de controle; (ii) real-time, sendo o tráfego gerado pelo processamento de *streams* de vídeo e áudio; (iii) leitura/escrita, modelando o acesso de palavras individuais de memórias; (iv) transferência de blocos, exemplificado pela transferência de grandes blocos de dados das aplicações ou de memórias. Dois tipos de distribuição espacial são empregados: uniforme e local (onde os vizinhos tem maior probabilidade de receber pacotes). Embora as taxas utilizadas sejam similares às utilizadas em aplicações reais, não há qualquer variação tanto no tamanho dos pacotes ou nos intervalos de transmissão.

[SOT06] utiliza *traces* que descrevem a movimentação de dados em CMPs (Chip Multiprocessors). Estes *traces* são utilizados como entrada em um algoritmo que extrai três propriedades do tráfego: (i) características de rajada, onde é observada a freqüência na qual são injetados pacotes durante o período ON, e o tamanho destas rajadas; (ii) as distâncias percorridas pelos pacotes na rede; (iii) qual a proporção do tráfego total é injetada a partir de cada roteador da rede. Segundo os autores, o erro do método em comparação com a simulação utilizando todos os tráfego das aplicações reais é menor do que 5%.

[PAN05] avalia o consumo de energia em NoCs de acordo com uma diversidade de modelos de taxa de injeção de dados e padrões de tráfego espacial. É utilizado o modelo Pareto ON-OFF para modelar tráfego auto-similar. Também é utilizado o processo Poisson, onde durante o período ON os pacotes são enviados aos destinos de acordo com a distribuição exponencial. As distribuições de tráfego espacial variam sua localidade de acordo com o cenário simulado, podendo ser uniforme (sem nenhuma localidade) até altamente local, sendo os vizinhos destinos de 80% do tráfego gerado (fator de localidade = 0.8).

[BER05] propõem um método para geração de NoCs otimizadas para aplicações específicas. Estudos de caso incluem a execução de processadores de vídeo, cuja interconexão de núcleos é especificada na forma de grafos (Figura 2.10). As aplicações são MPEG4, VOPD (*Video Object Plane Decoder*), PIP (*Picture-In-Picture*) e MWD (*Multi-Window Display*). Além destes, os autores também realizam experimentos com um processador de rede com 16 nodos e com um filtro DSP com 6 núcleos.

2.5.1 A APLICAÇÃO 3GPP-LTE

A aplicação que será tomada como referência neste trabalho é a parte de tratamento da recepção de dados de um sistema LTE (*Long Term Evolution*), sendo descrito em [CLE09], ilustrada na Figura 2.13. A interface LTE baseia-se na técnica de Acesso Múltiplo por Divisão de Freqüências Ortogonais (OFDMA – do inglês *Orthogonal Frequency Division Multiple Access*), sendo empregada em sistemas de comunicação 4G. No OFDMA, usuários do sistema são multiplexados em freqüência, sendo cada dado de usuário transmitido em um subconjunto de portadoras de um símbolo OFDM. Em comparação ao sistema FDM (*Frequency Division Multiplex* - Multiplexação por divisão de Freqüência), a técnica OFDM oferece adicionalmente a divisão de uma única transmissão em múltiplos sinais com menor ocupação espectral. Esta possibilidade, adicionado ao uso de técnicas avançadas de modulação em cada componente, resulta em um sinal com grande resistência à interferência.

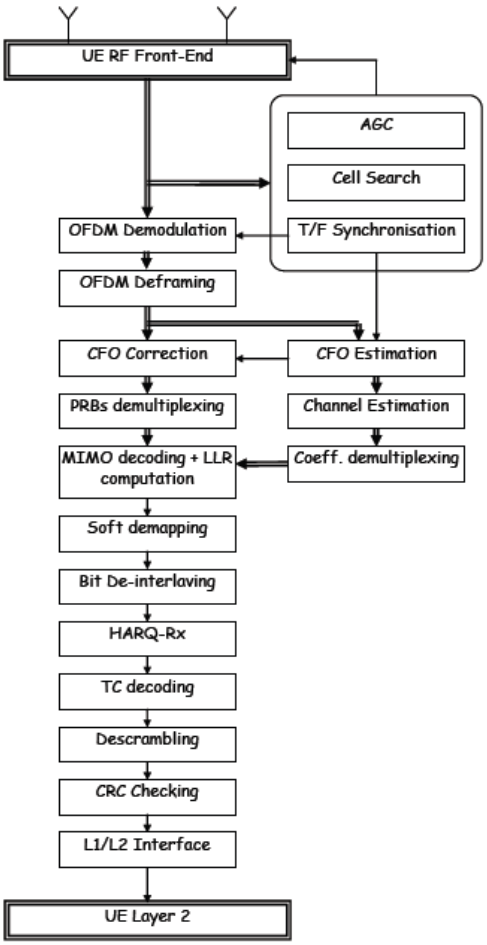
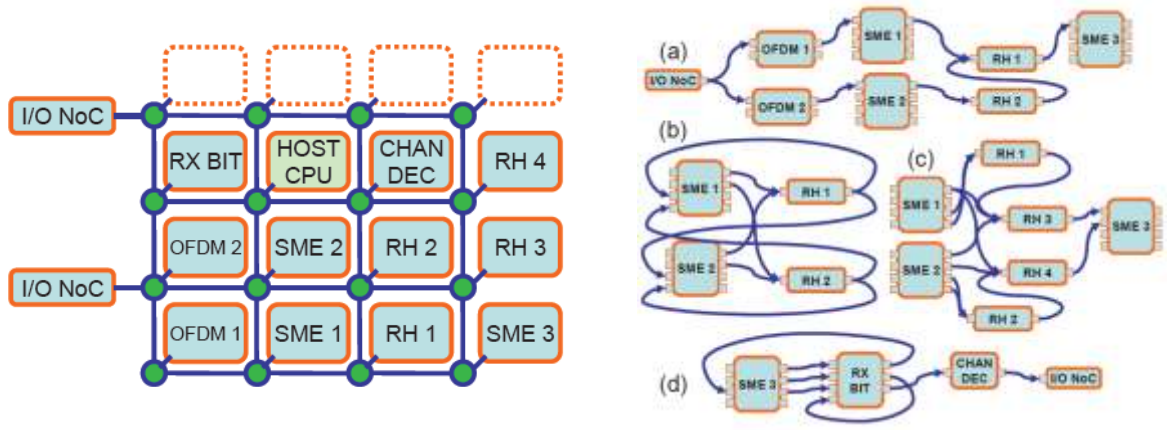


Figura 2.13 – Descrição functional do receptor OFDM [CLE09].

A Figura 2.13 apresenta o esquema do receptor, onde dois sinais oriundos de duas antenas são amostrados e demodulados. Após a demodulação OFDM, as informações e os símbolos de referência são separados em cada segmento de antena para posterior processamento. O módulo de estimativa de deslocamento de freqüência de portadora residual (CFO) baseia-se na correlação entre réplicas do mesmo símbolo para cada antena de recepção no domínio da freqüência.

A NoC que comporta esta aplicação (Figura 2.14(a)) possui conectados aos seus roteadores diferentes núcleos, caracterizando um MPSoC heterogêneo. Este MPSoC é denominado *Magali*. Um ponto importante a compreender na utilização de aplicações executando na *Magali* é o conceito de *fases*. Uma fase é um mapeamento de uma parte da aplicação. Assim, uma aplicação é particionada em diversas fases, sendo o sistema *reconfigurado* diversas vezes ao longo da execução da aplicação. A aplicação LTE é particionada em 4 fases, conforme ilustrado na Figura 2.14(b).



(a) (b)

Figura 2.14 – Mapeamento do MPSoC Magali [CLE09]: (a) NoC empregada e recursos a ela conectados; (b) Fases e fluxos de dados da aplicação alvo.

Na fase (a), são realizados a demodulação de símbolo e extração de *frames* de sub-portadora, sendo realizados pelos recursos OFDM (um símbolo para cada antena). Os módulos RH1 e RH2 computam a estimativa de CFO a partir de sub-portadoras armazenados nos módulos SME1 e SME2. Os resultados desta operação são enviados ao módulo SME3 e serão utilizados como parâmetros de configuração para a próxima fase. Cada recurso RH computa um fluxo de antena, sendo os resultados obtidos no RH1.

A fase (b) corresponde à correção CFO e estimativa de canal realizada por RH1 e RH2. Existem dois pares de símbolos piloto por TTI (*Time Transmission Interval* – Intervalo de Tempo de Transmissão). O módulo RH1 trata o primeiro para de ambas as antenas e o RH2 o segundo par. Os coeficientes de canal são armazenados nos SMEs. Na fase (c), sub portadoras de dados são enviados de SME1 e SME2 para o decodifcador MIMO paralelizado entre RH3 e RH4. RH1 e RH2 são reconfigurados para realizar interpolação de coeficientes de canal. Finalmente, na fase (d), o final da demodulação é realizado pelos módulos RX_BIT e CHAN_DEC configurado para computação turbo-decoder 3GPP-LTE. Um benefício que a NoC traz para a aplicação é a redução da

latência pelo fato dos fluxos de dados serem executados em paralelo. A Figura 2.15 mostra o cronograma de decodificação TTI para o cenário de pior caso (maior vazão).

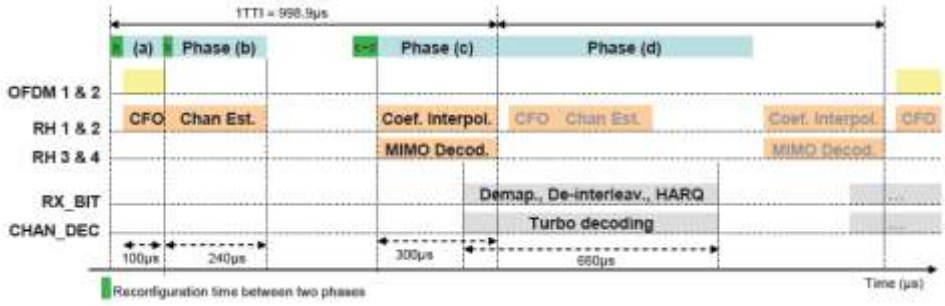


Figura 2.15 – Cronograma da aplicação [CLE09].

2.6 CONSIDERAÇÕES SOBRE GERAÇÃO DE TRÁFEGO

Esta Seção resume os métodos apresentados para geração de tráfego em NoCs e sua utilização no contexto deste trabalho. Os requisitos de desempenho de aplicações multimídia constituem ambientes relevantes na pesquisa de MPSoCs, visto a sua considerável utilização, como é apresentado na Tabela 2.2. Normalmente estas aplicações compreendem grandes blocos de dados, cuja transmissão deve respeitar intervalos de tempo restritos. [PAN05] utiliza o processo Pareto On-OFF para modelar suas taxas de transmissão, visto que este processo simula a característica da auto-similaridade. Em [HU04], *traces* de um decodificador de vídeo são capturados para serem posteriormente utilizados em simulação. A propriedade de auto-similaridade de tráfego multimídia também é explorada em [VAR04], onde *traces* de aplicações reais são utilizados como entrada para um gerador de tráfego sintético. [BER05] apresenta 4 grafos que descrevem a interconexão de núcleos pertencentes a aplicações de processamento de vídeo.

Tabela 2.2 – Modelagem de tráfego adotada no estado da arte.

Autor	Aplicação modelada	Modelo de taxas de injeção de dados	Modelo de distribuição espacial
[PAN05]	MPEG2	Pareto ON-OFF	Não-uniforme, com diferentes graus de localidade
[HU04]	Aplicação multimídia	Coleta de *traces* de um decodificador de vídeo H263	Complemento e Uniforme
[VAR04]	Decodificador de vídeo	Coleta de *traces* e aplicação de auto-similaridade	De acordo com o mapeamento da aplicação
[BER05]	MPEG4, VOPD, PIP, MWD, Rede e DSP	Grafo com taxa de comunicação constante entre os módulos durante a atividade do tráfego	De acordo com o mapeamento da aplicação
[SOT06]	Movimentações de dados em CMPs	Coleta de *traces* e aplicação de auto-similaridade	De acordo com o mapeamento da aplicação
[BOL04]	Sinalização e acessos a memória	Constant Bit Rate	Uniforme e não-uniforme
Proposta deste trabalho	**Sintético, sinalização, acessos a memória e 3GPP-LTE**	**Constant Bit Rate, UR e coleta de *traces***	**Complemento e *hot-spot***

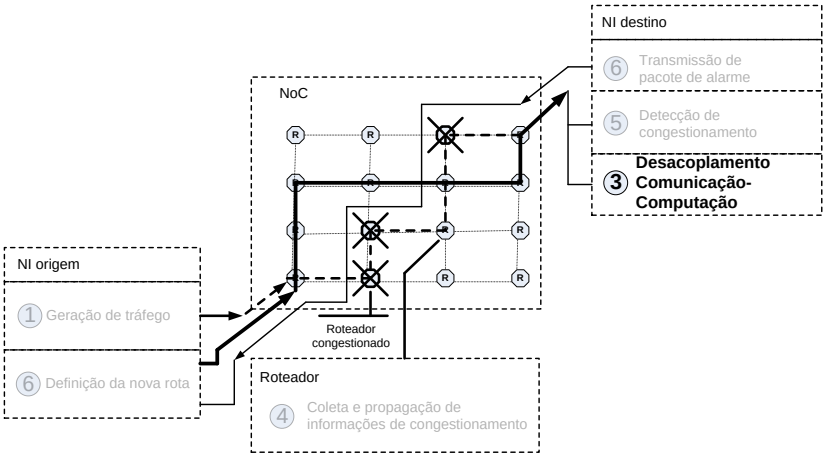
Movimentações de dados que ocorrem em CMPs (*Chip Multiprocessors*) são modeladas com a utilização de *traces* por [SOT06] e com a distribuição Pareto ON-OFF por [PAN05]. Este tipo de tráfego inclui operações complexas, as quais envolvem grandes

blocos de dados transferidos entre os módulos IP, em contraste com o que ocorre em dados de sinalização. Blocos de dados transferidos entre módulos IP definem comunicações que normalmente ocorrem entre dispositivos de armazenamento (discos e memórias voláteis) em conjunto com dados de sinalização. A sinalização corresponde a informações de controle que freqüentemente representam requisições originadas de unidades de processamento para memórias ou outras unidades de processamento. Este tipo de tráfego normalmente requer limites de encaminhamento *hard-real time*, no qual prazos para entrega de dados são extremamente rígidos. Outro exemplo de aplicação são as redes de sensores, nas quais sinais de controle são enviados para todos os nodos periodicamente, para verificar quais deles estão ativos. Por esta razão, este tipo de tráfego pode ser especificado utilizando o modelo CBR, como tamanhos reduzidos de pacotes, como ocorre em [BOL04].

A distribuição espacial de tráfego é especificada de acordo com a maneira como os nodos são mapeados, podendo, desta forma, haver maior ou menor localidade temporal entre os mesmos. Normalmente, o mapeamento é realizado em tempo de projeto, sendo os pares que se comunicam com maior intensidade colocados próximos uns dos outros de modo à melhor atender aos requisitos de entrega de dados.

No contexto deste trabalho, são realizadas avaliações sobre o sistema de monitoração e roteamento adaptativo proposto utilizando-se duas abordagens. Para tráfego *sintético*, em que serão utilizados para distribuição espacial os padrões complemento e *hot-spot*, com o objetivo de verificar como se comportam o roteamento adaptativo de acordo com diferentes graus de localidade, as taxas de injeção serão constantes. Para tráfego *real*, serão analisadas e aplicadas propriedades da aplicação 3GPP-LTE, utilizando principalmente a sua característica de alta localidade.

3. DESACOPLAMENTO COMUNICAÇÃO-COMPUTAÇÃO



Como exposto no Capítulo 1, o tráfego de aplicações típicas que executam sobre MPSoCs incluem sinais de controle, transferências de blocos de memória e *streams* de multimídia. Tais aplicações são caracterizadas por requisitos temporais (prazos para envio/recebimento de dados) e volumes de dados a serem transmitidos. Entretanto, a transmissão de um dado fluxo através de uma NoC com chaveamento por pacotes e método de repasse *wormhole* pode modificar a taxa original de dados, o que resulta em prazos não obedecidos no IP destino. Este fenômeno é conhecido como *flutuação de carga*. Três processos introduzem flutuação de carga: empacotamento, processamento no roteador e concorrência entre fluxos. Variações na latência causadas pela flutuação de carga são chamadas de *jitter*.

A Figura 3.1 ilustra um exemplo típico de dados chegando a um IP alvo com flutuação de carga. Na figura, IP0 gera dados para IP8, em uma NoC 3x3 (Figura 3.1(a)). A Figura 3.1(b)-A mostra a geração de 2 mensagens em uma dada taxa. A Figura 3.1(b)-B representa a injeção de dados na rede após o processo de empacotamento (acrescentando informações de cabeçalho em cada pacote), sendo a primeira fonte de flutuação de carga. Então, este fluxo atravessa a NoC, e em cada *hop* roteamento e arbitragem são executados, o que constitui a segunda fonte de flutuação de carga. O fluxo chega no IP8 (Figura 3.1(b)-C), com diferentes atrasos entre *flits*. Se este fluxo é bloqueado por outro, ocorre um terceiro fator que acarreta flutuação de carga, a concorrência entre fluxos.

Uma técnica para minimizar ou ainda eliminar a ocorrência de *jitter* é a introdução de um *buffer* de desacoplamento (*D-buffer*) no IP destino, os quais recebem dados da NoC, com *jitter*. O IP destino consome dados deste *buffer* na taxa da aplicação, sem *jitter*. Dois problemas devem ser resolvidos para implementar o *D-buffer*: (*i*) qual deve ser o tamanho do *buffer*? (*ii*) quanto tempo deve ser esperado antes do início do consumo de dados pela aplicação (*threshold*)? O tamanho do *buffer* bem como o *threshold* devem ser dimensionados com o objetivo de evitar: (*i*) *perda de dados*, que acontece se a NoC não

pode escrever no *D-buffer* (devido ao mesmo estar cheio); (*ii*) postergação indefinida, que acontece se o IP não possui dados para ler do *D-buffer* (devido ao mesmo estar vazio); (*iii*) *aumento de latência*, que acontece se o consumo de dados por parte do IP começa muito tarde.

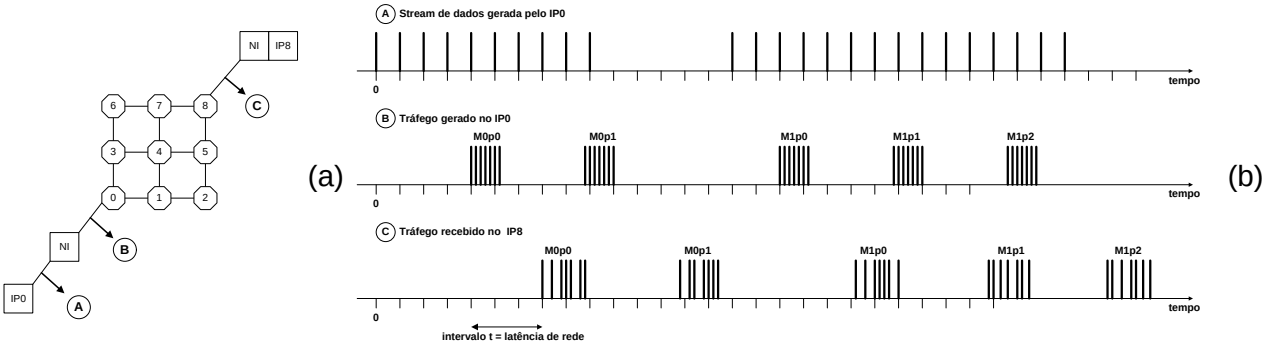


Figura 3.1 – O fenômeno da flutuação de carga em MPSoCs baseados em NoCs. (a) A fonte de tráfego IP0 envia dados para o destino de tráfego IP8; (b) ilustração de jitter em diferentes pontos no caminho entre a fonte e o destino.

3.1 TRABALHOS RELACIONADOS

Esta seção revisa trabalhos relacionados ao dimensionamento de *buffers* e sua relação com parâmetros de QoS. Hu et al. [HU06] apresentam um método para dimensionamento de roteadores internos componentes de NoCs, utilizando formalismos da teoria de filas. O principal objetivo da técnica empregada é a minimização das latências de todas as comunicações que ocorrem na NoC, com ocupação reduzida em área. Os autores consideram o pacote como unidade de armazenamento em *buffer*, o que caracteriza o método de repasse *store-and-forward*. Este método não é freqüentemente utilizada em NoCs, pelo fato de acarretar maior latência e consumo de área em comparação ao método wormhole. Isto porque o tamanho de cada *buffer* deve ser no mínimo o tamanho do maior pacote que pode ser transmitido pela rede.

Varatkar e Marculescu em [VAR04] demonstram a presença de auto-similaridade em tráfegos de aplicações multimídia. Através dos experimentos realizados observa-se que é possível definir um tamanho adequado para *buffers* de codificadores MPEG, de modo a evitar *overflow*. Este trabalho, no entanto, não apresenta qualquer menção ao parâmetro *threshold*. Tal fato se deve ao cenário proposto não apresentar qualquer nível de concorrência, o que pode acarretar *jitter* por *starvation* no IP destino.

Chandra et al. [CHA04] apresentam um método para dimensionar *buffers* considerando taxas de produção e consumo de pacotes transmitidos em rajada. A vazão é a métrica empregada para comparar a utilização de *buffers* localizados nos núcleos IP alvos (denominados pelos autores como *buffers atômicos*) ou encontrados em roteadores intermediários (denominados pelos autores como *buffers distribuídos*). Os resultados apresentados mostram uma maior vazão quando *buffers* distribuídos são utilizados. Neste método, a NoC é otimizada para um cenário de tráfego estático, sendo inadequado para aplicações dinâmicas.

Manolache et al. [MAN06] propõem um método baseado em heurísticas de tráfego para otimização de espaço em *buffer* em termos de tamanho e ocupação, considerando requisitos de latência e estimativas de descarte de pacotes. Neste método, os eventos de comunicação são mapeados na rede e/ou pacotes são atrasados na fonte (técnica conhecida como *formatação de tráfego*), com o objetivo de evitar concorrência entre diferentes fluxos. Os resultados obtidos mostram o efeito da formatação de tráfego na redução da quantidade total de espaço em *buffer* necessária em roteadores intermediários. Outro resultado obtido foi o aumento do número de aplicações que podem ser implementadas em uma quantidade específica de *buffers* disponíveis. Embora este trabalho proponha um método baseado em atraso na transmissão de dados na fonte, ele não discute a recuperação da taxa original na IP destino.

Nicopoulos et al. [NIC06] propõem uma estrutura de *buffer* unificada que dinamicamente aloca canais virtuais e recursos de bufferização de acordo com condições de tráfego. Este trabalho apresenta resultados enfatizando a redução de latência e aumento na vazão total na rede. No entanto, o método não provê garantias com relação a estes parâmetros na recepção dos dados pelos IPs destinos.

Coenen et al. [COE06] apresentam um algoritmo para dimensionamento de *buffers* em IPs destinos em uma NoC projetada com canais virtuais e controle de fluxo baseado em créditos. O objetivo do trabalho é garantir uma taxa de consumo de dados constante nos destinos com requisitos de QoS sem haver perda de dados. Os autores consideram as propriedades periódicas tanto da produção quanto do consumo dos dados em *buffers* de transmissão e recepção, respectivamente. Dois vetores armazenam informações que expressam os momentos de chegada de dados e o requisito de consumo de dados pelo núcleo IP destino. Este trabalho, entretanto, não considera a concorrência entre diferentes fluxos, o que pode tirar a periodicidade constante do tráfego.

3.2 MÉTODO DE DIMENSIONAMENTO PROPOSTO

Esta seção apresenta o método proposto para computar os valores de tamanho e o *threshold* para os *D-buffers*. Os *D-buffers* são parte da interface de rede do IP destino, como ilustrado na Figura 3.2. A definição correta do tamanho do *buffer* evita perda de dados, e a correta definição do *threshold* evita postergação indefinida. Como descrito no início deste Capítulo, a perda de dados acontece quando a NoC não pode escrever no *D-buffer* (devido ao *buffer* estar cheio). A postergação indefinida acontece quando o IP não possui dados para ler do *D-buffer* (devido ao *buffer* estar vazio). Estes dois parâmetros são obtidos utilizando dados coletados na porta de saída de cada roteador.

O método de dimensionamento de *buffers* aqui proposto é *off-line*, ou seja, realizado em tempo de projeto. Desta forma, a adaptação do tamanho de *buffers* ocorre de acordo com fenômenos que venham a descaracterizar o tráfego após o mesmo passar pelo processo de *packaging* (Figura 2.5). Como ilustrado na Figura 1.5, o dimensionamento de *buffers* é aplicado em nível de arquitetura (tempo de projeto - Figura 1.5).

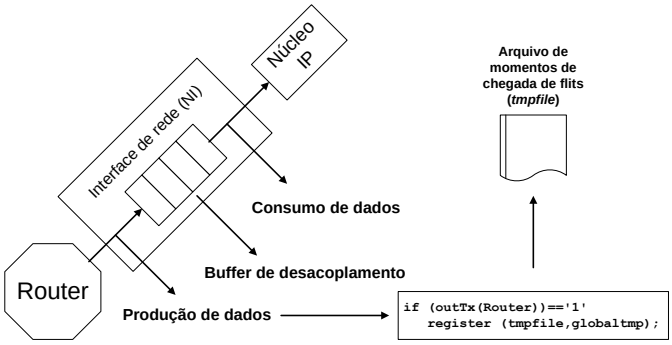


Figura 3.2 – Coleta de dados no *D-buffer* do NI [TED08].

As conseqüências do *jitter* introduzido por eventos na rede aos pacotes são armazenadas em um arquivo de *timestamps, o* qual contabiliza as chegadas dos *flits* (*tmpfile*, Figura 3.2). Um contador global é responsável pela computação do número de ciclos de relógio necessários, (variável *globaltime*) desde o início da simulação da NoC (simulação com precisão de ciclo de relógio). Um novo *flit* chega na porta de saída de um dado roteador quando o sinal *outTX* é ativado (a ativação deste sinal corresponde ao método de controle de fluxo utilizado na maior parte das NoCs: *handshake* ou *credit-based*). Para cada nova chegada de um *flit*, o valor atual do contador global é armazenado no arquivo de *timestamps* do roteador.

O método para dimensionamento de um *D-buffer* preenche dois vetores. O primeiro, denominado *recv*, coleta dados gerados após simulação na porta de saída do roteador alvo (*produção de dados*, na Figura 3.2). O segundo vetor, denominado *cons*, é preenchido de acordo com a taxa na qual o IP deve consumir dados. Estes vetores contêm em cada posição o valor ‘1’ ou ‘0’, de acordo com as seguintes regras:

- recv[index] = ‘1’, se um *flit* é recebido na porta de saída no momento *index*, senão ‘0’;
- cons[index] = ‘1’, se um *flit* deve ser consumido pelo IP destino no momento index, senão ‘0’.

O tamanho dos vetores *recv* e *cons* é igual ao número de ciclos de relógio correspondente ao parâmetro IMA (*Inter Messages Arrival*), que é o tempo decorrido entre duas mensagens (Figura 3.3). Este parâmetro, em conjunto com o parâmetro *size_ON*, formam os parâmetros QoS para a aplicação alvo, sendo obtidos em tempo de projeto a partir de arquivos de tráfego (*traces*) que descrevem o comportamento da aplicação na rede. O timestamp *TR0* corresponde ao momento de recepção do primeiro *flit* na porta de saída do roteador.

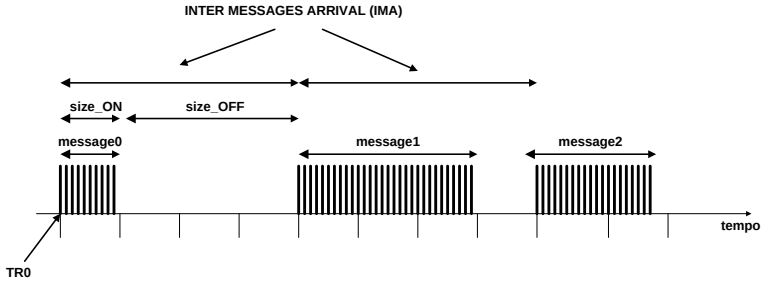


Figura 3.3 – Parâmetros utilizados na definição de tamanho de *buffer* e *threshold* [TED08].

A Figura 3.4 apresenta o algoritmo para preenchimento do vetor *recv*. $Tbegin_K$ e $Tend_K$ nas linha 1 e 2 definem o intervalo onde *flits* pertencentes à mensagem *k* devem ser recebidos, de acordo com os parâmetros *IMA* e *TR0*. A linha 3 inicializa o vetor *recv*, preenchendo todas as posições com o valor '0'. A linha 4 começa um laço no qual é buscado no arquivo *tmpfile* (momentos de chegadas dos *flits*) um valor dentro do mensagem *k* (linha 5). A diferença entre o momento de chegada do *flit* (valor *i*) e o momento de chegada do primeiro *flit* da mensagem ($Tbegin_K$ – linha 6) definem o índice do vetor *recv*. O vetor *recv* na posição computada recebe '1' (linha 7). O algoritmo termina quando o valor lido no arquivo *tmpfile* corresponde a um *flit* pertencente à próxima mensagem (linha 9).

```
1:  Tbegin_k ← TR0 + k*IMA
2:  Tend_k ← Tbegin_k + IMA - 1
3:  recv ← 0
4:  for all i ∈ tmpfile
5:    if i >= Tbegin_k  and i<= Tend_k  then
6:      index ← i – Tbegin_k
7:      recv[index] ← 1
8:    elsif i > Tend_k then
9:      exit
10:   end if
11: end for
```

Figura 3.4 – Algoritmo para preenchimento do vetor *recv*.

O vetor de consumo (*cons*) considera para cada mensagem *k* o parâmetro $size\_ON_k$ (em ciclos de relógio) e a taxa de consumo. A taxa de consumo é normalizada à largura de banda do canal. A Figura 3.5 apresenta o algoritmo para preencher o vetor *cons*. A linha 1 inicializa o vetor, preenchendo todas as posições com '0'. A linha 2 começa o laço, armazenando '1' no vetor *cons*, de acordo com a taxa de consumo.

```
1:  cons ← 0
2:  for all i ∈ SIZE_ON_k
3:    cons[i] ← 1 ∀ (i MOD 1/rate)=0
```

Figura 3.5 – Algoritmo para preenchimento do vetor *cons*.

A Figura 3.6(a) ilustra a execução do algoritmo de preenchimento do vetor *recv*. O momento de chegada dos *flits*, armazenados em *tmpfile*, são {130, 131, 132, 134, 146, 147, 149, 150, 166, 167, 168, 169}. Executando o algoritmo apresentado na Figura 3.4, o vetor *recv* é preenchido de acordo com a Figura 3.6(a). A Figura 3.6 (b) ilustra a execução do algoritmo para preencher o vetor *cons*. O tamanho definido para *size_ON* é de 28 ciclos de relógio e a taxa de consumo é de 25% (0.25). Desta forma, a cada 4 ciclos (1/0.25) um *flit* deve ser consumido do *D-buffer*. Os algoritmos são executados para todas as mensagens simuladas.

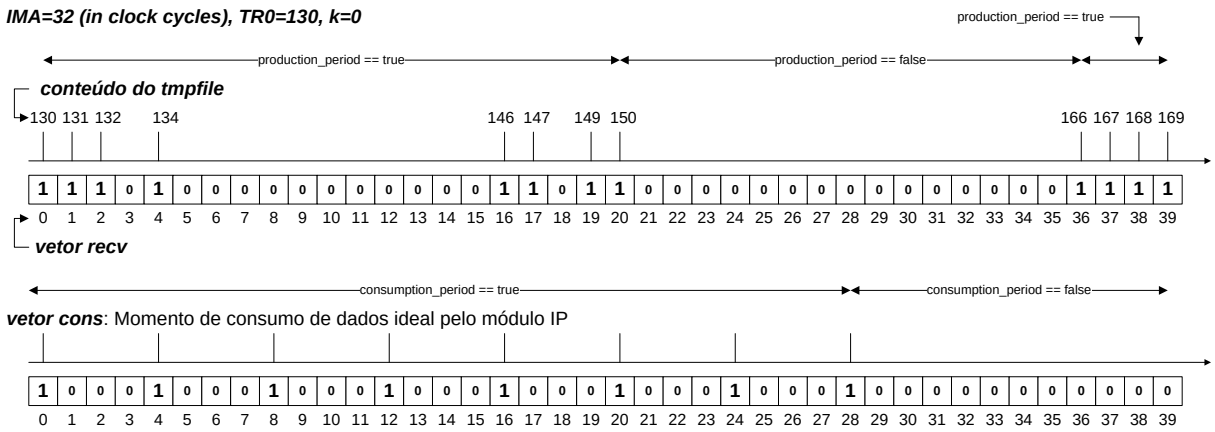


Figura 3.6 – Vetores (a) *recv* (recepção) e (b) *cons* (consumo).

Para definir os valores de tamanho e *threshold* do *D-buffer*, duas variáveis booleanas são consideradas: período de produção e período de consumo. O período de produção é verdadeiro no intervalo que compreende a recepção dos *flits* da mensagem atual, ou durante a recepção de *flits* pertencentes à próxima mensagem. O período de consumo é verdadeiro durante o período ON do intervalo *IMA*. Na Figura 3.6(a) é possível observar que parte dos elementos do vetor *recv* é preenchida com ‘1’s dentro do período de consumo. Os últimos 4 *flits* que chegam pertencem à próxima mensagem, mas aparecem na mensagem atual devido à flutuação de carga produzida por tráfego concorrente.

A Tabela 3.1 ilustra as condições que podem acontecer de acordo com os valores dos vetores *recv* e *cons*, e os valores das variáveis de produção e consumo. A condição *recv*[*i*]=’1’ e *cons*[*i*]=’1’ indica uma recepção e consumo de *flit* na fatia de tempo *i*. Neste caso, nenhum incremento no tamanho do *D-buffer* e nem no *threshold* é necessário.

Tabela 3.1 – Condições para definições do tamanho do *buffer* e *threshold* (*x* significa *não considerado*).

recv[i]	cons[i]	Período de produção	Período de consumo	Situação
0	1	qualquer	falso	Não aplicável, o vetor *cons* é verdadeiro somente durante o período de consumo de dados.
		qualquer	verdadeiro	O IP deve consumir um *flit* no *slot* de tempo i, mas não foi recebido neste *slot*. Esta situação acontece quando *flits* são bloqueados dentro da rede. Neste caso, é necessário incrementar a quantidade de tempo a esperar para o início do consumo de dados, isto é, aumentar o valor de *threshold* (por exemplo, índice 28 na Figura 3.6).
1	0	verdadeiro	verdadeiro	A recepção de um *flit* no slot *i*, antes do momento em que deve ser consumido. Esta condição freqüentemente ocorre após uma situação de congestionamento. Neste caso, é necessário incrementar o tamanho do *D-buffer* (por exemplo, índice 2 na Figura 3.6).
		verdadeiro	falso	Se o *flit* recebido pertence à mensagem atual, mas foi recebido fora do período de consumo, tanto o tamanho do *D-buffer* quanto o valor do *threshold* devem ser incrementados. Se o *flit* recebido pertence à próxima mensagem, um slot adicional de *buffer* é necessário se a diferença entre o número de *flits* produzidos e consumidos até este momento é menor que o tamanho do *buffer*. A quantidade de tais *flits* deve ser considerada para o cálculo do *D-buffer* em relação à próxima mensagem (por exemplo, índice 36 na Figura 3.6).
		falso	*x*	Não aplicável, todos os valores ‘1’ armazenados no vetor recv são inseridos em algum período de produção igual a verdadeiro.

O algoritmo apresentado na Figura 3.7 computa os valores de tamanho e *threshold* do *D-buffer*, considerando todos as mensagens simuladas (variável *simulated_messages*). É computada a diferença instantânea entre os dois vetores, sendo este valor armazenado na variável *acc*. A variável *acc* é inicializada com o número de *flits* recebidos na mensagem anterior, mas que pertencem à mensagem atual (linha 2). Outra variável, *nbflits*, conta o número de mensagens recebidos no NI destino, e é inicializado com zero a cada nova análise de mensagem.

```
1:  for all k ∈ simulated_messages
2:    acc ← first_flits
3:    first_flits ← 0
4:    nbflits ← 0
5:    for all i ∈ msg_k then
6:      if cons_period==true or
           (prod_period==true and nbflits<size(msg_k))
7:           acc ← acc + (recv[i]-cons[i])
8:           if acc > higher_acc  then
9:               higher_acc ← acc
10:          end if;
11:          if acc < lower_acc then
12:            lower_acc ← acc
13:          end if;
14:          nbflits++
15:     else if (recv[i]==1)
16:        first_flits=first_flits+1
17:     end if;
18:   end for;
19: end for;
20: Bsize ← higher_acc + |lower_acc|
21: Bthreshold ← |lower_acc|*1/rate;
```

Figura 3.7 – Algoritmo para preenchimento do vetor *cons*.

De acordo com a Tabela 3.1, a linha 6 do algoritmo expressa a condição para incrementar os valores de tamanho e *threshold* do *buffer*. Quando esta condição é verdadeira, a variável *acc* é atualizada (linha 7), o mesmo acontecendo para o maior (linha 8-10) e menor (linhas 11-13) valores de *acc*. Na linha 14 é incrementado o número de *flits* pertencentes ao período ON. Quando a condição expressa na linha 6 é falsa e um *flit* é recebido, a variável *first_flits* é incrementada, para ser utilizada na próxima mensagem (linha 16). O tamanho do *D-buffer* corresponde à adição do valor máximo de *acc* (*higher_acc*) com o seu valor mínimo absoluto (*lower_acc*) (linha 20). O valor mínimo absoluto de *acc* multiplicado pela taxa de consumo indica a quantidade de tempo que deve ser esperada antes do início do consumo de *flits*, ou seja, o valor do *threshold* (linha 21).

3.3 CONSIDERAÇÕES SOBRE DIMENSIONAMENTO DE *BUFFERS*

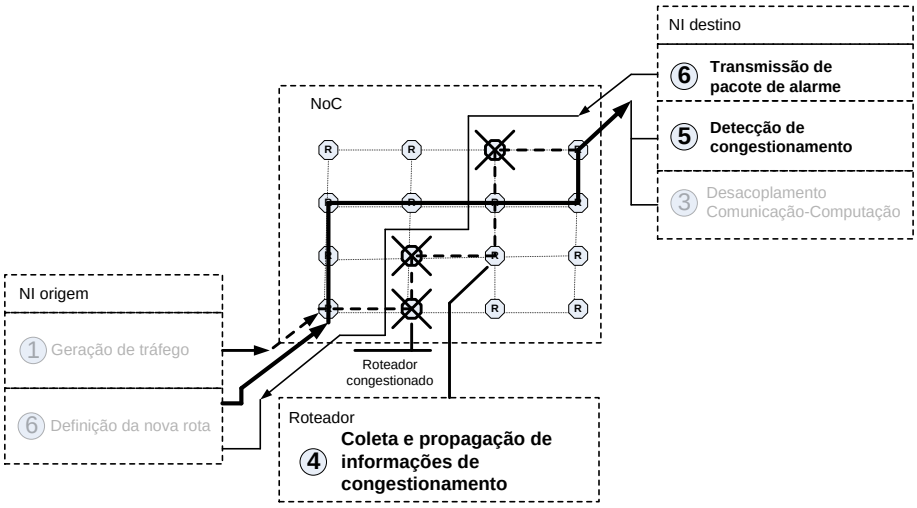
A Tabela 3.2 apresenta um resumo dos trabalhos relacionados a dimensionamento de *buffers*. Estes trabalhos apresentam basicamente três métodos para dimensionamento de *buffers*. A maior parte destes trabalhos [HU06][VAR04][MAN06] executam o dimensionamento de *buffers* em roteadores intermediários. O principal objetivo é a redução de latência média em todo o sistema embarcado. Métodos para requisitos rígidos de QoS são implementados em [MAN06] e [COE06], para aplicações sensíveis à latência e vazão, respectivamente.

Tabela 3.2 – Características de trabalhos relacionados a dimensionamento de *buffers*.

Autor	Objetivo	Dimensionamento de *buffers* interno ou externo?	Efeitos de tráfego concorrente são considerados?
[HU06]	Minimizar a latência media, com área de ocupação de *buffer* reduzida.	Interno	Sim
[VAR04]	Otimização de tamanho de *buffer* para módulo de um codificador MPEG, sob tráfego auto-similar.	Interno	Não
[CHA04]	Um algoritmo para reduzir o espaço de busca para obtenção de um tamanho otimizado de *buffers*.	Ambos	Não
[MAN06]	Otimização de espaço em *buffer* em termos de tamanho e ocupação, baseado em heurísticas de tráfego.	Interno	Sim
[NIC06]	Alocação de canais virtuais dinâmicos e recursos em *buffer*, de acordo com condições de tráfego.	Estrutura unificada	Sim
[COE06]	Dimensionamento de *buffers* em NIs de roteadores alvo, para garantir uma taxa de consume de dados constante pela aplicação.	Externo	Não
Proposta deste Trabalho	**Eliminação de ocorrências de *buffer overflow* e *starvation* na operação de *buffers* de interfaces de rede externas em NoCs.**	**Externo**	**Sim**

Uma parte significativa dos trabalhos discutidos não considera a concorrência entre fluxos. Além disso, quando modelos abstratos de NoC são utilizados (por exemplo, descrições em SystemC) o jitter induzido pelo processo de empacotamento e o processamento de roteamento/arbitragem não é levado em consideração pelos métodos de dimensionamento. O método exposto na Seção 3.1 mostra um método genérico para definição dos valores de tamanho e threshold para *D-buffers*, considerando a influência do empacotamento, roteamento/arbitragem e concorrência entre fluxos.

4. MONITORAÇÃO DE TRÁFEGO



A monitoração de tráfego em NoCs tem por objetivo observar o comportamento de seus eventos de comunicação, de modo a oferecer suporte a algum dispositivo de controle, que por sua vez atua de modo a manter o correto funcionamento no MPSoC ao qual esta rede está inserida. O elevado número de núcleos em MPSoCs com requisitos rígidos de desempenho, além da própria complexidade inerente às NoCs fazem da monitoração um aspecto crítico no projeto de sistemas embarcados. Neste capítulo são apresentados componentes e configurações que podem ser aplicados em um sistema de monitoração, trabalhos relacionados ao emprego de monitoração em NoCs, e o sistema de monitoração proposto na presente Tese.

4.1 COMPONENTES DE UM SISTEMA DE MONITORAÇÃO

O processo de monitoração ocorre basicamente em duas etapas: *coleta de informações* e *processamento dos dados* coletados. *Probes* (pontos de teste) são anexados a componentes do sistema, onde coletam as informações de tráfego (Figura 4.1). A conexão dos *probes* à rede pode ser realizada de maneira *externa* ou *interna*. *Probes* conectados de forma externa (*Probes* Externos - PEs), são aqueles acoplados às Interfaces de Rede dos IPs receptores de tráfego. Na conexão interna, um sub-conjunto ou o total de roteadores da NoC possui um *probe* acoplado (*Probes* Internos - PIs).

Os dados então coletados devem ser transmitidos para um sistema de controle (denominado MSA – *Monitoring Service Access Point* - de acordo com [CIO06]), que por sua vez os processa e toma alguma decisão para tratar os eventos coletados de acordo com os problemas diagnosticados. As informações de monitoração normalmente incluem valores de variáveis relativas a eventos que interferem no desempenho da aplicação, sendo considerados parâmetros de contenção e liberação de pacotes, bem como a utilização de filas de roteadores.

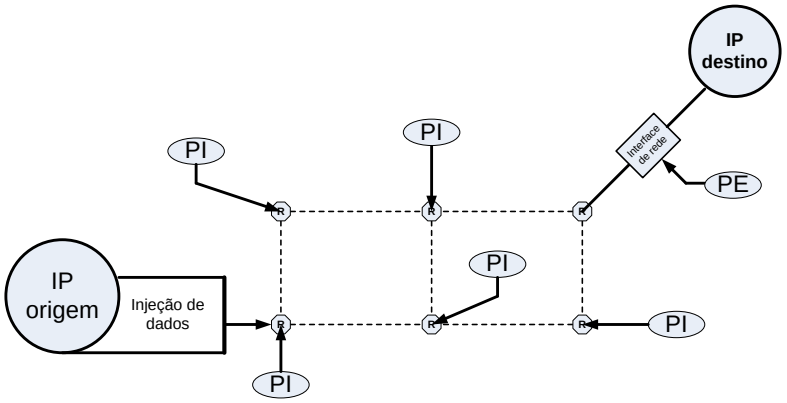


Figura 4.1 – Monitoração interna (PI) e externa (PE).

4.2 PARÂMETROS DE MONITORAÇÃO

Esta sessão descreve quais parâmetros são observados durante o processo de monitoração. Observa-se em trabalhos relacionados à avaliação de três tipos de eventos, que influenciam no desempenho das rede: a *contenção* de pacotes, a *liberação* de pacotes e a *utilização* de filas.

A avaliação da *contenção de pacotes* verifica a obediência de requisitos relacionados ao tempo de envio de mensagens de uma dada aplicação. Os seguintes parâmetros podem ser monitorados: (*i*) *latência*, intervalo de tempo decorrido entre o início da transmissão de um pacote e sua completa recepção no destino; (*ii*) *jitter*, grau de variabilidade dos valores de latência, através do cálculo do seu desvio padrão; (*iii*) *cpf*, número de ciclos para transmitir um *flit* entre roteadores vizinhos.

Os requisitos de *liberação de pacotes* são avaliados com o objetivo de verificar o cumprimento ou não de requisitos de taxa de transmissão de dados pela fonte, recepção no destino e ocupação dos canais da rede. O *tráfego aceito* especifica a relação existente entre as taxas de injeção e as taxas de recepção, sendo os dados a serem analisados coletados nas interfaces externas dos núcleos geradores e receptores de tráfego. A *taxa de utilização de canais* especifica o percentual de tempo em que o mesmo está ocupado com a transmissão de dados.

Durante a monitoração de *utilização de filas* são verificados os *buffers* de entrada dos roteadores (*buffers* internos) e/ou aqueles conectados às interfaces de rede (NIs) que intermediam a comunicação entre núcleos IP e os roteadores aos quais estão conectados (*buffers* da NI). Parâmetros importantes são: (*i*) a quantidade de *slots* preenchidos; (*ii*) a quantidade de dados não inserida em *buffer* da NI pelo fato do mesmo estar cheio; (*iii*) o número de vezes que a aplicação quis consumir dados de *buffer* da NI pelo fato do mesmo estar vazio; (*iv*) o número de ocorrências em que o crédito em *buffers* internos de roteadores vizinhos estiver em nível baixo.

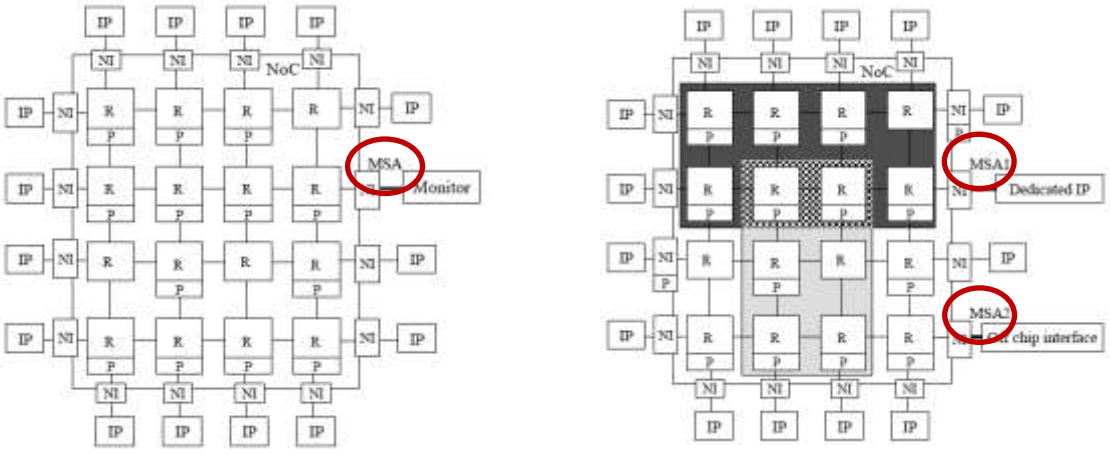
O processamento de dados de monitoração pode ocorrer em momentos distintos dentro do fluxo de projeto de MPSoCs. Quando o processamento é dito *off-line*, os dados são avaliados após a simulação do sistema. Os dados capturados servem como referência para alguma modificação na arquitetura da rede, realizada em tempo de projeto.

Entretanto, se os dados são processados durante a operação do sistema, o processamento é dito *on-line*. O processamento *off-line* é mais apropriado para tratamento de tráfego estático, onde é maior a previsibilidade do comportamento da aplicação. Considerando, entretanto, que um ambiente de simulação de MPSoCs é passível de tráfego dinâmico, torna-se necessário o processamento *on-line* de informações de monitoração, o que possibilita o tratamento da rede em um momento próximo à ocorrência de eventos não previstos.

Dados de controle informam qual foi o evento decorrido a partir de problemas gerados pelo tráfego da rede, podendo incluir os seguintes tipos de ocorrência: (*i*) *buffer* overflow (através de informações de crédito em *buffers* de roteadores vizinhos ou sinal originado da aplicação); (*ii*) *starvation* (quando a aplicação não tem dados a serem consumidos do *buffer* da interface de rede); (*iii*) altos valores de latência e/ou desvio padrão da latência; (*iv*) flutuação de carga, quando a taxa de injeção/encaminhamento/recepção de dados apresenta alta variação no caminho percorrido pelos pacotes de uma aplicação. Cada evento pode ser ainda associado ao momento em que ocorreu, sendo este parâmetro normalmente denominado *timestamp*.

4.3 MONITORAÇÃO CENTRALIZADA E DISTRIBUÍDA

Duas abordagens podem ser adotadas quanto à localização do serviço de processamento dos dados de monitoração, sendo ilustradas na Figura 4.2. No serviço de monitoração *centralizado* a informação obtida pelos *probes* (P) é coletada em um ponto central (MSA em destaque na Figura 4.2 (a)). A convergência do tráfego de monitoração para um único ponto pode acarretar congestionamento na rede, sendo esta abordagem mais adequada para redes pequenas. No serviço de monitoração *distribuído*, a informação a ser monitorada é coletada de diferentes sub-conjuntos de componentes de NoC, localizados em diferentes pontos da rede. Na Figura 4.2(b), MSA1 e MSA2 controlam diferentes regiões da rede. Nesta abordagem, gargalos de desempenho são removidos e obtém-se escalabilidade, o que torna esta estratégia mais adequada para redes de maior tamanho.



(a) (b)

Figura 4.2 – Serviço de monitoração [CIO04]: (a) centralizado (b) distribuído.

4.4 TRANSMISSÃO DE INFORMAÇÕES DE MONITORAÇÃO

As maneiras como podem ser transmitidas as informações de monitoração são ilustradas na Figura 4.3. Observa-se nos exemplos abaixo a utilização de um processamento de informações centralizado, onde o MSA é um núcleo IP conectado à rede. É possível, entretanto, a utilização de uma configuração distribuída. Em (a) uma rede física específica conecta os *probes*, podendo esta estrutura assumir uma topologia diferente em relação à rede de dados, ou até mesmo ser adotado um número menor de *probes*, dando cobertura a regiões mais críticas da rede. O fato dos dados de monitoração trafegarem em uma rede isolada traz como vantagem a ausência de interferência que seria causada por dados de aplicações. As desvantagens desta abordagem estão relacionadas ao custo em área de silício e a dificuldade no reuso da estrutura para outros tipos de rede. Podem ser ocasionados ainda problemas elétricos na estrutura da rede, afinal, a inserção de mais fios aumenta a probabilidade de aparecimento de *crosstalk*, que é um fenômeno que pode ocasionar erros na transmissão dos dados.

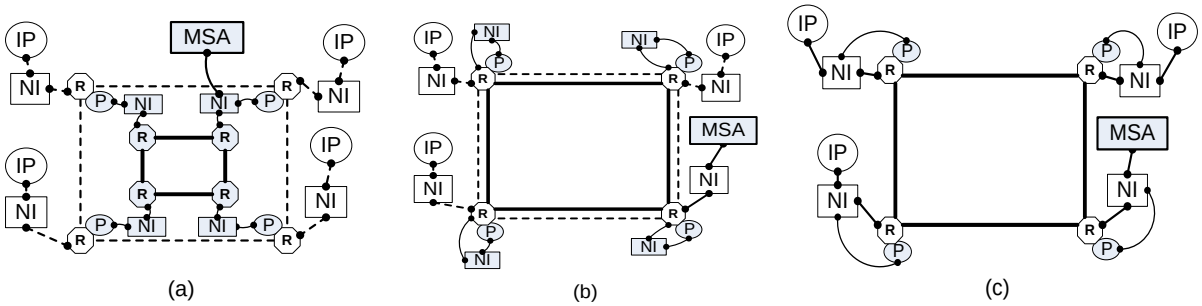


Figura 4.3 – Alternativas para transmissão de dados de roteamento [CIO06]. Linhas cheias identificam canais de controle e linhas tracejadas indicam linhas de dados de aplicações.

Em (b) é mostrada a abordagem onde a NoC é a estrutura de comunicação utilizada, com os dados de monitoração transmitidos em fios dedicados. Esta abordagem também traz como desvantagem a área em silício ocupada e a possibilidade de ocorrência de *crosstalk*, assim como mostrado na primeira abordagem. O fato dos enlaces serem conectados aos mesmos roteadores que a rede de dados de aplicações traz ao sistema flexibilidade, uma vez que é possível o acionamento da transmissão de dados por estas vias quando as mesmas não estiverem enviando dados de monitoração.

Finalmente em (c) é mostrada a abordagem onde os enlaces que interligam os roteadores têm sua utilização multiplexada entre tráfego de dados de monitoração e de aplicações. Neste caso, é possível a interferência dos dados de aplicações na transmissão de dados de monitoração, o que pode ser minimizado com a adoção de um mecanismo de escalonamento de comunicação baseado em prioridades. De acordo com [CIO06] o menor custo em área e reuso da estrutura constituem as vantagens desta abordagem, em comparação com as outras duas anteriormente citadas.

4.5 TRABALHOS RELACIONADOS À MONITORAÇÃO EM NOCS

Marescaux et al. em [MAR05a] utilizam dois monitores, localizados nos roteadores da NoC: *monitor de buffer* e *monitor de congestionamento* (Figura 4.4). O monitor de *buffer* utiliza quatro sinais: *new_packet*, acionado quando um novo pacote chega ao roteador; *source*, que descreve a fonte do pacote; *priority*, que especifica a prioridade do pacote e *packets_outqueue*, que informa quantos pacotes estão no *buffer*. O congestionamento é monitorado através da verificação de quantos pacotes estão armazenados no *buffer*. Se existirem mais pacotes no *buffer* do que especificado em um nível de *threshold*, significa que o mesmo está congestionado. Uma vez detectado o congestionamento, deve ser montada uma mensagem de notificação para um destino que deve ser selecionado.

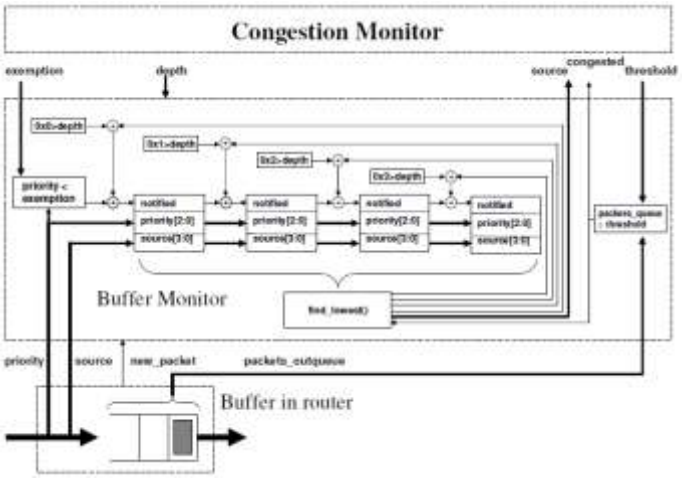


Figura 4.4 – Esquema de monitoração proposto por [MAR05a].

Com o intuito de guiar decisões de envio de pacotes de notificação, o *buffer* do monitor armazena informações relativas a pacotes recentes em uma fila separada de histórico, denominada *hFIFO*, a qual contém o endereço fonte e a prioridade do pacote. É então invocada a função *find_lowest* para pesquisar todos os campos de prioridade no *hFIFO* para encontrar o endereço fonte do pacote que possui a menor prioridade. Se dois pacotes possuem a mesma prioridade, a fonte do pacote mais recente é selecionada.

Uma mensagem de notificação de congestionamento contendo o endereço fonte e a prioridade é então reportada ao monitor de congestionamento pertencente à NI fonte. A entrada na FIFO de histórico é marcada como *notified* para evitar notificações subseqüentes ao monitor de congestionamento. A marca *notified* é local ao *hFIFO* do monitor de *buffer*, portanto o mesmo pacote pode acionar mais notificações de congestionamento nos próximos *hops* ao longo do caminho de um fluxo, podendo gerar na NoC mais mensagens de notificação em relação ao mesmo pacote. Finalmente, a mensagem de controle é gerada, utilizando fios específicos para transmitir notificações de congestionamento sobre a rede para as fontes de tráfego relacionadas, as quais ajustam a sua taxa de injeção.

Van den Brand et al em [BRA07] propõem uma estratégia de controle de congestionamento com o objetivo de limitar a latência na rede, denominada CCBE

(*Congestion Controlled Best Effort*). Os autores definiram a utilização dos enlaces como sendo o parâmetro de medição de congestionamento, pelo fato de o considerarem mais preciso que a ocupação em *buffers*, que seria uma conseqüência da contenção no enlace. Medidas de congestão realizadas no enlace especificam de forma mais direta em que ponto da rede a congestionamento ocorre.

A técnica *Model Predictive Control* (MPC) combina predições baseadas na modelagem de um sistema com dados de medição. Controladores MPC oferecem suporte para tratar latências com variabilidade (*jitter*), o que é considerado um fator crítico. Parâmetros de referência, como por exemplo, utilização de enlaces e cargas oferecidas por conexões *best-effort* podem ser especificados ao MPC, o qual toma decisões de controle com base nestes valores para que o sistema não oscile. A Figura 4.5 ilustra a estrutura de controle de congestionamento proposta. O controlador MPC é utilizado de forma centralizada em conjunto com o serviço de monitoração.

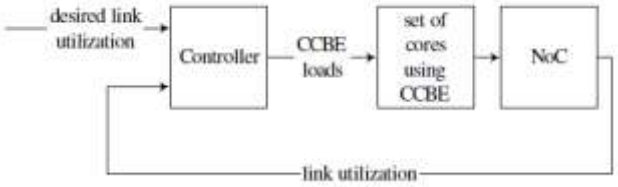


Figura 4.5 – Visão geral do controlador de congestão proposto por [BRA07].

Probes espalhados pela rede são utilizados para obter as medidas de utilização dos enlaces. Tais informações são transmitidas ao longo de canais de Serviço Garantido, para atingir o objetivo de se obter um sistema confiável, sem interferência de outros fluxos da rede. A partir dos dados de medição, é realizado o controle de injeção de dados na rede pela fonte, de modo a evitar congestão.

Em [CIO04], Ciordas et al. propõem e analisam os benefícios de um serviço de monitoração genérico para NoCs onde os *probes* de monitoração são conectados a componentes de NoC (roteadores ou interfaces de rede), sendo permitido: captura não-intrusiva de informações em tempo de execução; modelagem baseada em eventos monitorados, configuração, controle e utilização da arquitetura de monitoração. A transmissão de dados de monitoração ocorre através de conexões GT, BE de forma exclusiva ou alternada, para balancear o tráfego do serviço de comunicação.

O modelo de programação apresentado descreve a maneira na qual o serviço de monitoração irá trabalhar, sendo composto de uma seqüência de passos de configuração dos *probes* e os meios de implementar estes passos. O serviço de monitoração é configurado em tempo de execução, por um MSA. O serviço pode ser distribuído ou centralizado, como mostrado na Figura 4.2, o que traz flexibilidade ao sistema, uma vez que os subconjuntos de *probes* podem ser reconfigurados em qualquer instante. O gerenciador de tráfego regula o tráfego do MSA para os probes (quando for requisitada a configuração dos *probes*) e o tráfego dos *probes* para o MSA (quando forem requisitadas informações de monitoração).

Toda a informação monitorada é modelada na forma de *eventos*. Cada evento possui os seguintes parâmetros: *identificador*, o qual define a sua classe; *timestamp*, que define o momento no qual é gerado o evento; *produtor*, que é a entidade que gera o evento; e *atributos*, que definem valores relacionados aos eventos. Os *probes* de monitoração responsáveis pela captura das informações na NoC não são necessariamente conectados a todos os componentes da NoC, sendo sua localização definida em tempo de projeto. Para a definição desta localização é necessário observar o compromisso de seu custo com a quantidade de eventos observados (observabilidade). Eventos monitorados são relacionados: (*i*) à configuração das comunicações de usuário (abertura e fechamento de conexões); (*ii*) às informações das aplicações da rede, como por exemplo movimentações de dados da memória; (*iii*) à configuração do funcionamento da rede; (*iv*) à alertas da rede, como por exemplo indisponibilidade em *buffers*; (*v*) aos próprios serviços de monitoração, como por exemplo, perda de dados.

Kim et al. em [KIM07] propõem um sistema de monitoração de tráfego onde são medidos diversos parâmetros em tempo de execução, com o objetivo de, através dos valores medidos, modificar os tamanhos de *buffers*, bem como os caminhos de roteamento. A Figura 4.6 mostra a estrutura do sistema de monitoração de tráfego, que consiste de três sub-sistemas: interface hospedeira, controlador central e unidades de monitoração. A interface hospedeira transfere resultados de monitoração de tráfego para um computador através de uma conexão Ethernet. O controlador central externo à rede habilita/desabilita cada unidade de monitoração baseado em um escopo de requisições de monitoração a partir de regiões da rede e um intervalo de tempo.

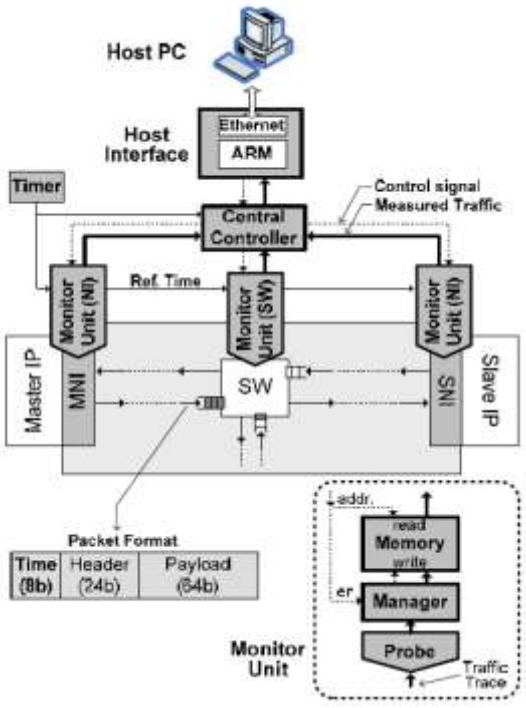


Figura 4.6 – Sistema de monitoração de tráfego proposto por [KIM07].

A unidade de monitoração consiste de um *probe* de tráfego, um gerenciador de tráfego e uma memória de tráfego. Um *probe* de monitoração é conectado a um roteador ou a uma interface de rede com o objetivo de registrar parâmetros de tráfego em tempo

real, como latência fim-a-fim, preenchimento de *buffers* e utilização de enlaces. O gerenciador de tráfego então armazena os dados em sua memória local após anexar um *timestamp* em cada registro utilizando como referência um contador global conectado a todas as unidades de monitoração. Durante a operação de uma aplicação, todos os resultados monitorados são armazenados na memória local correspondente, a qual é acessível pela CPU conectada à interface hospedeira através do controlador central. Este monitor pode ser anexado a qualquer componente da NoC em tempo de execução.

Al Faruque et al. em [ALF08] propõe um sistema de monitoração baseado em *eventos* que ocorrem em NoCs. Este infraestrutura oferece adaptatividade em termos de *sistema* e de *arquitetura*. A adaptação em nível de *sistema* é aplicada mapeando-se aplicações distribuídas baseados em *agentes* (entidades computacionais, executadas via software), em tempo de execução. A adaptação em nível de arquitetura ocorre pela execução de um algoritmo de roteamento e atribuição de *VCB*s (*Buffers* de Canais Virtuais) de acordo com a sua demanda, além de tratar das funcionalidades dos roteadores. A arquitetura do nodo roteador e sua NI com os módulos de monitoração são ilustrados na Figura 4.7.

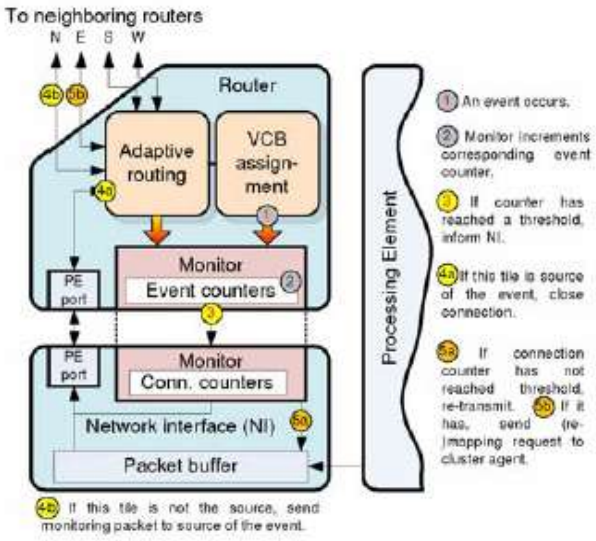


Figura 4.7 – Arquitetura do roteador proposto em [ALF08].

Os eventos monitorados são aqueles que acontecem na operação dos roteadores: (*i*) *TTL-expire-event*, que ocorre quando um pacote ultrapassa o número máximo de roteadores pelos quais ele passou; (*ii*) *No-route-found-event*, que ocorre quando o algoritmo de roteamento falha ao procurar rotas disponíveis; (*iii*) *No-buffer-event*, que ocorre quando o árbitro não encontra um VCB para armazenar o pacote que chegou, sendo o mesmo descartado pela rede; (*iv*) *Buffer-full-event*, que ocorre quando o árbitro já estabeleceu uma conexão a um VCB, mas não consegue inserir pacotes pelo fato do mesmo estar cheio, o que resulta em congestionamento na rede, diferentemente do descarte que ocorre em (*iii*).

Para transmissão de dados de monitoração, o monitor de tráfego comunica-se com a NI conectada ao seu roteador, utilizando o enlace da porta local, a qual pertence à rede de comunicação de dados regulares. Portanto, ocorre a competição de tráfego de

monitoração com tráfego de dados pela utilização de recursos da porta local. Para minimizar o impacto de desempenho causado pelos pacotes de aplicação, atribui-se maior prioridade aos pacotes de monitoração. É importante destacar, ainda, que pelo sistema ser dirigido a eventos, pacotes de monitoração só são gerados quando ocorre algum dos problemas acima listados.

[MAR10] propõe uma estrutura de monitoração de tráfego para MPSoCs baseados em NoC. Tal estrutura é integrada ao microkernel do processador que controla o fluxo de execução do MPSoC HeMPS [WOS07]. Cada porta de entrada de um roteador pertencente à NoC possui um monitor conectado, que realiza a contagem da quantidade de *flits* que chegam. Uma janela de tempo é definida para a atualização destes valores em um registrador, que por sua vez tem seu valor capturado por um módulo Gerador de Pacotes de Controle (GPC), conectado ao roteador. Desta forma, de acordo com uma certa periodicidade definida no microkernel, os resultados de monitoração são enviados pelo GPC a um MSA centralizado, conectado ao processador mestre do MPSoC. A Figura 4.8 ilustra a arquitetura interna de um roteador central com os monitores (cada um rotulado como *M*) e o módulo GPC.

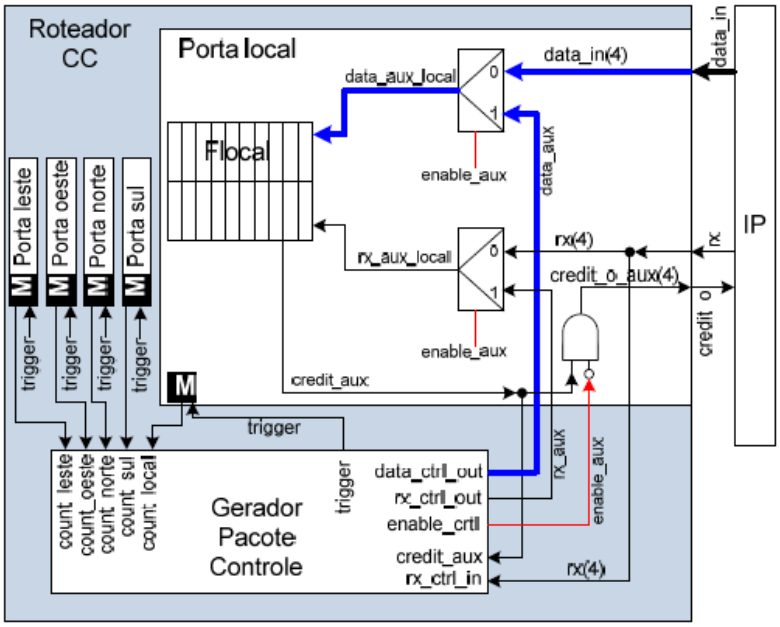


Figura 4.8 – Arquitetura do monitoração proposta em [MAR10].

[MOR10] propõe a infra-estrutura MoNoC, apresentada na Figura 4.9, composta por monitores e interfaces de rede, onde é realizado roteamento adaptativo na origem do fluxo. A MoNoC realiza a monitoração da ocupação dos canais de comunicação entre origens e destinos, sem a necessidade de um gerenciamento centralizado. Isto é possível a partir da funcionalidade das interfaces de rede, que disponibilizam serviços, que podem ser contratados levando em consideração o nível de QoS exigido pela aplicação. Monitores distribuídos pela rede são conectados às portas de saída dos roteadores da MoNoC, observando a ocupação de cada canal da rede, em tempo de execução. Os dados coletados permitem a avaliação da utilização dos canais, obtendo-se desta forma o nível de congestionamento de um dado caminho.

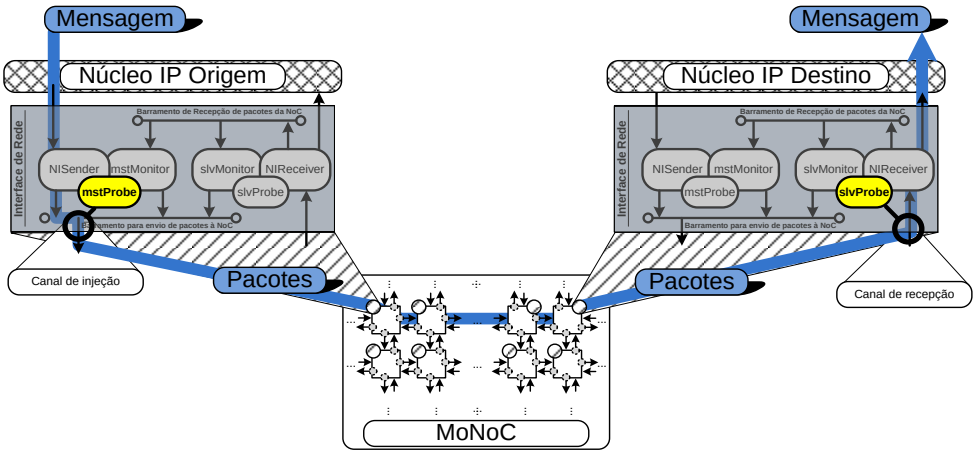


Figura 4.9 – Arquitetura de comunicação MoNoC, apresentada em [MOR10].

4.6 SISTEMA DE MONITORAÇÃO PROPOSTO

Esta Sessão apresenta o método de monitoração proposto para esta Tese. Inicialmente são abordados aspectos do protocolo utilizado, onde são descritos os processos executados nas interfaces de rede externas e nos roteadores. São desta forma apresentados os formatos de pacotes utilizados e como são estabelecidas e encerradas sessões de monitoração. Ao final desta sessão é apresentado o roteador HERMES-PLR, que é uma extensão do roteador HERMES, proposto por [MOR04]. Os blocos da arquitetura HERMES-PLR, assim como a sua validação compõem a última parte deste Capítulo.

4.6.1 PROCESSO DE MONITORAÇÃO E DETECÇÃO DE CONGESTIONAMENTO

De acordo com o que se abserva em publicações relacionadas à monitoração, os tráfegos monitorados normalmente possuem requisitos de QoS. Desta forma, devem ter periodicamente seu comportamento observado para verificação dos níveis de QoS atingidos e, se necessário, tratar os fluxos que não obedeçam aos requisitos. Fluxos QoS são aqueles com requisitos de entrega em algum parâmetro, podendo ser vazão, latência, jitter ou prazos (*deadlines*) de chegada.

4.6.1.1 TIPOS DE PACOTES

São adotadas neste trabalho, quatro classes de pacotes: SREQ, DATA, ALARM e CLEAN. Todo pacote que trafega na rede possui um *header*, que possui dois *flits*, e, dependendo do pacote, *flits* de *payload* (dados úteis para aplicação que executa no IP destino). Necessariamente o primeiro *flit* de um pacote indica o seu caminho até o roteador destino (campo PTT – *Path To Target*). Cada direção pode assumir os valores {N (norte), S (sul), E (leste), W (oeste)}, sendo utilizados 2 bits para cada hop. Quando a direção corrente é a oposta à última direção tomada, significa que a porta local é a que deve ser acessada. O processamento do roteamento é descrito na Seção 4.6.2.3. O segundo *flit* de cada pacote é de controle, e possui em seus dois primeiros bits o campo

PT (*Packet Type*), identificando o seu tipo. Outro importante dado é o tamanho do *flit*, fixo em 32 bits.

Os fluxos de aplicações que geram tráfego são compostos por mensagens, que por suas vez são quebradas em pacotes. Todo fluxo QoS deve iniciar com uma requisição de comunicação, sendo transmitida por um pacote do tipo SREQ. Um pacote SREQ (*Session REQuest*) contém três *flits* (Figura 4.10). O primeiro é o caminho até o destino. O segundo *flit* contém as seguintes informações:

- PT – tipo do pacote, configurado em 00;
- BOM – (Begin Of Message), que indica o inicio de uma mensagem;
- EOM – (End Of Message), que indica o fim de uma mensagem;
- IDENT – informa ao roteador intermediário a sua identificação no caminho do fluxo
- Sx – coordenada x do endereço do nodo origem
- Sy – coordenada y do endereço do nodo destino
- F – Bit de falha no estabelecimento da comunicação.

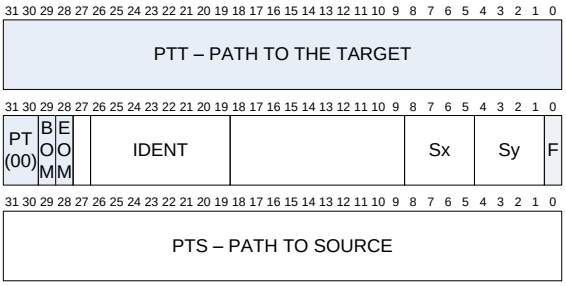


Figura 4.10 – Formato de um pacote SREQ.

O terceiro *flit* é o PTS (Path To Source), que indica o caminho percorrido para o estabelecimento da comunicação, sendo utilizado para envio do pacote de confirmação ou não de abertura de sessão.

Pacotes do tipo DATA são aqueles que levam os dados das aplicações. Tais dados podem ter ou não requisitos de QoS, o que é especificado no bit 0 do *flit* de controle. Desta forma, pacotes com requisitos de QoS tem este bit em 1. Pacotes BE, no entanto, são aqueles que são encaminhados pela rede seguindo a regra de melhor esforço, sem possuir, portanto, restrições de qualidade de serviço. Estes pacotes vão sempre ser encaminhados pelo mesmo caminho, ou seja, não terão sua rota adaptada em caso de congestionamento. Pacotes DATA BE tem o bit ‘0’ configurado em ‘0’.

O header de um pacote do tipo DATA (Figura 4.11) é composto por dois *flits*. O primeiro *flit* é o caminho até o destino. O segundo *flit* é constituído pelos campos listados a seguir. *Flits* de payload com os dados da aplicação completam o pacote.

- PT – tipo do pacote, configurado em 01;
- BOM – (Begin Of Message), que indica o inicio de uma mensagem;
- EOM – (End Of Message), que indica o fim de uma mensagem;

- IDENT – informa o roteador intermediário que deve inserir informações monitoradas no pacote;
- OCUP – campo preenchido pelo roteador escolhido para inserir no pacote informações de monitoração;
- Sx – coordenada x do endereço do nodo origem;
- Sy – coordenada y do endereço do nodo origem;
- Q – Bit que indica se o pacote é um pacote com restrições de QoS. Somente pacotes com bit QoS em 1 terão seu tráfego monitorado pelos roteadores intermediários. Desta forma, o roteador ao detectar um pacote BE não fará o processamento dos campos BOM, EOM, IDENT, OCUP, Sx e Sy.

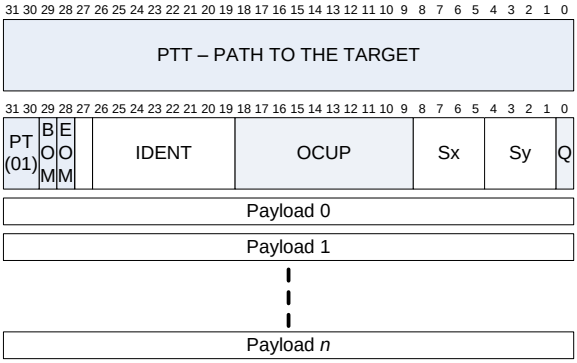


Figura 4.11 – Formato de um pacote DATA.

Pacotes do tipo ALARM (Figura 4.12) possuem três funções. A primeira é sinalizar o estabelecimento ou não de uma sessão de tráfego, requisitada no envio de um pacote SREQ. A segunda é sincronizar o envio de mensagens, de modo a tornar seu recebimento em ordem. A terceira é informar à origem do tráfego os pontos do caminho do fluxo que estão congestionados. Um pacote ALARM é composto por dois *flits*, sendo o primeiro o caminho até o destino. O segundo *flit* é composto pelos seguintes campos:

- PT – tipo do pacote, configurado em 10
- BOM – (Begin Of Message), que indica o inicio de uma mensagem, sendo configurado em 1;
- EOM – (End Of Message), que indica o fim de uma mensagem, sendo configurado em 1;
- AS – ALARM Status, que indica como vai ser interpretado o pacote, podendo assumir os valores 00 (sessão estabelecida), 01 (sessão não estabelecida), 10 (ALARM vazio) e 11 (ALARM não vazio).
- CONG – carrega o estado de congestionamento do caminho do fluxo tratado por este pacote, onde cada bit indica se um hop do caminho está congestionado.

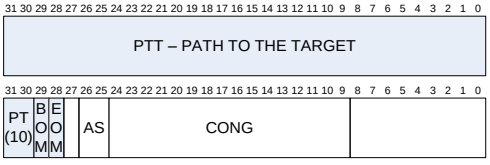


Figura 4.12 – Formato de um pacote do tipo ALARM.

Pacotes do tipo CLEAN são aqueles enviados para limpar registros em tabelas RCT, localizadas em roteadores pertencentes a caminhos que tiveram congestionamento detectado. Os pacotes CLEAN podem ser utilizados em duas situações: (*i*) se no momento de estabelecimento de sessão houverem roteadores congestionados, sendo enviado ao destino com o objetivo de limpar as tabelas RCT que foram inicializadas pelo fluxo; (*ii*) no momento de troca de caminho, para limpar as tabelas RCT do caminho antigo. Para todo pacote CLEAN enviado pela origem, haverá o recebimento um pacote CLEAN confirmando a limpeza do caminho. Pacotes do tipo CLEAN possuem os seguintes campos:

- PT – tipo do pacote, configurado em 11
- BOM – (Begin Of Message), que indica o inicio de uma mensagem;
- EOM – (End Of Message), que indica o fim de uma mensagem;
- RA – (Request Acknoledge), que indica se é realizada uma requisição de limpeza do caminho (10 – enviado da origem para o destino do tráfego QoS), ou a confirmação da limpeza (01 – enviado do destino para a origem do tráfego QoS).

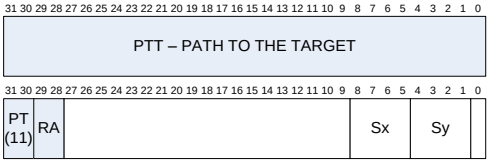


Figura 4.13 – Formato de um pacote do tipo CLEAN.

4.6.1.2 PROTOCOLO DE MONITORAÇÃO

Esta seção apresenta as estruturas de dados utilizadas no processo de monitoração de fluxos QoS proposto neste trabalho. Em seguida é mostrado o protocolo de troca de pacotes necessário durante as fases de estabelecimento de sessão, envio de dados da aplicação, notificação de congestionamento e limpeza de tabelas. Com o intuito de armazenar informações de congestionamento de fluxos QoS, três tabelas são utilizadas neste trabalho:

- **SCT** (*Source Congestion Table*), inserido na NI do IP fonte, contendo, para cada hop, 3 campos: *numero do hop*; *caminho*, que identifica a porta de saída escolhida em cada hop; *cong*, que indica se o hop está congestionado ou não.
- **RCT** (*Router Congestion Table*), disponível em todos os roteadores da NoC, possuindo um conjunto de entradas, sendo uma para cada fluxo QoS que passa através do roteador. Cada entrada armazena: *status* do registro (1 bit), *identificação* do fluxo (8 bits), o número do *hop* (4 bits), e a *métrica* utilizada para detecção de congestionamento (10 bits).
- **TCT** (*Target Congestion Table*), inserido na NI do IP destino. Armazena o nível de congestionamento de cada roteador do caminho fonte-destino.

A Figura 4.14 ilustra um exemplo detalhando uma sessão de monitoração para um fluxo denominado ‘X’, com roteadores intermediários rotulados de 1 a 7. O roteador destino, o qual utiliza informações armazenadas na TCT, possui conhecimento do tráfego no caminho sendo utilizado pelos pacotes DATA do fluxo ‘X’.

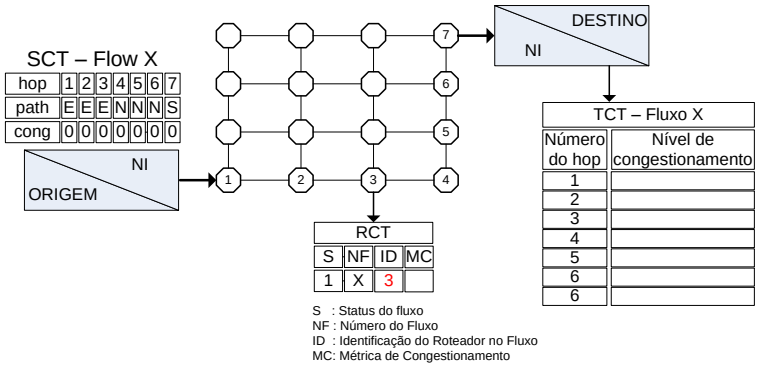
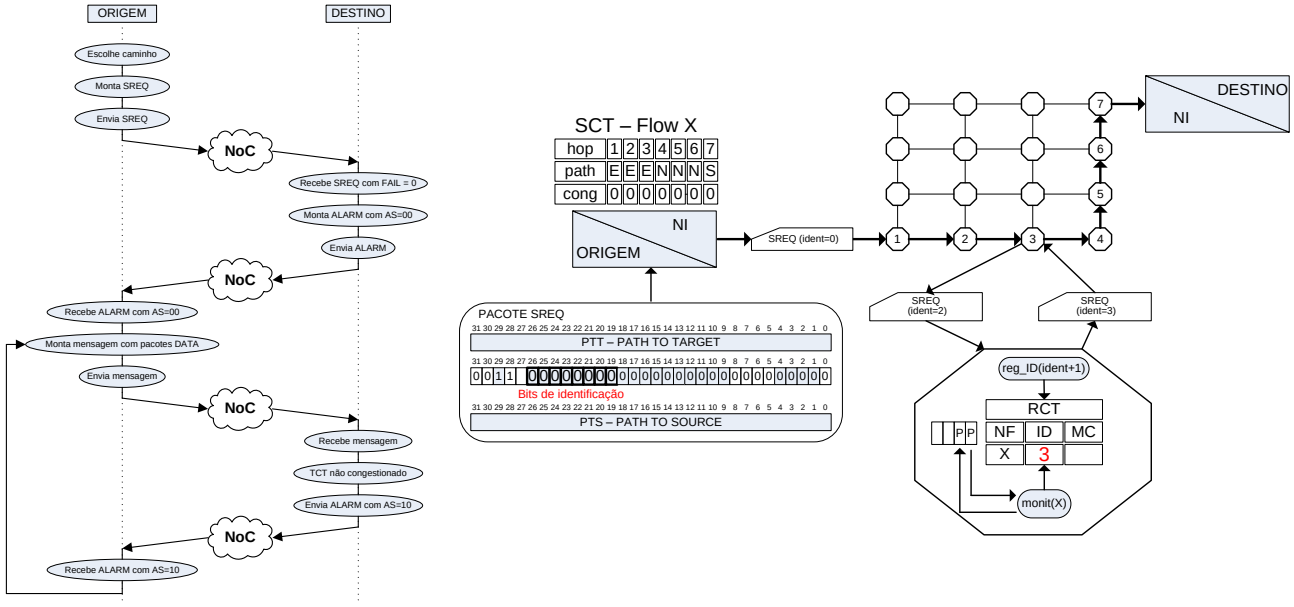


Figura 4.14 – Tabelas utilizadas no processo de monitoração de tráfego.

A Figura 4.15(a) apresenta o fluxo de ações correspondente ao estabelecimento de uma sessão de tráfego e a transmissão de mensagens, sem congestionamento na rede. Cada sessão de tráfego corresponde ao envio de uma ou mais mensagens ao IP destino, sendo as mensagens por sua vez quebradas em pacotes de tamanho constante. Embora uma sessão seja estabelecida, não há reserva de recursos, a exemplo do que acontece na técnica *circuit-switching*. Inicialmente, é escolhido um caminho, de acordo com o algoritmo *oblivious* mínimo (detalhado na Seção 5.3).



(a) (b)

Figura 4.15 – Estabelecimento de sessão e envio de dados através de um pacote SREQ.

O primeiro pacote do fluxo, do tipo SREQ, é enviado para estabelecer uma sessão de tráfego entre o par origem-destino. O pacote SREQ possui inicialmente o bit F em ‘0’, e realiza a inicialização do campo de identificação de roteadores no fluxo de todos os RCTs do caminho, e da tabela TCT, localizada no destino. É importante ressaltar que um fluxo é

identificado pelas coordenadas geográficas do roteador origem do tráfego. A Figura 4.15(b) mostra um exemplo detalhando a inicialização de um caminho a ser monitorado para um fluxo denominado ‘X’, com roteadores intermediários rotulados de 1 a 7. O roteador destino, através da utilização da tabela TCT, possui conhecimento global do caminho utilizado pelos pacotes DATA do fluxo ‘X’.

Uma vez o pacote SREQ recebido pelo destino e se bit F permanecendo em ‘0’, significa que a inicialização das tabelas foi realizada com sucesso. É então enviado um pacote ALARM, com o campo AS configurado em ‘00’. Uma vez este pacote recebido na origem, é ativado o envio das mensagens que contém os dados da aplicação. Ao final do envio de cada mensagem, o destino envia um pacote ALARM, o qual possui o campo AS configurado em ‘10’. Observar que a granularidade para a troca de caminho é a mensagem e não o pacote. Assim, ao final de uma dada mensagem, podem haver um ou mais roteadores congestionados.

Cada pacote DATA pertencente à mensagem é transmitido com um valor diferente para o campo *ident* (que varia de 1 até o número de hops do caminho). Quando um roteador do caminho recebe um pacote DATA, ele verifica o campo *ident*, e se o mesmo está de acordo com o seu endereço, a sua informação de nível de congestionamento é inserida no campo *ocup* do pacote. A informação inserida corresponde ao parâmetro utilizado para avaliação do cumprimento do requisito de QoS, podendo ser, por exemplo a quantidade de *flits* armazenados no *buffer* de entrada, a taxa de encaminhamento dos dados pelo roteador, ou mesmo o tempo gasto por pacotes para atravessar o roteador.

A Figura 4.16 ilustra a requisição para capturar a o estado de congestionamento do roteador 3 (*ident*=3). A ocupação em *buffer* é o parâmetro MC (*Métrica de Congestionamento*) escolhido para o exemplo. O roteador 3, desta forma, possui para o fluxo ‘X’ valor de ocupação de 2 *flits*. Este valor é então encaminhado para o roteador destino através do campo *ocup*. Quando o pacote DATA chega ao roteador destino, o conteúdo do campo ocup é inserido no TCT. O TCT, dimensionado de acordo com o número de hops do caminho, é endereçado pelo campo ident. Como mostrado na figura, a tabela possui 7 entradas, cada uma correspondendo aos 7 hops do caminho do fluxo ‘X’. Nota-se a atualização da terceira entrada da tabela.

Se durante a fase de estabelecimento de sessão o pacote SREQ recebido pelo destino do tráfego tiver seu valor de F em ‘1’, significa que um ou mais roteadores do caminho não puderam inicializar suas tabelas RCT. Um dado roteador não pode receber um novo fluxo QoS quando sua RCT estiver cheia (condição expressa nos bits de *Status* das tabelas). A Figura 4.17 mostra o diagrama de seqüência que ilustra as ações a serem executadas quando o fluxo não pode ser inicializado. Neste caso, o destino monta e envia à origem um pacote ALARM com campo AS configurado em ‘01’. Após o recebimento deste pacote, a origem envia um pacote CLEAN com RA em ‘10’, requisitando o processo de limpeza das tabelas RCT. Este pacote pelo destino é então recebido e devolvido à origem com RA em “01”. Finalmente, após o recebimento deste pacote, a origem escolhe de acordo com o algoritmo *oblivious* mínimo o próximo caminho.

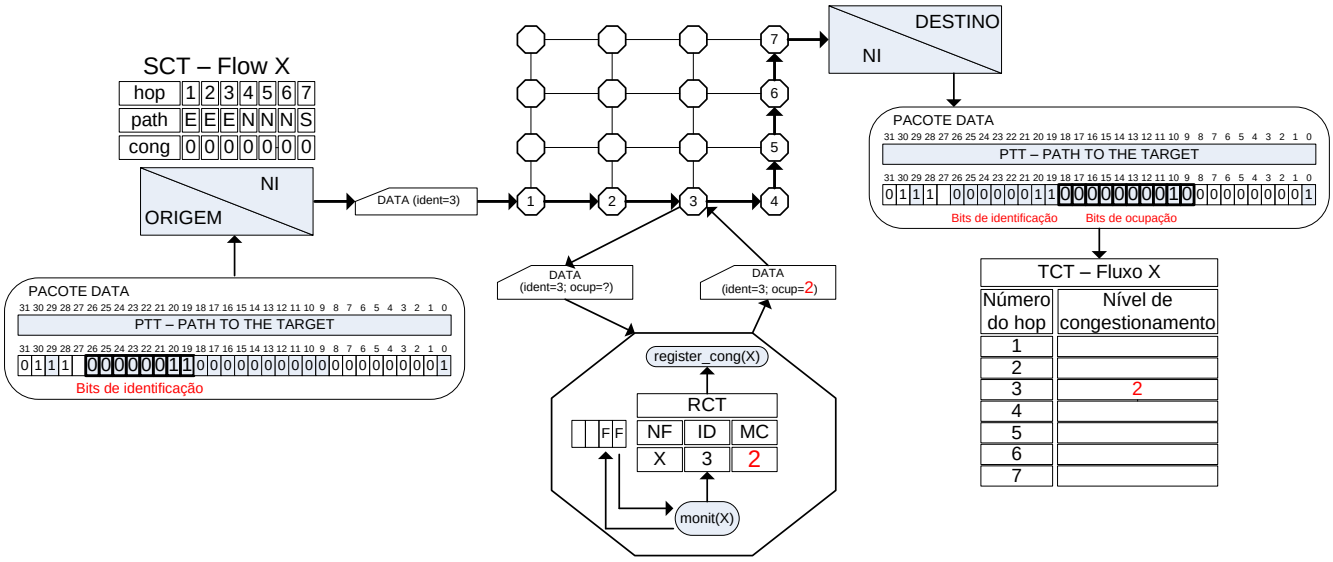


Figura 4.16 – Registro de um valor de congestionamento monitorado. O roteador em questão é o de número 3, sendo o seu valor de congestionamento (2) atualizado no TCT.

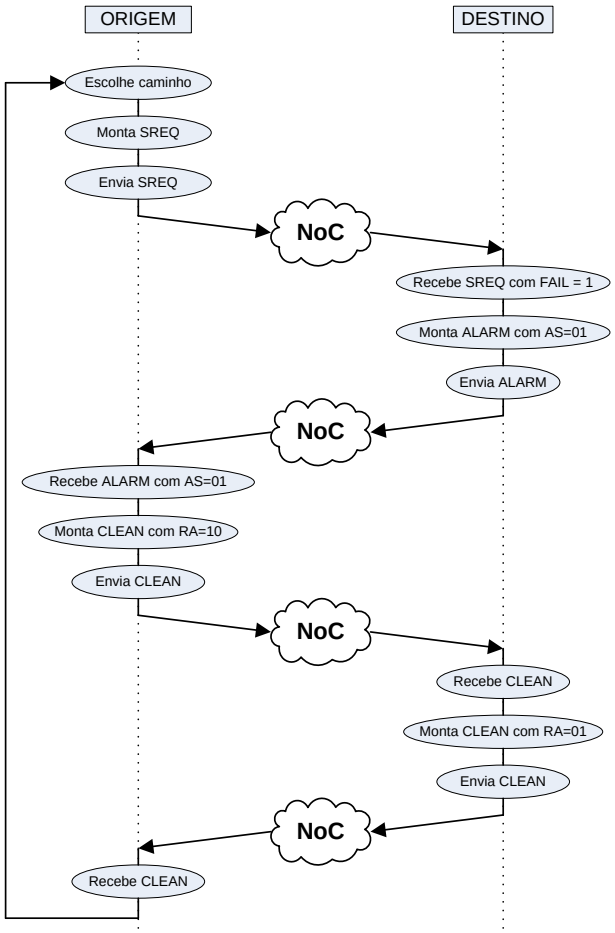


Figura 4.17 – Ações executadas quando da falha de estabelecimento de uma sessão.

A Figura 4.18 mostra o fluxo executado quando ocorre a detecção de congestionamento no caminho. Ao final da transmissão de cada mensagem, a NI do roteador destino compara os níveis de congestionamento encontrados na TCT a um dado

limiar (*threshold*), previamente definido de acordo com os requisitos de QoS da aplicação que gera o fluxo. Um pacote de ALARM é montado com AS=“11”, indicando que seu campo de congestionamento não está vazio. Um pacote ALARM que indica congestionamento contém o endereço (número do hop) dos roteadores congestionados. Quando o IP origem do tráfego QoS recebe este tipo de pacote ALARM, ele deve primeiramente enviar um pacote CLEAN para apagar os registros nas tabelas RCT dos roteadores do caminho congestionado. Para isso, são preenchidos os campos *Sx* e *Sy* do pacote, identificando o fluxo cujas entradas devem ser eliminadas. Após o recebimento da resposta de CLEAN, a NI do roteador destino computa um novo caminho, de acordo com o algoritmo apresentado no Capítulo 5.

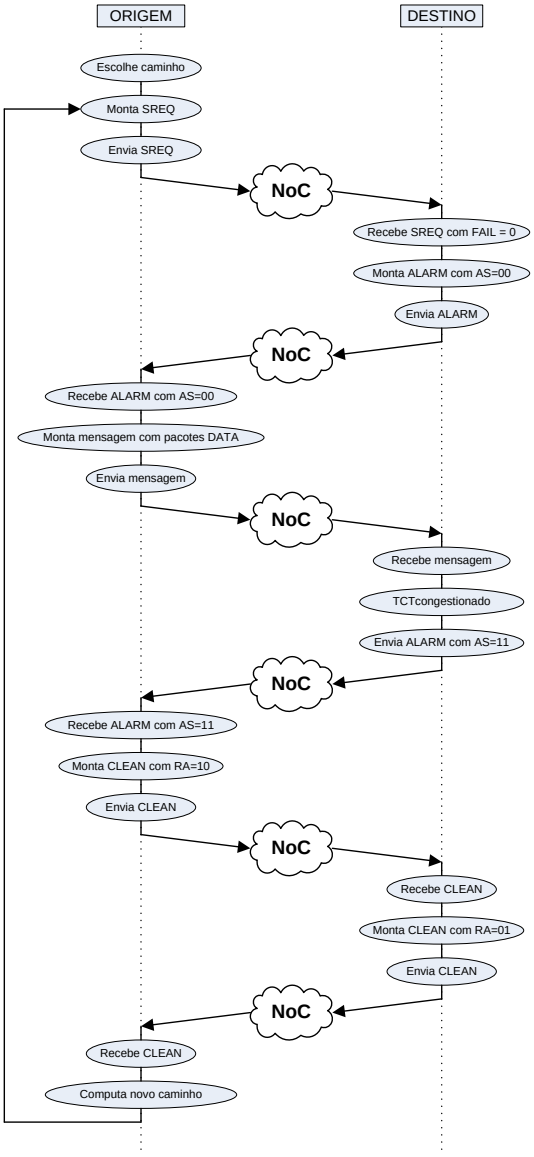


Figura 4.18 – Diagrama de sequência executado quando da detecção de congestionamento no caminho de um fluxo.

Como exemplo ilustrativo, considere a NI destino (roteador 7) na Figura 4.19. É possível observar que um nível de congestionamento igual a 5 é atingido nos roteadores 2, 3 e 6. Neste momento, um novo pacote ALARM é configurado com o endereço dos três roteadores congestionados.

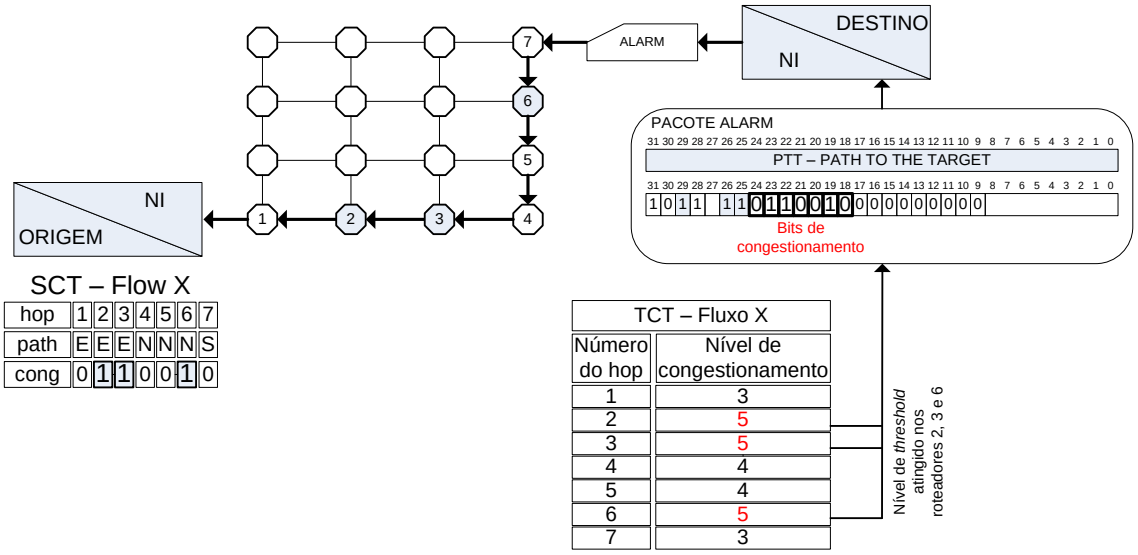


Figura 4.19 – Pacote do tipo ALARM sendo gerado, informando em seus bits de congestionamento que os roteadores 2, 3 e 6 estão congestionados.

O segundo *flit* do pacote ALARM no exemplo é detalhado na Figura 4.20. Como os roteadores 2, 3 e 6 são os que estão congestionados, o segundo, o terceiro e o sexto bits da parte do pacote que informa congestionamento são configurados em ‘1’. A NI do roteador origem atualiza o campo *cong* da tabela SCT.

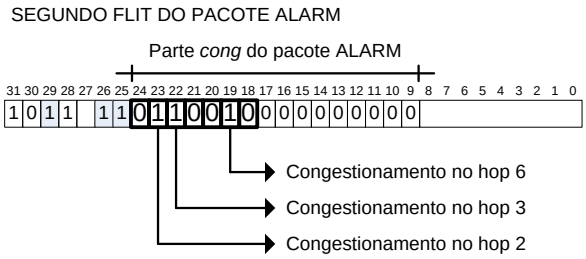


Figura 4.20 – Pacote ALARM indicando roteadores congestionados.

4.6.2 O ROTEADOR HERMES–PLR

Esta Seção apresenta o roteador Hermes-PLR (*Path Load Routing*), que executa o protocolo de monitoração descrito neste Capítulo. Este roteador é uma versão estendida da NoC HERMES [MOR04], tendo como principais características: (*i*) roteamento na origem; (*ii*) método de chaveamento *wormhole* (*iii*) *buffers* na entrada; (*iv*) controle de fluxo baseado em créditos.

4.6.2.1 ASPECTOS ARQUITETURAIS

A Figura 4.21 mostra a interface de comunicação entre roteadores vizinhos na NoC Hermes-PLR. Este roteador apresenta adaptações em relação à NoC Hermes-VC [MEL06], adicionando-se ao roteador desta arquitetura suporte ao método de monitoração apresentado neste trabalho, além de roteamento na origem.

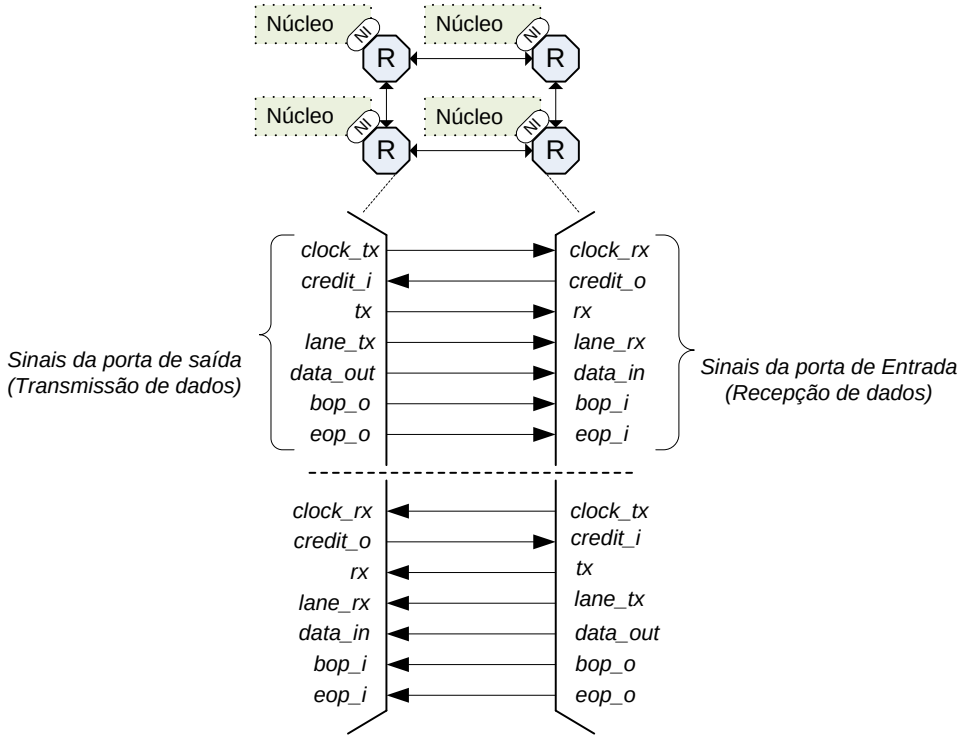


Figura 4.21 – Sinais do roteador Hermes-PLR.

Como explicado no Capítulo 2, mensagens são quebradas em pacotes, para viabilizar a sua transmissão em um meio de comunicação como a NoC. Cada pacote é ainda dividido em unidades chamadas *flits*, sendo esta a unidade de informação transmitida entre os roteadores. O sinal *clock_tx* sincroniza a transmissão de dados. Para transmissão de um *flit*, é necessário primeiramente verificar se o roteador vizinho possui espaço em seu *buffer*. Esta condição é observada na linha *credit_i*, que indica se há pelo menos uma posição em *buffer* disponível no roteador vizinho para armazenar o *flit*. O próximo passo é notificar ao roteador vizinho a disponibilidade do *flit*, através do sinal *tx*. O sinal *lane_tx* indica em qual canal virtual o *flit* está sendo transmitido. O sinal *data_out* contém o *flit* transmitido. Os sinais *bop_in* e *eop_in* indicam se o *flit* é de início ou fim de pacote, respectivamente. Estes sinais são fundamentais, por serem utilizados como referência na coordenação da operação da máquina de estados pertencente ao módulo que gerencia a entrada de *flits* no roteador receptor.

O roteador Hermes possui uma lógica de controle de chaveamento centralizada e até 5 portas bidirecionais: *East*, *West*, *North*, *South* e *Local*. A porta Local estabelece a comunicação entre o roteador e seu núcleo. As outras portas são conectadas aos roteadores vizinhos. A Figura 4.22(a) apresenta a arquitetura do roteador Hermes-PLR. Cada porta de entrada é conectada a um *buffer* de *flits* (B). Este *buffer* tem duas funções. A primeira é o armazenamento temporário de *flits*, que estão bloqueados pelo fato de ainda não terem permissão do módulo de arbitragem para serem transmitidos. A segunda é analisar o conteúdo de *flits* de header, de modo a coordenar a operação entre os módulos de arbitragem e RCT.

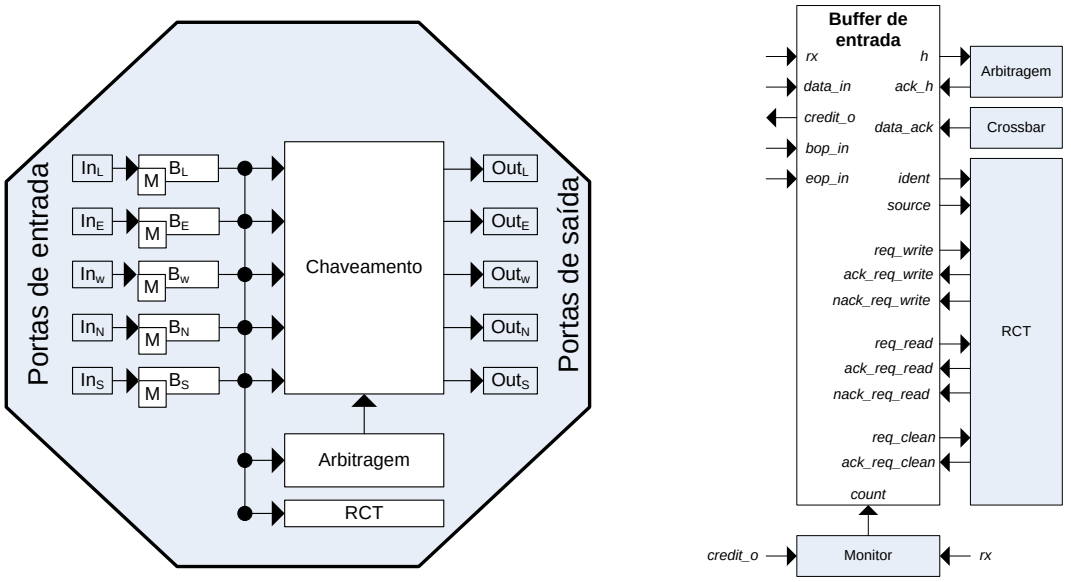


Figura 4.22 – Estrutura do roteador Hermes-PLR.

Conectado a cada *buffer* existe um módulo de monitoração (M), que tem a função de contar quantos *flits* entram no roteador, dentro de uma janela de tempo. Estes dados são utilizados posteriormente para avaliação do estado de congestionamento do roteador, pelo módulo conectado à tabela TCT, localizada no destino do tráfego. O módulo de arbitragem/roteamento tem a função de escalonar a porta cujos dados serão encaminhados, e de definir a porta de saída dos pacotes. O módulo RCT armazena dados do fluxo da sessão de tráfego que está sendo monitorada.

A Figura 4.22(b) detalha a interface entre o módulo de *buffer* de uma porta de entrada com os demais módulos da NoC. Os sinais *rx*, *data_in*, *credit_o*, *bop_in* e *eop_in* são sinais conectados a porta de entrada do roteador, explicados acima. O sinal *h* requisita roteamento, sendo ativado quando um *flit* de *header* chega ao roteador. O sinal *ack_h* é ativado pelo módulo de roteamento/arbitragem se a requisição é atendida, habilitando a transmissão dos *flits* restantes do pacotes. O sinal *data_ack* indica que um *flit* foi transmitido pela porta de saída. A ativação de um *data_ack* em combinação com o fim de um pacote faz com que o módulo de *buffer* volte ao seu estado inicial, onde é esperado um novo pacote. Os sinais *ident* e *source* são utilizados quando um fluxo QoS requisita conexão. O sinal *ident* expressa a identificação do roteador no fluxo QoS corrente. O sinal *source* armazena a origem do fluxo.

As requisições para leitura e escrita na tabela RCT são realizadas através dos sinais *req_read* e *req_write*, respectivamente. O sucesso ou não das requisições são expressas nos sinais *ack_req_write*, *nack_req_write*, *ack_req_read*, *nack_req_read*. O sinal *count* armazena o número de *flits* contados pelo módulo *Monitor*, sendo este valor armazenado no campo *ocup* de um pacote QoS, quando o valor de *ident* for o mesmo do valor do roteador atual do fluxo. Observa-se que, nesta implementação, não há o campo *ocup* na tabela RCT. Com isto, o custo do roteador em área reduz, ao passo que apenas valores instantâneos do contador dos pacotes são armazenados no campo *ocup* dos pacotes. Finalmente, os sinais *req_clean* e *ack_req_clean* representam respectivamente

requisição de limpeza e confirmação de limpeza da tabela RCT, na entrada especificada no sinal *source*.

O módulo RCT gerencia a operação da tabela que armazena propriedades das sessões QoS. O módulo é central, mas contém um processo de gerenciamento para cada porta de entrada. Uma tabela RCT possui os seguintes campos: (*i*) *status* (1 bit), que informa se uma dada linha da tabela está ocupada ou não; (ii) *origem* (8 bits), que informa a identificação do roteador conectado ao núcleo que originou o fluxo QoS; (*iii*) *hop* (4 bits), que informa a identificador do roteador no caminho.

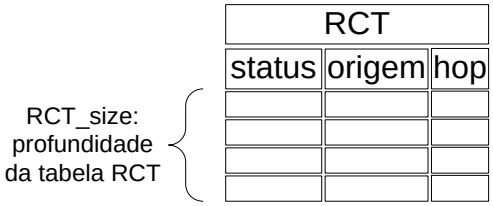


Figura 4.23 – Tabela RCT.

Três operações podem ser realizadas sobre esta tabela: escrita, leitura e limpeza. No processo de *escrita*, o roteador é configurado para receber o fluxo QoS ou para ter sua operação desabilitada, quando o fluxo QoS for encerrado ou não puder ser inicializado. Quando é detectado que o pacote recebido pelo roteador é do tipo SREQ, o *buffer* do roteador entra em um estado de tentativa de escrita na tabela RCT. O sinal *req_write* é então colocado em '1'. De acordo com a origem do tráfego, é verificado qual a posição da tabela será ocupada.

A Figura 4.24 ilustra a computação da posição na tabela RCT. Para evitar o processo de busca na tabela, e lógica adicional para cálculo de endereços, utiliza-se uma técnica de *hashing*. O índice de acesso à RCT é denominado *index*. Inicialmente é realizada a operação *xor* bit a bit entre os quatro bits da posição *x* do roteador origem (*source_x*) e os quatro bits da posição y do roteador correspondente (*source_y*). O resultado desta operação é armazenado em uma variável temporária denominada *index_tmp*. O próximo passo é realizar uma operação xor bit a bit entre os dois bits menos significativos dois mais significativos de *index_tmp*. O resultado desta operação é armazenado em *index*.

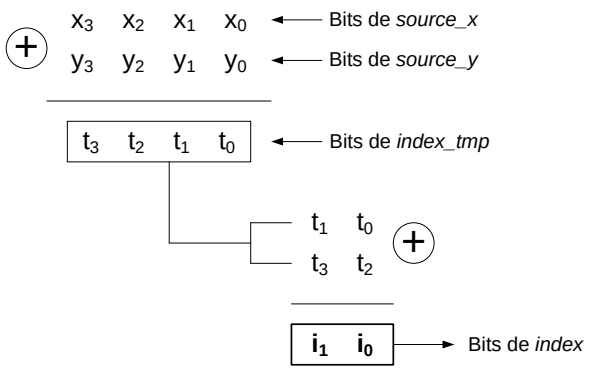


Figura 4.24 – Obtenção da entrada da tabela RCT (*index*).

Uma vez definida a posição onde devem ser armazenadas informações do fluxo, é necessário verificar se é possível criar a nova entrada, observando-se para isso o valor de *status* da tabela RCT na posição *index*. Se a posição *index* da tabela não estiver ocupada, *ack_req_write* é colocado em '1', e os valores de origem e hop são registrados. Caso a linha relativa a posição *index* já tiver sido alocada para outro fluxo, *nack_req_write* é colocado em '1' e o bit F de SREQ é também colocado em '1', notificando que pelo menos uma da tabelas RCT do caminho não pôde ser alocada ao fluxo.

No processo de *leitura*, o roteador analisa o tipo do pacote que está entrando, observando-se o seu segundo *flit*. Se este pacote for do tipo DATA, a sua origem (*source_x* e *source_y*) e o campo *ident* são passados para o módulo que controla o RCT, sendo ativado o sinal *req_read*. Se *ack_req_read* é ativado pelo módulo RCT, significa que o roteador corrente é aquele que deve inserir no pacote o seu estado de congestionamento (armazenado pelo sinal *count*). Caso *nack_req_read* seja ativado, significa que outro roteador deve inserir no pacote o seu estado de congestionamento.

O processo de *limpeza* inicia na detecção de um pacote do tipo CLEAN, sendo ativado o sinal *req_clean* com o endereço do roteador origem do tráfego QoS. A entrada *index* é calculada, sendo o bit de *status* correspondente configurado em '0'. A resposta *ack_req_clean* do módulo RCT para o *buffer* de entrada confirma a operação.

4.6.2.2 IMPLEMENTAÇÃO EM HARDWARE

Os roteadores da NoC Hermes-PLR foram descritos em VHDL no nível RTL (*Register Transfer Level*). As descrições foram sintetizadas para o FPGA xc5vlx330-2ff1760 da Xilinx. No processo de síntese utilizou-se a ferramenta XST que compõe o pacote de ferramentas ISE versão 11.4. Os resultados de área (em LUTs – *Look-Up Tables*) capturados após a síntese para um roteador central da Hermes-PLR são apresentados na Tabela 4.1. Para efeitos de comparação, são apresentados os resultados de área da NoC Hermes original, cujas configurações são as seguintes: (*i*) roteamento distribuído (*ii*) *buffers* com profundidade de 8 *flits* (*iii*) *flits* com tamanho de 32 bits; (*iv*) 2 canais virtuais.

Tabela 4.1 – Resultados de implementação em área de um roteador central da Hermes-PLR.

Hermes original		Hermes-PLR		Aumento (+) ou redução(-)?
Componente	LUTs	Componente	LUTs	
1. Roteador Central	**2.193**	1. Roteador Central	**3.181**	+ 45%
2. Porta de Entrada	270	2. Porta de Entrada	432	+ 60%
2.1. *Buffer*	141	2.1. *Buffer*	213	+ 51%
3. Arbitragem/Roteamento	405	3. Arbitragem	253	- 37.5%
4. Porta de saída	163	4. Porta de saída	205	+ 25.8%
		5. Monitor	88	
		6. Módulo de RCT	135	

O roteador central da Hermes-PLR é composto por (*i*) 5 portas de entrada, sendo que cada uma possui 2 canais virtuais e *buffer* para armazenamento de até 8 *flits*; (*ii*) 2 módulos que controlam tabelas RCT; (*iii*) módulo de monitoração; (*iii*) módulo de arbitragem; (*iv*) 5 portas de saída.

Comparando a área das duas arquiteturas, observa-se que a Hermes-PLR possui tamanho maior que a Hermes original em quase todos os componentes. Isso é esperado, considerando que: (i) a lógica para gerenciamento de *buffers* na Hermes-PLR é mais complexa, uma vez que é realizada a decodificação do tipo de pacote; (ii) as portas de saída na Hermes-PLR são constituídas cada uma de um *crossbar* completo, habilitando todos os *turns* a serem feitos, requisito necessário para o método de roteamento proposto, que pode utilizar os *turns* do WF ou do EF, dependendo da posição do nodo destino (detalhes na Seção 5.6). O fato a lógica de roteamento ser extraído da Hermes-PLR faz com que a sua parte de conexão de portas de entrada com portas de saída seja reduzida em comparação à Hermes original.

Dado que o algoritmo WF possui como característica utilizar uma dada *lane* e o EF a outra *lane*, é possível reduzir a área do roteador proposto, considerando que somente um dos canais virtuais de cada porta vai receber fluxos QoS. Desta forma é possível colocar em uma das lanes em cada porta um módulo de *buffer* que não vai realizar qualquer análise por pacotes ali armazenados, igual à rede original.

4.6.2.3 TRATAMENTO DO ROTEAMENTO NA ORIGEM

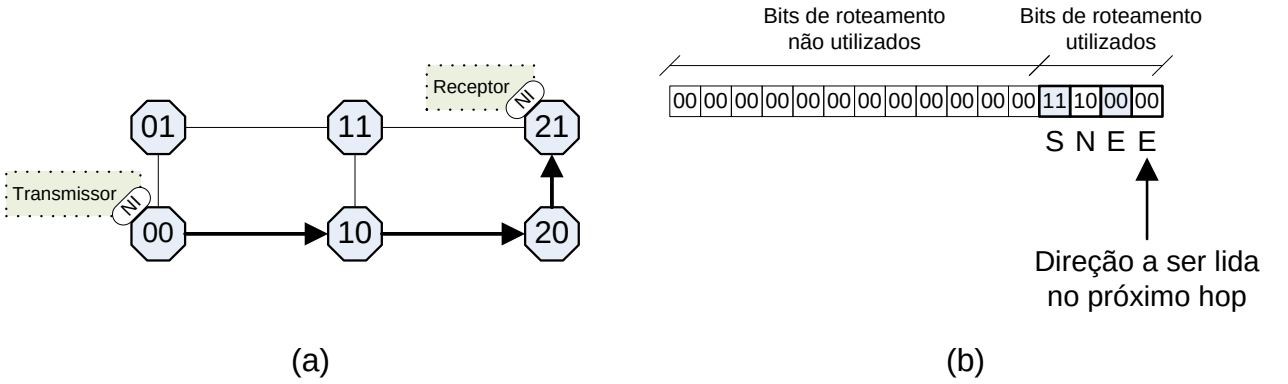
Como explicado na Seção 4.6.1.1, o sistema de monitoração de tráfego apresentado nesta Tese utiliza quatro tipos de pacotes: SREQ, DATA, ALARM e CLEAN. Os pacotes possuem duas partes: *header* e dados úteis (*payload*). O *header* é ainda dividido em dois *flits*. O *flit* de roteamento contém a rota a ser seguida pelo pacote. O segundo *flit* contém informações particulares a respeito de cada tipo de pacote, sendo estas informações detalhadas na Seção 4.6.1.1.

A rota do pacote apresenta as direções que o pacote vai tomar ao longo do caminho entre a origem e o destino. Cada direção pode assumir os valores {E (*leste*), W (*oeste*), N (*norte*), S (*sul*)}. Para codificação da direção a ser tomada são utilizados 2 bits para cada hop, de acordo com a convenção apresentada na Tabela 4.2.

Tabela 4.2 – Codificação para as possíveis direções de um caminho.

PORTA	E	W	N	S
BITS	00	01	10	11

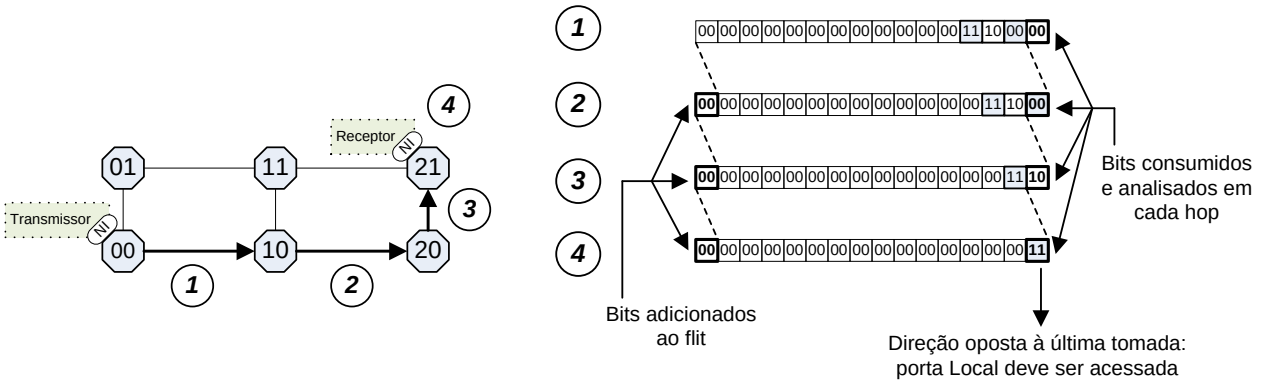
A Figura 4.25(a) mostra um exemplo de caminho, onde o roteador “00” envia dados para o roteador “21”. Se a rota for definida na origem de acordo com o algoritmo XY, o pacote é então roteado duas vezes para *leste* e uma vez para o *norte*. Em (b) é mostrado a configuração do *flit* de roteamento. Convenciona-se que a direção a ser tomada é indicada nos bits menos significativos do *flit*.



(a) (b)

Figura 4.25 – Exemplo de caminho e respectivo formato de *flit* de roteamento.

A Figura 4.26 ilustra o *flit* de roteamento e a configuração que o mesmo assume a cada *hop*, para o caminho introduzindo na Figura 4.25. Observa-se que a cada hop, os 30 bits mais significativos são deslocados dois bits à direita, sendo colocado “00” nos bits mais significativos do pacote. A porta local é a acessada se a direção atualmente analisada é oposta à última direção tomada. A última direção tomada é obtida pelo conhecimento da porta por onde entrou o *flit*. Para o exemplo, a direção *sul* indica que a porta local é a escolhida, visto que a direção norte foi a última tomada, ou seja, a direção por onde saiu o *flit* no hop anterior.



(a) (b)

Figura 4.26 – Exemplo de processamento de *flit* de roteamento.

4.6.2.4 VALIDAÇÃO FUNCIONAL DAS OPERAÇÕES DE GERENCIAMENTO DO RCT

Serão apresentados aqui os eventos que ocorrem para durante a leitura, atualização e limpeza de um registro em uma tabela RCT. O cenário exemplo ilustra a situação onde o roteador origem do tráfego QoS possui coordenadas x=0 e y=1 e o destino coordenadas x=3 e y=2 (Figura 4.27).

A Figura 4.28 mostra o comportamento dos sinais de um RCT pertencente a um roteador que possui ident=3. O roteador origem do tráfego QoS possui coordenadas x=0 e y=1. Os retângulos indicam a chegada de pacotes. O primeiro pacote que chega é de requisição de sessão (SREQ). Em ***1***, observa-se que o sinal *req_write* é então ativado para iniciar a escrita na tabela RCT. O valor de *index* em 1 é obtido substituindo-se *source_x*=0 e *source_y*=1.

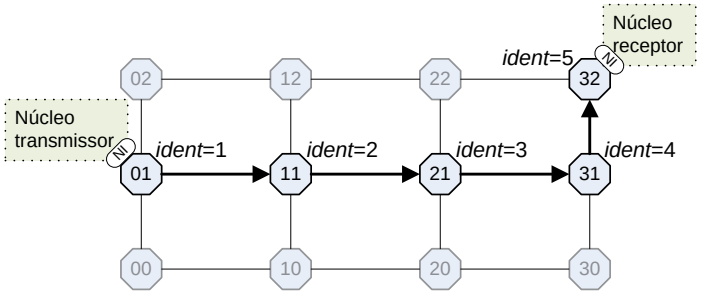


Figura 4.27 – Fluxo utilizado para ilustrar validação do módulo RCT.

Figura 4.28 – Operação de inicialização e atualização de dados utilizando a tabela RCT.

Após a confirmação dada pelo sinal *ack_req_write*, observa-se em ***2*** alteração nos bits da primeira posição da tabela RCT. O bit de status de RCT é configurado em ‘1’. O valor de *origem* é alterado para “00000001”, onde os quatro primeiros bits correspondem à posição *x* do nodo origem e quatro últimos bits correspondem à posição *y*. O campo *hop* apresenta o valor “0011” indicando que ele é o terceiro hop do caminho. Observa-se em ***3*** que o terceiro pacote que chega traz como parâmetro *hop* o valor “0011”, que é o mesmo do campo *hop* da tabela RCT. Isso faz com que a requisição de leitura seja respondida afirmativamente com *ack_req_read* em ‘1’. Notar que para os pacotes restantes, a resposta é negativa para a requisição de leitura, com *nack_req_read* em ‘1’.

A Figura 4.29 mostra os eventos que ocorrem durante um processo de limpeza. Em ***1***, observa-se a chegada de um pacote do tipo CLEAN (segundo *flit* possui valor “E0000002”). Em ***2***, o sinal req_clean é ativado e sua resposta é positivamente dada no próximo ciclo, através da ativação do sinal *ack_req_clean*. Observar que o bit de status da tabela RCT na posição 1 muda de ‘1’ para ‘0’ logo após a resposta dada por *ack_req_clean*.

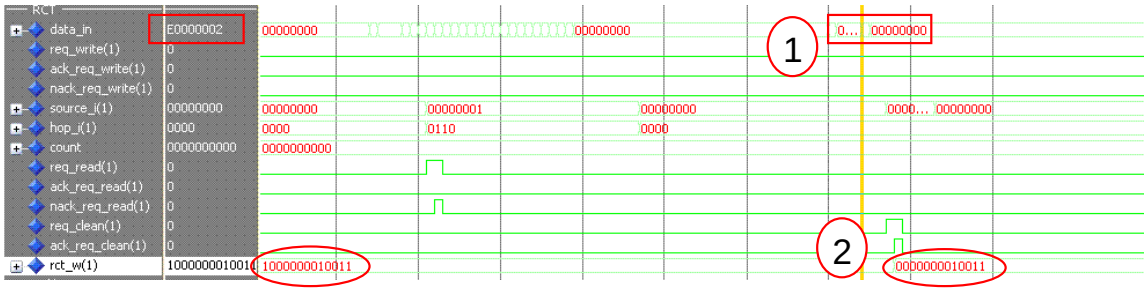


Figura 4.29 – Operação de limpeza da tabela RCT.

4.6.3 CONSIDERAÇÕES SOBRE MONITORAÇÃO DE TRÁFEGO

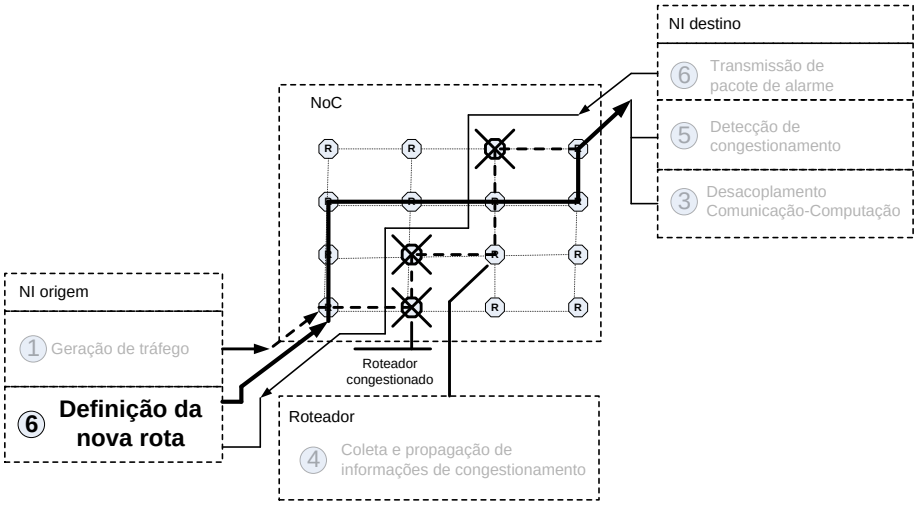
A Tabela 4.3 resume comparativamente os trabalhos relacionados à monitoração de tráfego aqui citados. Observa-se que todos os trabalhos realizam processamento on-line de informações de roteamento e colocam os *probes* nos roteadores da rede. O preenchimento de *buffers*, assim como a utilização de enlaces constitui as métricas mais consideradas nas medições. Quanto ao processamento de dados de monitoração, a abordagem centralizada é a alternativa mais utilizada. Informações são transmitidas através de canais virtuais, na maior parte dos trabalhos.

No contexto desta Tese, *probes* de monitoração são colocados em cada roteador da rede, onde as informações são capturadas em tempo de execução. Tais informações atualizam tabelas RCT, que são consultadas de acordo com o especificado em pacotes do tipo DATA. A utilização de pacotes da aplicação para transporte de informações de tráfego é um diferencial deste trabalho em relação aos correlatos do estado da arte. Tal processo é desta forma distribuído, sendo cada roteador habilitado a informar o seu estado de congestionamento. A taxa de saída de *flits* é a métrica de monitoração escolhida, pelo fato dos pacotes de estabelecimento de conexão serem gerados com requisitos de vazão da aplicação.

Tabela 4.3 – Trabalhos relacionados a algoritmos de monitoração em NoCs.

Autor	**Parâmetros monitoração da rede**	**Momento de processamento de dados de monitoração**	**Processamento de informações de monitoração**	**Estrutura de transmissão de dados de monitoração**	**Localização de probes de monitoração**
[MAR05a]	Utilização de *buffers*	On-line	Distribuído	Canais físicos dedicados	Roteadores
[BRA07]	Utilização de enlaces	On-line	Centralizado	Canal virtual de serviço garantido	Roteadores
[CIO04]	Utilização de *buffers*	On-line	Centralizado ou distribuído	Através de canais virtuais GT e/ou BE	Roteadores
[KIM07]	Latência, utilização de *buffers* e utilização de enlaces	On-line	Centralizado	Conexão ponto-a-ponto com o controlador central	Roteadores
[ALF08]	Tempo de vida de pacotes, rotas não-encontradas, indisponibilidade de *buffers* e *buffers* cheios	On-line	Distribuído	Canais virtuais	Roteadores
[MAR10]	Taxa de encaminhamento de *flits*	On-line	Centralizado	Canais de dados	Roteadores
[MOR10]	Ocupação de canais de saída em rotadores	On-line	Distribuído	Canais virtuais específicos	Roteadores
Proposta deste trabalho	**Taxa de encaminhamento de *flits***	**On-line**	**Distribuído**	**Canais de dados**	**Roteadores e interfaces de rede externas**

5. ALGORITMOS DE ROTEAMENTO ADAPTATIVOS PARA REDES MALHA



Um algoritmo de roteamento tem por objetivo estabelecer o caminho a ser seguido quando da transmissão de cada mensagem ou pacote em uma rede desde a fonte de tráfego até o destino [DUA03]. A escolha de um algoritmo de roteamento adequado é um elemento crítico no projeto de redes de comunicação de dados. Algoritmos de roteamento permitem uma melhor distribuição da carga total da rede em suas regiões, quando o padrão de tráfego não é uniforme, de forma a maximizar a vazão total das comunicações. Um algoritmo de roteamento também influencia na distância percorrida pelos pacotes, em número de *hops* (saltos de roteamento), influenciando na latência média das mensagens.

Outro aspecto importante é a capacidade do algoritmo ao trabalhar em situações de falha ou congestionamento na rede. A capacidade de adaptação de um algoritmo de roteamento aumenta a chance dos pacotes trafegarem em locais distantes de pontos quentes da rede (*hot spots*) ou com falha. Se por exemplo, uma falha ocorre em um enlace ou nodo da rede através dos quais dados são roteados, ela pode perturbar ou até mesmo impedir a operação do restante do sistema. Entretanto, se um algoritmo pode ser reprogramado para se adaptar à falha, o sistema pode continuar a funcionar com uma pequena queda de desempenho [DAL04].

Uma propriedade desejável de um algoritmo de roteamento é a capacidade do mesmo em evitar situações de *livelock* e *deadlock*. Uma situação de *livelock* ocorre quando um pacote trafega na rede sem nunca chegar ao seu destino. Tal situação ocorre quando se utiliza algoritmos de roteamento *não-mínimos*. Se não há um número de *hops* limite em que o pacote pode ser roteado sem chegar ao destino, há a possibilidade do mesmo ocupar a rede indefinidamente. A situação de *deadlock*, por outro lado, ocorre quando um grupo de pacotes permanece bloqueado, esperando recursos alocados por outros pacotes em uma configuração de dependência cíclica. Com o objetivo de evitar *deadlocks*, [GLA94] propõem regras para proibir curvas no caminho de um pacote de modo

a serem obedecidos por algoritmos de roteamento, eliminando a possibilidade da ocorrência de ciclos de espera de recursos. Tais regras constituem o que na literatura é conhecido como *turn model*.

Este capítulo apresenta inicialmente uma taxonomia acerca de algoritmos de roteamento, considerando diversos parâmetros. Posteriormente são analisados os principais algoritmos de roteamento utilizados em redes malha, com método de repasse *wormhole*. Ainda, são apresentadas algumas propostas encontradas no estado da arte empregadas em NoCs. Finalmente, é apresentado o algoritmo de roteamento proposto na presente tese, situando-o no espectro de adaptatividade composto pelos algoritmos tradicionais apresentados.

5.1 TAXONOMIA DE ALGORITMOS DE ROTEAMENTO

A Figura 5.1 [DUA03] ilustra uma taxonomia para algoritmos de roteamento, de acordo com diversos critérios, importantes para a definição das características um dado algoritmo. Cada linha contém abordagens alternativas que podem ser seguidas de acordo com cada critério.

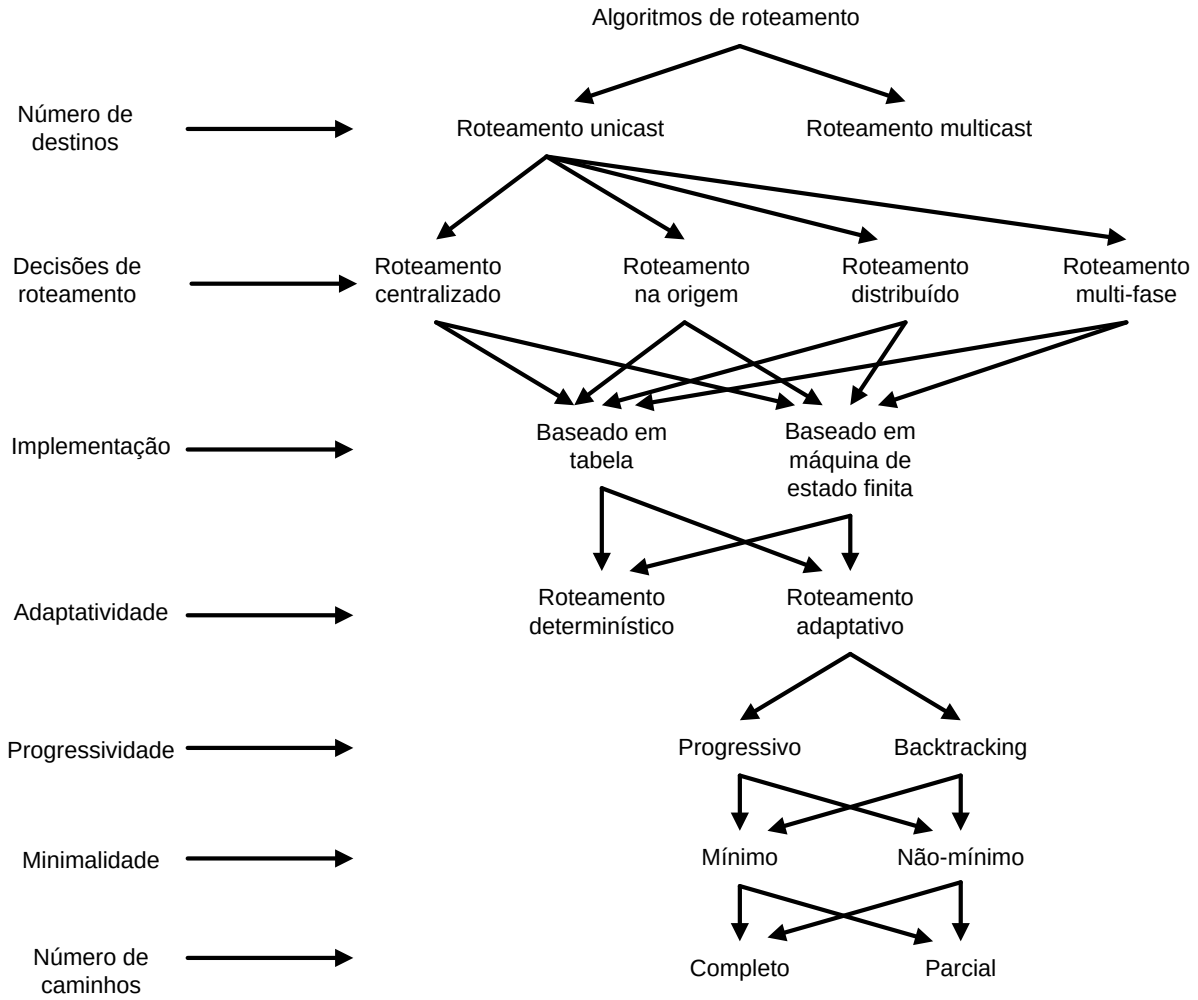


Figura 5.1 – Taxonomia de algoritmos de roteamento [DUA03].

De acordo com o número de destinos, um algoritmo de roteamento pode encaminhar cada pacote para um único destino (roteamento *unicast*) ou para múltiplos destinos (roteamento *multicast*). Algoritmos de roteamento também podem ser classificados de acordo com o local onde as decisões de roteamento são tomadas. No roteamento *centralizado* o caminho é especificado por um controlador centralizado. O roteamento é dito *na origem* quando todas as decisões de roteamento para um pacote são realizadas inteiramente na origem. Normalmente, o nodo origem tem a sua disposição uma tabela com no mínimo uma rota para cada destino. Se, dado um destino, é retornado um conjunto de rotas, uma delas é selecionada de acordo com algum critério e atribuída ao pacote. No roteamento *distribuído*, as decisões de roteamento são tomadas em cada *hop* pelo qual passa o pacote. O roteamento multi-fase é um esquema híbrido, onde o nodo fonte computa alguns nodos intermediários. O caminho até estes nodos é definido de forma distribuída, podendo o pacote ser encaminhado para todos os nodos intermediários computados (roteamento multicast) ou apenas para o último nodo destino (roteamento *unicast*).

A implementação de um algoritmo de roteamento também pode adotar basicamente duas estratégias. No roteamento baseado em *tabelas*, cada *hop* da rede possui a relação entre as fontes e destinos de tráfego, ou seja, qual a porta de saída para um pacote que aparece em uma dada porta de entrada. No roteamento baseado em uma *máquina de estados finita*, um algoritmo é executado para definir o caminho para cada pacote a ser roteado. Nestes dois casos o algoritmo de roteamento pode ser *determinístico* ou *adaptativo*. No roteamento determinístico, o caminho a ser percorrido pelo pacote é completamente definido pelos seus endereços fonte e destino. No roteamento adaptativo, informações sobre o estado do tráfego na rede são consideradas para decidir como o pacote vai ser encaminhado.

Algoritmos adaptativos são classificados também de acordo com a sua progressão na rede. Algoritmos *progressivos* movem o cabeçalho adiante, reservando um novo canal em cada operação de roteamento. Algoritmos de *backtracking* permitem ao cabeçalho do pacote retornar pelo caminho reverso desabilitando canais anteriormente reservados. De acordo com a minimalidade, um algoritmo de roteamento adaptativo pode ser classificado como *mínimo* ou *não-mínimo*. Em um algoritmo *mínimo* somente são considerados canais que levam o pacote mais próximo ao seu destino. Esta obrigatoriedade não ocorre em um algoritmo *não-mínimo*, podendo-se neste caso o pacote ser roteado para um canal que o leve para mais longe de seu destino. Finalmente, um algoritmo de roteamento adaptativo pode ser classificado em função do número de caminhos, podendo ser *completamente adaptativo* ou *parcialmente adaptativo*. É possível ter-se, por exemplo, um algoritmo completamente adaptativo que possa escolher entre todos os caminhos mínimos possíveis (*completamente adaptativo mínimo*). Por outro lado, algoritmos parcialmente adaptativos tentam aliar a flexibilidade dos completamente adaptativos com a baixa complexidade de algoritmos determinísticos.

5.2 ALGORITMOS DE ROTEAMENTO PARCIALMENTE ADAPTATIVOS

Esta Seção apresenta uma descrição de algoritmos de roteamento para redes malha, freqüentemente utilizados como referência de comparação quando da criação de novos algoritmos. Estes algoritmos foram propostos no trabalho seminal de Glass e Ni [GLA94], onde são descritos como baseados no modelo *turn model*. Este modelo oferece uma abordagem sistemática para o desenvolvimento de algoritmos de roteamento adaptativos, tanto mínimos quanto não mínimos, para alguns tipos de topologia de rede. O objetivo é evitar a ocorrência de *deadlock*, que, como dito anteriormente, é um evento causado por pacotes que esperam por recursos utilizados por outros pacotes, em ciclo. A Figura 5.2 ilustra um exemplo de ocorrência de *deadlock*. O fluxo *A* ocupa os canais *u* e *v*, mas não pode progredir enquanto não puder utilizar o canal w. Ao mesmo tempo, o fluxo *B* ocupa os canais *w* e *x*, mas não pode progredir enquanto não puder utilizar o canal *u*. Nenhuma das conexões é liberada por um fluxo para beneficiar o outro. Portanto, ambos os fluxos estão em *deadlock*.

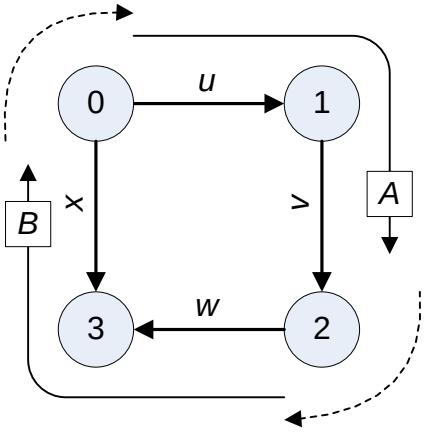


Figura 5.2 – Exemplo de ocorrência de *deadlock* [DAL04].

A premissa fundamental do *turn model* é a proibição do menor número de *turns*, de modo que a dependência cíclica não aconteça. Um *turn* define uma mudança de direção quando da execução de um algoritmo de roteamento. Considerando que um pacote pode seguir por uma de quatro possíveis direções (leste, oeste, norte e sul), há oito possíveis mudanças de direção e dois ciclos abstratos, como mostrado na Figura 5.3. De acordo com o exposto em [GLA94], se pelo menos duas mudanças de direção forem proibidas, torna-se viável a implementação de algoritmos livres de *deadlock*.

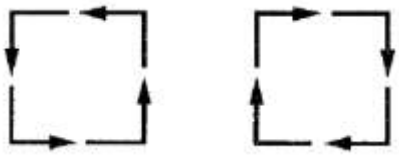


Figura 5.3 – Ciclos abstratos em uma malha bi-dimensional [DUA03].

O algoritmo XY é determinístico, sendo os *flits* de um pacote primeiramente roteados na direção X até que cheguem à coordenada X do destino, sendo em seguida roteados na direção Y até a coordenada Y do destino, como ilustra a Figura 5.4. A Figura

mostra que quando um pacote está na direção Y, nenhum *turn* pode ser executado (linhas pontilhadas). Observa-se a alta restrição imposta pelo algoritmo XY, uma vez que apenas quatro *turns*, são permitidos.

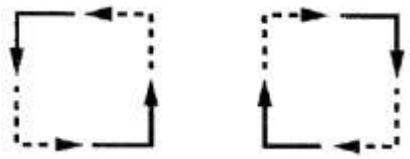


Figura 5.4 – Os quatro *turns* permitidos para o roteamento XY (setas sólidas) [DUA03].

O algoritmo WF (*West First*) define que pacotes com destinos à esquerda de uma fonte devem ser roteados de forma determinística, enquanto que pacotes com destinos à direita devem ser roteados de forma adaptativa, ou seja, observando o estado do roteador vizinho. As mudanças de direção proibidas são as duas para oeste (Figura 5.5).

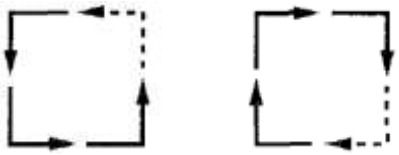


Figura 5.5 – Os seis *turn*s permitidos para o algoritmo *west-first* (setas sólidas) [DUA03].

No algoritmo NL (*North-Last*), se o IP destino de um pacote estiver acima da fonte, os pacotes são roteados deterministicamente. Se o destino estiver abaixo, os pacotes podem ser roteados de forma adaptativa nas direções oeste, leste ou sul, e em último caso para norte. As mudanças de direção proibidas são as duas possíveis quando o pacote está na direção norte (Figura 5.6).

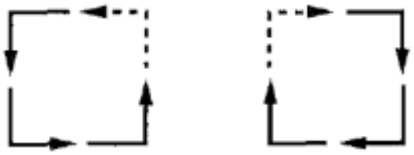


Figura 5.6 – Os seis *turn*s permitidos para o algoritmo *north-last* (setas sólidas) [DUA03].

No algoritmo NF (*Negative-First*), os pacotes são roteados primeiro nas direções negativas, isto é, para as direções sul ou oeste. Se o destino do tráfego estiver à esquerda e acima da origem, ou se estiver à direita e abaixo, os pacotes são roteados deterministicamente. Demais condições permitem que o roteamento seja adaptativo. As mudanças de direção proibidas são as duas de uma direção positiva para uma direção negativa (Figura 5.7).

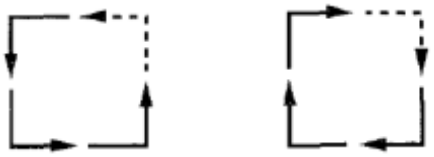


Figura 5.7 – Os seis *turn*s permitidos para o algoritmo *negative-first* (setas sólidas).

Outros algoritmos parcialmente adaptativos utilizados na construção de supercomputadores, e que são utilizados em propostas de roteadores em NoCs são o *Odd-Even* e o *Street-Sign*. Para entendimento do algoritmo Odd-Even [CHI00] é necessário considerar que em uma rede malha 2D uma coluna é dita par se a primeira coordenada de uma dimensão é representada por um número par. Por exemplo, em uma malha *m x n*, a coluna onde todos os nodos possuem endereço (2, j) *para* $0 \leq j \leq n-1$, é uma coluna par (Figura 5.8). Considere-se também que um *turn* é rotulado com as letras que representam as direções envolvidas. Por exemplo, um *turn* SN define uma mudança de direção do sul para o norte. Direções são rotuladas como E (leste), W (oeste), N (norte) e S (sul). O algoritmo *Odd-Even* especifica que são proibidas mudanças de direção EN e ES em colunas pares, e NW e SW em colunas ímpares. De acordo com [HU04], que adotou este algoritmo no roteador DyAD, o grau de adaptatividade oferecido pelo algoritmo Odd-Even distribui o tráfego de forma mais homogênea na rede, em comparação com os algoritmos WF, NL e NF.

O algoritmo de roteamento *street sign* foi originalmente proposto para a arquitetura *iWarp* [BOR88], sendo utilizado na NoC *Xpipes* [BER04]. Neste algoritmo emprega-se roteamento na origem, sendo realizado acesso a uma tabela que indica o endereço destino dos dados. Tal informação consiste de bits de direção que são lidos em cada roteador intermediário do caminho indicando a porta de saída na qual os *flits* pertencentes a um dado pacote são encaminhados.

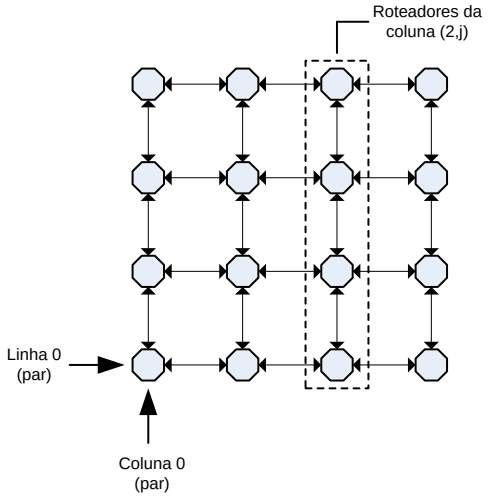


Figura 5.8 – Convenção de linhas e colunas utilizadas no algoritmo *Odd-Even*.

5.3 ROTEAMENTO *OBLIVIOUS*

No roteamento denominado *oblivious*, os pacotes são roteados sem haver o conhecimento do estado da rede [DAL04]. Esta característica o torna simples de implementar e analisar, uma vez que o aproveitamento de informações sobre a rede potencialmente traz maior complexidade ao roteamento. O roteamento na origem é utilizado na execução do roteamento *oblivious*, pela sua velocidade, simplicidade (uma vez que os nodos intermediários são computados na origem) e escalabilidade. De acordo com a classificação ilustrada na Figura 5.1, o roteamento *oblivious* é multi-fase, por ser um esquema híbrido na definição de caminhos.

O principal compromisso do roteamento *oblivious* é entre a localidade do tráfego e o balanceamento de carga. Embora o roteamento determinístico seja algumas vezes considerado idêntico ao *oblivious* [DUA03], ambos são distintos, pelo fato da escolha realizada pelo roteamento *oblivious* não ser necessariamente pré-determinada. Considerando, por exemplo, uma tabela de roteamento, a qual pode incluir diversas opções para uma porta de saída com base no endereço do IP destino. Uma destas opções pode ser selecionada aleatoriamente, ou de acordo com qualquer outro critério independente do estado da rede, contrariamente com o que acontece no roteamento determinístico, onde a mesmo caminho é sempre escolhido para um mesmo destino. Escolhe-se um algoritmo de roteamento inicial livre de *deadlock*.

Um exemplo de algoritmo que executa o roteamento *oblivious* é o algoritmo de *Valiant*. O algoritmo de roteamento aleatório de *Valiant* balanceia a carga para qualquer padrão de tráfego, em quase todas as topologias [DAL04]. Neste algoritmo, um pacote enviado de uma origem *s* para um destino *d* é primeiro enviado de *s* para um nodo intermediário *x*, escolhido aleatoriamente, para finalmente *x* enviar o pacote para *d*. Qualquer algoritmo de roteamento pode ser utilizado em qualquer uma das fases, mas em geral utilizam-se algoritmos de roteamento que balanceiam a carga para um padrão de tráfego uniforme. Para redes malha, o roteamento XY é apropriado [DAL04].

A Figura 5.9 mostra um exemplo de utilização do algoritmo de *Valiant* para encaminhar um pacote de um nodo *s*=00 para um nodo *d*=31 em uma rede malha 5x5 [DAL04]. O pacote é roteado via um roteador escolhido aleatoriamente (*x*=13). Durante a primeira fase, o pacote é roteado segundo o algoritmo XY do roteador *s*=00 para o roteador *x*=13, passando por 4 hops. Na segunda fase, o pacote é roteado de *x*=13 para *d*=31 passando por 4 hops. Desta forma, a solução aleatória passou por 8 hops, em um cenário onde um roteamento mínimo levaria 4 hops.

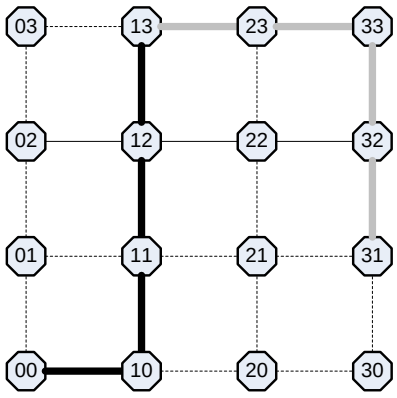


Figura 5.9 – Exemplo de caminho percorrido de acordo com o algoritmo de *Valiant*.

O roteamento *oblivious* mínimo objetiva atingir o balanço de carga do algoritmo de *Valiant*, sem abrir mão da localidade. Para isso, as rotas são restritas a serem mínimas. Sua implementação em uma malha restringe o nodo intermediário *x* a pertencer a um quadrante mínimo entre a fonte *s* e o destino *d*. O quadrante mínimo é a menor sub-rede bi-dimensional que contém *s* e *d* como nodos de borda.

A Figura 5.10 exemplifica o roteamento *oblivious* mínimo de *s*=00 a *d*=21. O primeiro passo é computar o endereço relativo Δ. A dimensão de Δ expressa o tamanho do quadrante mínimo (no exemplo, 2 hops na dimensão *x* e 1 hop na dimensão *y*). Uma vez que o tamanho do quadrante mínimo esteja calculado, um nodo intermediário x (pertencente ao quadrante) é selecionado. O pacote a ser enviado é então roteado de *s* para *x*, para finalmente ser roteado de *x* para *d*. Há, portanto, 6 possibilidades de nodos para *x*, sendo ilustradas na Figura 5.10 (b). A porção de cada rota antes do nodo intermediário é ilustrada com linhas sólidas em negrito, e a porção depois do nodo intermediário é ilustrada com linhas sólidas em cinza. É importante notar que os nodos origem e o destino podem ser selecionados como nodos intermediários.

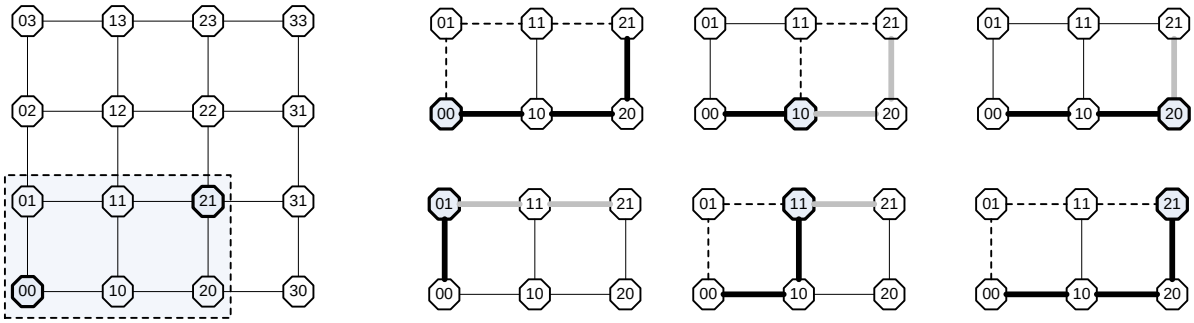


Figura 5.10 – Roteamento *oblivious* mínimo: (a)quadrado envolvente; (b) exemplos de rotas, ilustrando-se em destaque os roteadores intermediários.

O possibilidade de 6 nodos intermediários faz com que hajam três rotas possíveis, correspondentes aos três pontos nos quais o hop na direção *y* pode ser escolhido. A carga neste caso não é distribuída de maneira uniforme. No exemplo, a rota que muda para a direção *y* no nodo 20 aparece quatro vezes, enquanto que a rota que muda para y em 00 aparece somente uma vez. Este desbalanceamento pode ser minimizado, se for aleatorizada a ordem nos quais as mudanças de eixo são realizadas.

5.4 PROPOSTAS DE ALGORITMOS DE ROTEAMENTO ADAPTATIVOS PARA NOCS

Esta Seção apresenta propostas para algoritmos de roteamento utilizadas em NoCs. Cada trabalho será analisado individualmente, ressaltando o modo como é tratada a adaptatividade, além de aspectos arquiteturais dos roteadores.

Em [SAB07] é apresentado um algoritmo de roteamento dinâmico onde é utilizada a combinação dos algoritmos de roteamento *north-last* e *south-last*. Como exposto na Seção 5.2, no algoritmo *north-last* roteia um pacote deterministicamente, se o seu destino estiver acima e adaptativamente para oeste, sul ou leste (e em último caso para norte) se estiver abaixo. O algoritmo *south-last* é análogo ao *north-last*, onde é tomada a direção sul em último caso, quando a origem estiver acima, e ser determinístico quando o destino estiver abaixo. A Figura 5.11 mostra a arquitetura do roteador que executa o algoritmo proposto.

O algoritmo proposto toma como referência informações de congestionamento em cada roteador vizinho, coletadas por um controlador e decide a rota do pacote. A

adaptação do algoritmo ocorre através da verificação do estado de congestionamento dos vizinhos na horizontal. Caso eles estejam ocupados são verificados os estados dos roteadores norte ou sul, dependendo da localização roteador atual.

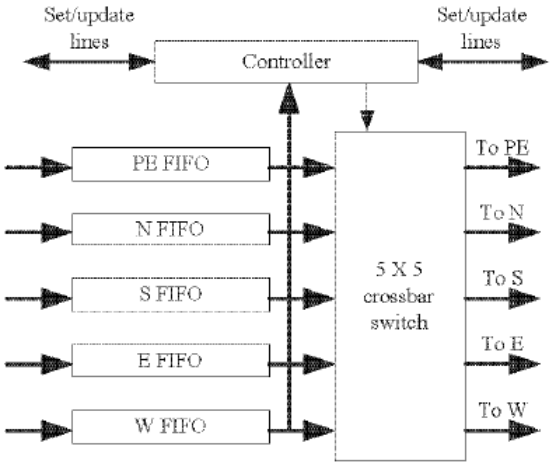


Figura 5.11 – Arquitetura do roteador proposto por [SAB07].

[ASC08] propõe uma estratégia de seleção de portas de saída cujas decisões baseiam-se na combinação do processamento de um algoritmo de roteamento com a ocupação de *buffers* de roteadores ao longo de caminhos candidatos. Esta estratégia utiliza o conceito de *neighbors-on-path*, que explora situações de indecisão que ocorrem quando a função de roteamento retorna vários canais de saída admissíveis. A visão de congestionamento desta proposta é estendida até dois hops, sendo que as portas de saída disponíveis são avaliadas com base em valores computados para cada porta de saída de cada vizinho. A Figura 5.12 mostra os sinais considerados para a execução do algoritmo.

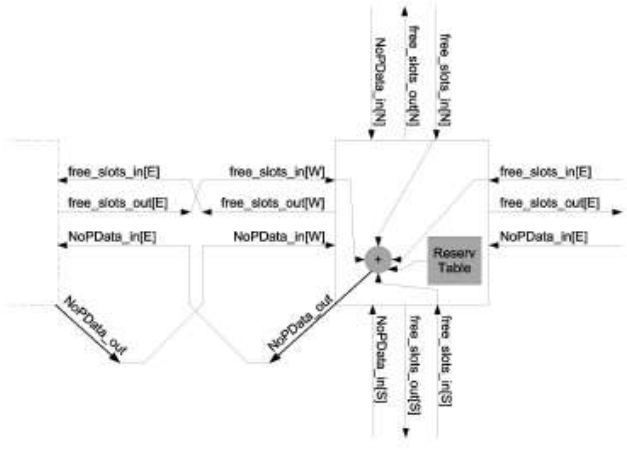


Figura 5.12 – Sinais do roteador proposto em [ASC08].

Considerando que uma direção *d* possa assumir os valores {*North*, *South*, *East*, *West*}. As variáveis *free_slots_in[d]* e *free_slots_out[d]* indicam o número de slots em *buffers* de entrada e saída, respectivamente, disponíveis no roteador vizinho na direção *d*. A variável *NoPData_in[d]* especifica dados computados pelo nodo vizinho na direção *d*.

Uma tabela é utilizada para especificar fluxos reservados para as portas de saída (*reserv table* na Figura 5.12).

A Figura 5.13 apresenta a arquitetura do roteador proposto em [ASC08]. Este roteador possui *buffers* de entrada, que por sua vez são conectados aos seus respectivos módulos de roteamento. O nodo destino, (*dst*), especificado pelo *flit* de *header*, em conjunto com a informação de estado de *buffer* (*NopData_in[]*), ambos vindos de roteadores adjacentes, são utilizados para tomar a decisão de roteamento. A canal de saída selecionado pelo módulo de roteamento é utilizado pelo árbitro para conectar o dado de entrada à sua saída. O conteúdo do bloco denominado *Algoritmo de Roteamento* (*Routing Algorithm* – detalhado na figura) consiste de dois componentes: *routing function*, que executa a função de roteamento, a qual disponibiliza o conjunto de canais candidatos; e um segundo bloco, denominado *NoP selection function*, que executa a escolha de um dos canais candidatos. Como pode ser visto, o bloco de seleção explora o conteúdo de *NoPData_in[]* proveniente de vizinhos adjacentes.

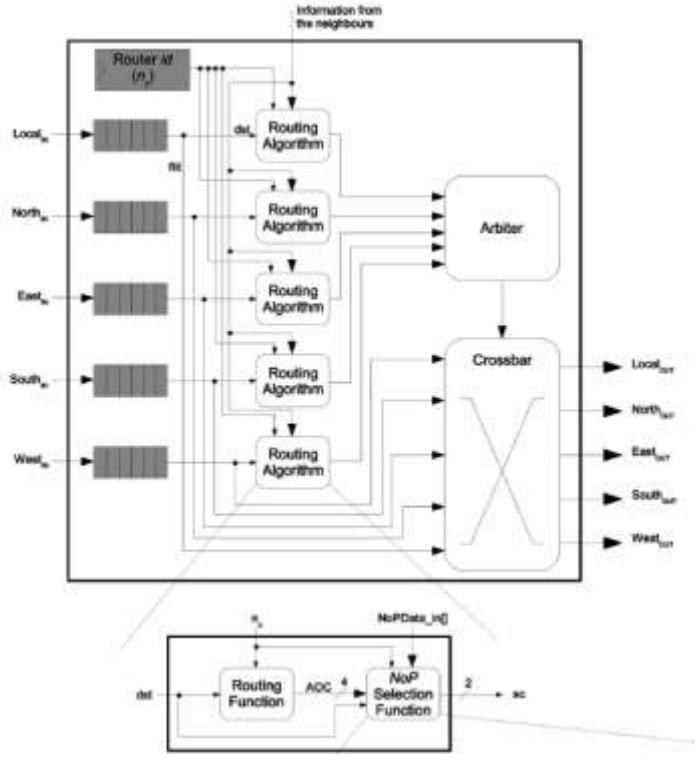


Figura 5.13 – Arquitetura do roteador proposto por [ASC08].

Em [LI06] é proposto o algoritmo *DyXY* (*Dynamic XY*) baseado no roteamento XY. Este algoritmo proporciona adaptatividade baseado nas condições de congestionamento dos roteadores vizinhos. Segundo os autores, o algoritmo é livre de *deadlock* e *livelock*, proporcionando sempre um dos caminhos mínimos possíveis entre um par origem-destino qualquer. Enquanto o pacote não estiver alinhado em um dos eixos (*x* ou *y*) com o roteador destino, o próximo roteador vizinho (em direção ao destino) é selecionado baseado no parâmetro *stress value*. O *stress value* representa a condição de congestionamento do roteador indicado pelo número de posições ocupadas em todos os *buffers* de entrada do roteador. Cada roteador armazena os *stress values* de todos os vizinhos. A Figura 5.14 ilustra a arquitetura do roteador.

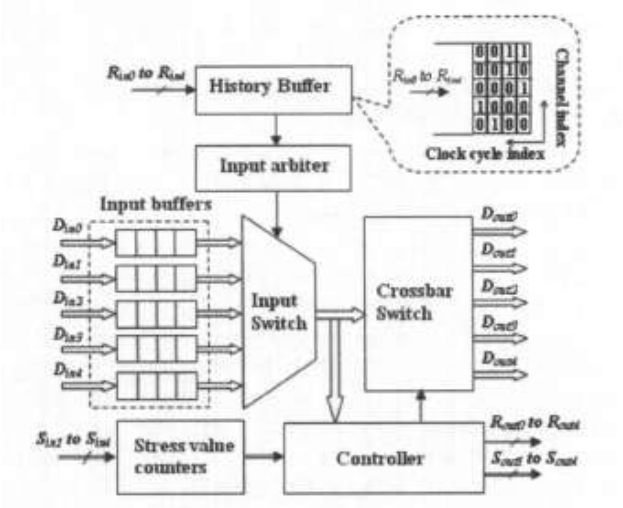


Figura 5.14 – Arquitetura do roteador proposto em [LI06].

A cada ciclo de clock, o módulo *history buffer* armazena as requisições das portas de entrada, as quais são selecionadas pelo módulo *input arbiter*. O roteamento é realizado pelo módulo *controller* baseado nos *stress values* vizinhos armazenados no módulo *stress value counters*. Ele também é responsável por distribuir o *stress value* atual do roteador para os roteadores vizinhos.

Em [HU04] é apresentado o roteamento *DyAD* (Dynamic Adaptive Deterministic), que combina as vantagens dos roteamentos determinístico e adaptativo mínimos. As decisões de roteamento levam em consideração a carga dos roteadores vizinhos. Quando a rede não está congestionada, o roteamento *DyAD* trabalha no modo determinístico, caso contrário o modo adaptativo é ativado. O algoritmo de roteamento adaptativo utilizado é o *odd-even*, apresentado na Seção 5.2. O algoritmo de roteamento determinístico utilizado é uma variação do *odd-even* chamada de *OE-fixed*. A Figura 5.15 ilustra a arquitetura do roteador *DyAD-OE*.

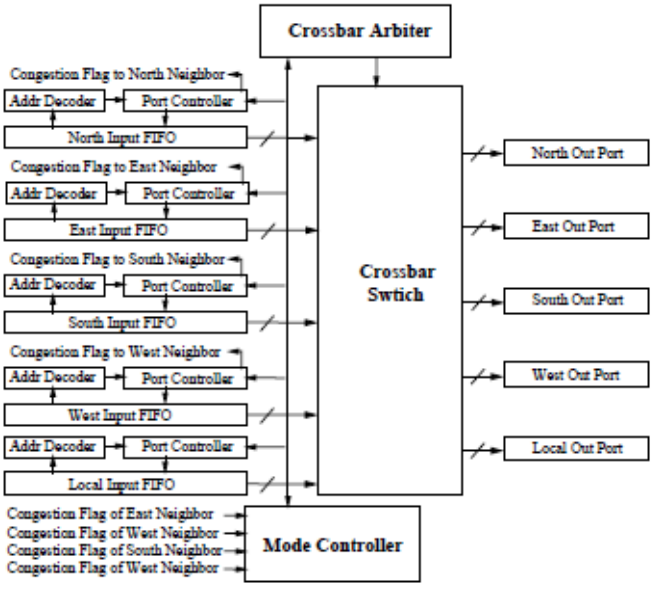


Figura 5.15 – Arquitetura do roteador DyAD-OE [HU04].

Baseado em *flags* de congestionamento (*Congestion Flag*) recebidas dos roteadores vizinhos, o módulo *Mode Controller* indica para os módulos *Port Controller* qual roteamento deve ser usado. Se alguma das flags estiver ativa, todos *Port Controllers* passam a operar no modo adaptativo, caso contrário o modo determinístico é utilizado. Dentre as possíveis portas para as quais o pacote pode ser repassado, quando o modo adaptativo é usado, é selecionada aquela que corresponde ao *buffer* vizinho com mais posições livres. Uma vez selecionada a porta de saída pelo *Port Controller*, este faz uma requisição ao *Crossbar Arbiter*, o qual utiliza uma política *first-come-first-served* para selecionar qual porta de entrada será servida. É importante também observar que os módulos *Port Controller* monitoram a ocupação dos *buffers* e enviam *flags* de congestionamento para os roteadores vizinhos.

[DAN06] propõe um algoritmo de roteamento onde a seleção da porta de saída utiliza roteamentos adaptativos mínimos e não-mínimos, com base em valores de congestionamento obtidos de roteadores vizinhos. Quando um congestionamento se forma nas vizinhanças do roteador, o roteamento não-mínimo é ativado, caso contrário o roteamento mínimo é usado. A Figura 5.12 ilustra a arquitetura do roteador.

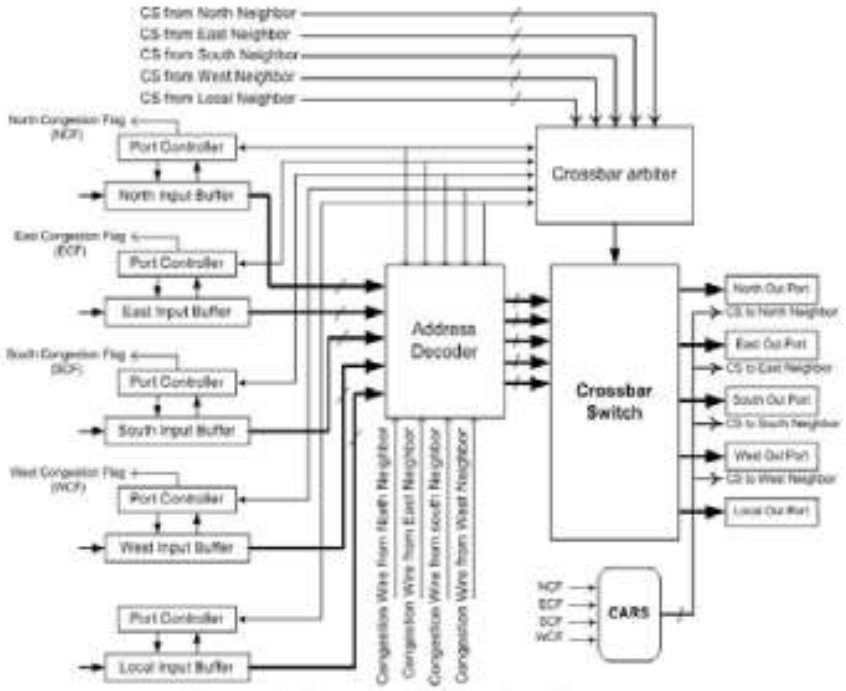


Figura 5.16 – Arquitetura do roteador proposto por [DAN06].

O módulo *Crossbar Arbiter* recebe sinais que indicam o estado de congestionamento (CS – *Congestion Status*) dos roteadores vizinhos e utiliza esta informação como prioridade para selecionar uma porta de entrada. A porta de entrada correspondente ao vizinho mais congestionado (maior prioridade) é a selecionada. O estado de congestionamento de cada roteador é calculado pelo módulo CARS (*Contention Aware Routing Selection*), que indica quantos *buffers* congestionados o roteador tem. Por exemplo, se dois *buffers* de entrada estiverem com seus *flags* de congestionamento (*Congestion Flag*) ativos, o estado de congestionamento será dois.

Em [LOF08] é proposto o algoritmo de roteamento BARP (*Balanced Adaptive Routing Protocol*). Cada roteador da NoC considera o seu próprio nível de congestionamento, onde dois níveis de *threshold* são monitorados. O parâmetro considerado é o número de pacotes no *buffer* de entrada. De acordo com o *threshold* atingido, pacotes são enviados para grupos críticos de roteadores (rotulados em *G*, na Figura 5.17), que se localizam próximos às origens de tráfego. Tais roteadores devem alterar a porta de saída de seus pacotes, de maneira a evitar regiões congestionadas.

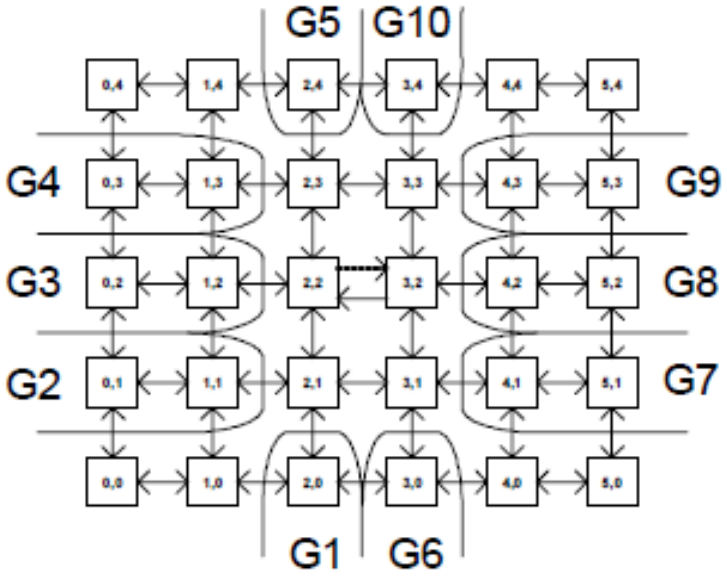


Figura 5.17 – NoC que executa o protocolo BARP, sendo os grupos de roteadores em G aqueles que vão adaptar o seu roteamento, com base em informações enviadas por outros roteadores [LOF08].

Ocorrências de *deadlock* são evitadas utilizando-se diferentes canais virtuais para cada direção de tráfego. O trabalho proposto por [LOF08] apresenta uma abordagem diferente em comparação com algoritmos que executam em roteadores como *DyAD* e *DyXY*, onde a monitoração de tráfego é realizada no mesmo que a adaptação. No protocolo BARP, o processamento de adaptação de rotas é realizado por um grupo de roteadores críticos. As notificações de congestionamento a estes roteadores geram muitas mensagens, o que aumenta a carga de tráfego na rede. Ainda, esta abordagem necessita um mecanismo para controle da chegada em ordem dos pacotes em seus respectivos destinos.

5.5 PROPOSTA DE COMPUTAÇÃO DE CAMINHOS

O algoritmo de roteamento desenvolvido pressupõe a utilização de estruturas de dados distribuídas na rede. As estruturas de dados presentes nos roteadores e na interface de rede do IP destino são responsáveis pela monitoração do tráfego e armazenamento de informações de congestionamento. A interface de rede do IP origem é responsável pela execução do algoritmo de roteamento proposto.

De acordo com a proposta deste trabalho, a definição de um novo caminho, que deverá evitar pontos congestionados, adota os seguintes pressupostos: *(i)* roteamento

mínimo, sendo que todos os caminhos têm o mesmo número de hops até o destino; *(ii)* o novo caminho deve utilizar, se possível, roteadores próximos ao caminho antigo, de modo a minimizar o impacto do novo fluxo nos outros fluxos já existentes; *(iii)* a escolha da direção *x* tem maior prioridade na computação do novo caminho; *(iv)* a pesquisa na direção *x* continua até que uma coluna *y* na posição *x* atual seja livre de congestionamento.

As estruturas de dados presentes no IP origem são a matriz *noc_cong* e o vetor *cong*. A matriz *noc_cong* armazena o histórico de congestionamento dos roteadores que fazem parte dos caminhos computados pelo algoritmo de roteamento. Considerando que roteamento mínimo adaptativo é utilizado, estes roteadores são aqueles localizados no retângulo definido pelos endereços dos pares origem-destino. A Figura 5.18 apresenta um exemplo de uma matriz *noc_cong*, onde cada posição assume um valor que pode ser ‘1’ (ponto de rede congestionado) ou ‘0’ (ponto de rede não congestionado). O vetor *cong* contém os bits recebidos no pacote ALARM, apresentado na Sessão 0. Este pacote carrega informações a respeito do congestionamento do fluxo que executa roteamento adaptativo na origem.

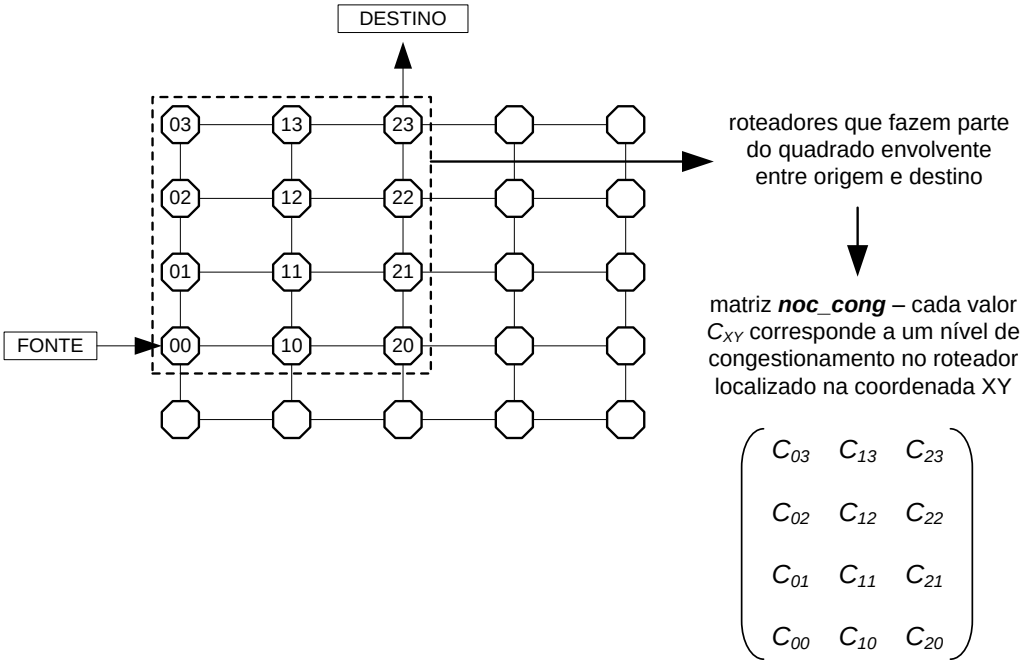


Figura 5.18 – Relação entre os endereços dos roteadores e a matriz *noc_cong*.

A recepção de um pacote do tipo ALARM com os roteadores congestionados dispara a execução do algoritmo apresentado na Figura 5.19 Quando um pacote ALARM é recebido, a tabela SCT (apresentada na Seção 4.6.1.2) é atualizada a partir do vetor *cong* (função *fill_SCT*). O exemplo na Figura 5.20 mostra um caminho com 9 hops, e congestionamento nos hops 2, 6, e 8. Após a atualização do SCT, a função *gen_noc_cong* insere novos endereços congestionados na matriz *noc_cong*.

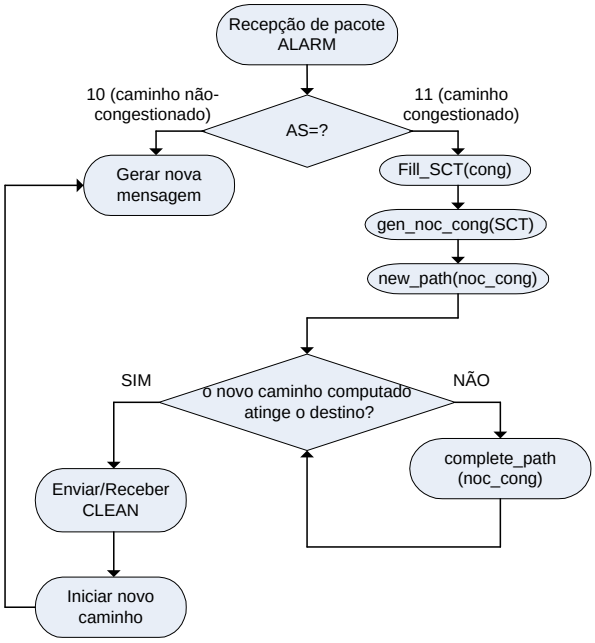


Figura 5.19 – Passos executados ao se tratar um pacote recebido na NI origem.

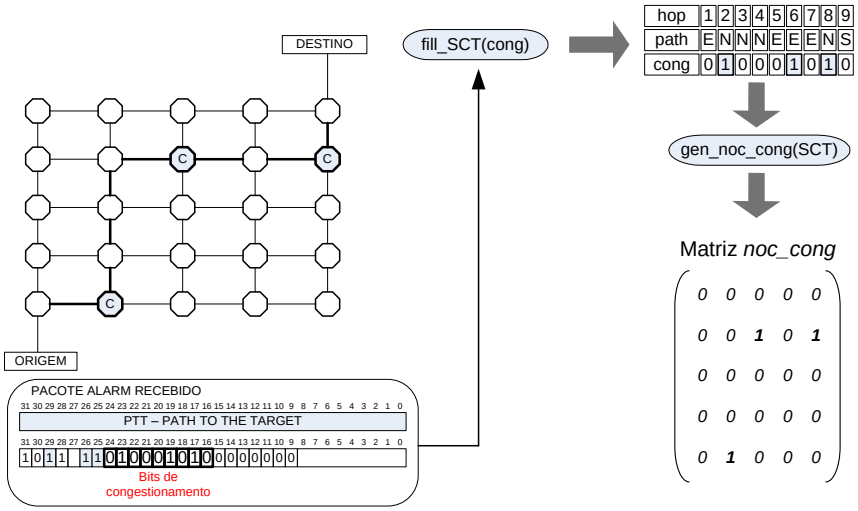


Figura 5.20 – Estruturas de dados utilizadas para computação do novo caminho.

A função *new_path* é a parte central do método de roteamento proposto, sendo detalhada no pseudo-código apresentado na Figura 5.21. O objetivo é obter um caminho parcial com linhas e colunas da rede sem roteadores congestionados. O algoritmo começa pesquisando roteadores não-congestionados na direção *x* (como acontece no algoritmo de roteamento XY). A direção *y* é tomada quando um vizinho da direção *x* está congestionado.

O laço de repetição para encontrar um caminho não-congestionado executa até que a coordenada *x* ou *y* do roteador destino seja atingida (linha 2). Na linha 3 é verificado se o próximo vizinho na direção *x* está congestionado. Em caso afirmativo, o próximo hop deve ser aquele localizado na direção *y* e uma nova iteração é executada. Isto é ilustrado na Figura 5.22(a), onde é detectado que o roteador 2 é aquele congestionado. Desta forma, a direção tomada no roteador 1 é a norte. Em caso negativo, os roteadores da direção *x* são analisados (linhas 7 a 38).

```
 1:new_path (noc_cong){
 2:  while ((x_actual!=x_target) &&
           (y_actual!=y_target)){
 3:    if (noc_cong[x_actual+1][y_actual] == true){
 4:      y_actual++;
 5:       add_in_path(new_path,y_direction);
 6:     }
 7:     else{
 8:       for (i=x_actual+1 ; i<=x_target ; i++){
 9:         if (noc_cong[i][y_actual]==true)
10:           break;
11:       }
12:       for (i_aux=x_actual+1 ; i_aux < i ; i_aux++){
13:          free_path=true;
14:          for (j=y_actual ; j<=y_target ; j++){
15:            if (noc_cong[i_aux][j]==true){
16:              free_path=false;
17:              break;
18:          }
19:          if (free_path==true){
20:            last_x_free=i_aux;
21:            x_free_found=true;
22:          }
23:       }
24:       if (x_free_found==true){
25:          for (m=x_actual ; m<last_x_free ; m++){
26:            add_in_path(new_path,x_direction);
27:            x_actual++;
28:          }
29:          for (m=y_actual ; m<y_target; m++){
30:            add_in_path(new_path,y_direction);
31:            y_actual++;
32:          }
33:       }
34:       else{
35:          y_actual++;
36:          add_in_path(new_path,y_direction);
37:       }
38:    }
39:  }
40:}
```

Figura 5.21 – Algoritmo para definição de novo caminho.

O trecho compreendido pelas linhas 8 a 11 executam a pesquisa do primeiro roteador congestionado no endereço *y* corrente. Este passo define o espaço de pesquisa para o endereço *y* atual. Na Figura 5.22(b), todos os roteadores em *y*=2 não estão congestionados. O laço entre as linhas 12 e 23 analisa as colunas para cada roteador na atual coordenada *y*. Para cada roteador em uma posição *x*, são verificados todos os roteadores que pertencem à coordenada *x* correspondente, e que podem fazer parte do novo caminho de roteamento. Se há pelo menos um roteador congestionado, a coluna relacionada à sua coordenada *x* é considerada como não sendo livre de congestionamento. O objetivo é encontrar a coluna livre de congestionamento mais próxima do roteador destino. A Figura 5.22(b) ilustra que há duas colunas que são livres de congestionamento, nas coordenadas em *x*=1 e *x*=3.

Uma vez que um elemento da coordenada *x* livre de congestionamento tenha sido encontrado, é acrescido ao caminho as direções horizontais e verticais, até que a coordenada *y* do roteador destino seja alcançada. No exemplo apresentado na Figura 5.22 (b), o caminho é completado com três roteamentos para a direção *x*. Na Figura 5.22(c) são mostradas 3 rotas na direção *y*. Isto é feito entre as linhas 24 e 33 do algoritmo apresentado na Figura 5.21.

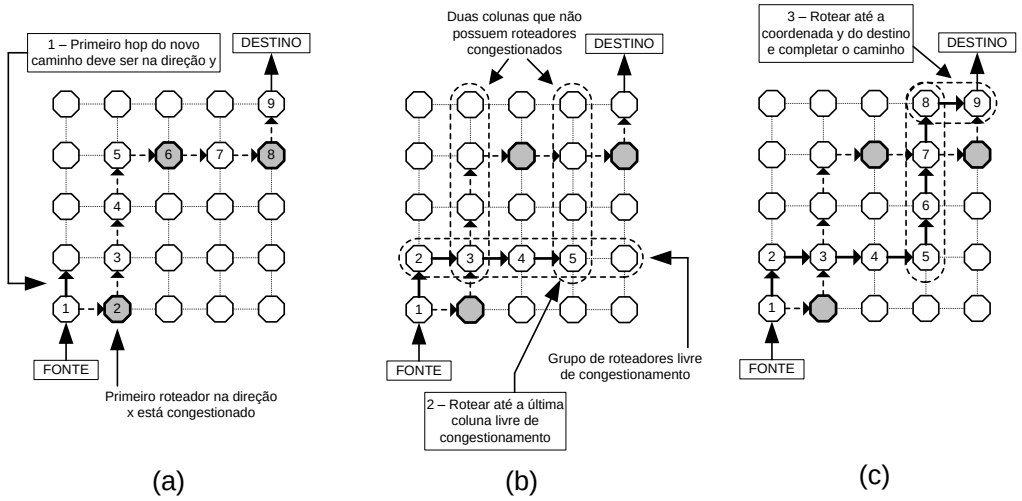


Figura 5.22 – Exemplo ilustrativo da computação de um novo caminho.

Se o caminho parcial estiver incompleto, torna-se necessário completar o caminho até o nodo destino. Isto é feito pela função *complete_path* (veja Figura 5.21). Devido ao fato que um alinhamento na direção *x* ou *y* com o roteador destino será obtido após a execução da função para encontrar o novo caminho, somente roteamento na direção horizontal ou vertical será realizado, dependendo do estado atual da variável *new_path*. A Figura 5.22(c) ilustra o caminho sendo completado até o destino, com um roteamento sendo feito na direção *x* (no exemplo, a direção *leste*).

Antes da adoção do novo caminho, um pacote especial do tipo DATA (contendo um bit para limpeza do campo *others*) é enviado ao destino utilizando o caminho antigo, limpando as entradas das tabelas de congestionamento dos roteadores intermediários. O novo caminho é então inicializado com um pacote do tipo DATA para abrir uma nova sessão de roteamento, identificando os roteadores deste caminho. Considerando que roteamento mínimo é adotado, o TCT do nodo destino permanece com o mesmo tamanho.

5.6 PREVENÇÃO DE *DEADLOCK* COM A UTILIZAÇÃO DE REDES VIRTUAIS

Em oposição ao roteamento distribuído, onde o caminho até o destino é definido *hop* a *hop*, o método proposto neste trabalho define e reconfigura o caminho de um fluxo na origem. Isto permite um controle eficiente para evitar ocorrências de *deadlock*, uma vez que os *turns* que serão executados são conhecidos *a priori* pela fonte. Ocorrências de *livelock* também são evitadas, considerando que somente roteamento mínimo é realizado.

Com o intuito de evitar ocorrências de *deadlock*, adota-se neste trabalho um conceito útil em propostas de algoritmos de roteamento, que é a divisão da rede física em *redes virtuais*. Uma rede virtual é um subconjunto de canais que são utilizados para rotear pacotes em torno de um conjunto particular de destinos. Os conjuntos de canais correspondentes a diferentes redes virtuais são separados. Dependendo do destino, cada pacote é injetado em uma rede virtual em particular, onde é roteado até o destino ser atingido. Em algumas propostas, pacotes que se movimentam em uma dada rede virtual têm alguma liberdade para utilizar uma outra rede virtual. Redes virtuais podem ser criadas utilizando-se conjuntos isolados de canais virtuais para cada uma, e mapeando

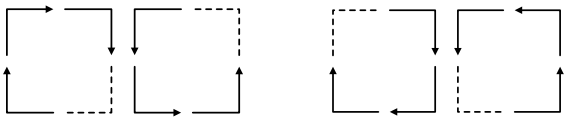
estes canais sobre o mesmo conjunto de canais físicos. É possível também a implementação das redes virtuais utilizando conjuntos separados de canais físicos.

Além do roteamento proposto neste trabalho ser na origem, ele também é *completamente adaptativo*, visto que as rotas estabelecidas podem ser modificadas independentemente da posição dos pares origem-destino. As primeiras metodologias de projeto para algoritmos de roteamento completamente adaptativos foram baseadas no conceito de redes virtuais [JES89][LIN91]. Este conceito facilita consideravelmente a tarefa de definir funções de roteamento livres de *deadlock*. Efetivamente, *deadlocks* são somente possíveis se existem dependências cíclicas entre canais que compartilham recursos da rede, como explicado no início deste Capítulo. Ao se restringir o conjunto de destinos para cada rede virtual, é possível restringir o conjunto de direções a serem seguidas por pacotes. Assim, cada rede virtual pode ser utilizada de modo que a função de roteamento correspondente não gere dependências cíclicas entre canais. Desta forma, transferências de pacotes entre redes virtuais não são permitidas, de modo que *deadlocks* sejam evitados.

As redes virtuais são implementadas neste trabalho através da multiplexação do canal físico em canais virtuais. Mais precisamente, dois canais virtuais são utilizados, sendo que um deles adota roteamento *west-first*, e o segundo canal virtual adota um algoritmo simétrico, *east-first*. A NI do roteador fonte escolhe um canal virtual de acordo com as seguintes regras:

1. *Se o destino de tráfego está à **direita** da origem, utilizar o **canal virtual 1**;*
2. *Se o destino de tráfego está à **esquerda** da origem, utilizar o **canal virtual 2**;*
3. *Se o destino de tráfego está **alinhado** na direção vertical ou horizontal com o roteador origem, utilizar o **canal virtual 1**.*

A Figura 5.23 apresenta os *turn*s permitidos, de acordo com as regras 1 (em (a)) e 2 (em (b)).



(a) West-first routing (b) East-first routing

Figura 5.23 – *Turns* permitidos no método de roteamento proposto.

5.7 CONSIDERAÇÕES SOBRE ROTEAMENTO ADAPTATIVO

A Tabela 5.1 apresenta a comparação entre as estratégias de roteamento adaptativo acima descritas. Os critérios adotados são a estratégia de roteamento, o local onde é realizada a monitoração, o que é monitorado e a política de escalonamento. A maior parte dos trabalhos relacionados realiza roteamento distribuído, sendo que em cada *hop* é analisado o estado de congestionamento para guiar a etapa de roteamento. O algoritmo proposto nesta tese apresenta um importante contraponto em relação aos

trabalhos relacionados, pelo fato de realizar a adaptação do roteamento *na origem*. A estratégia de roteamento define como é realizada a decisão para rotear um pacote. Observa-se que os algoritmos de roteamento analisam o congestionamento normalmente em *buffers* dos roteadores vizinhos. As propostas de análise de *buffers* variam desde a simples detecção de disponibilidade para receber dados (*buffer* cheio ou não cheio) até a verificação do número de posições ocupadas. Observa-se também que as estratégias de escalonamento variam. Na estratégia *round-robin* existe uma ordem pré-estabelecida para escalonar uma porta de entrada para ter seus dados roteados. Na política *first-in-first-out* as portas de entrada são atendidas na ordem em que solicitam roteamento. [DAN06] diferencia-se por atribuir prioridade de acordo com o congestionamento nas portas de saída.

Tabela 5.1 – Trabalhos relacionados a algoritmos de roteamento adaptativos em NoCs.

Autor	Adaptação	Estratégia de roteamento	Local da monitoração	Parâmetro de monitoração	Política de escalonamento
[SAB07]	Distribuída	Primeiro verifica estado de congestão de vizinhos na horizontal. Se estes não estão disponíveis, analisa na vertical.	Roteadores vizinhos	Disponibilidade dos roteadores vizinhos	Round-robin
[ASC08]	Distribuída	Define caminhos possíveis e escolhe aquele menos congestionado.	Porta de saída do roteador atual e de seu vizinho	Grau de preenchimento de *buffers*	Round-robin
[LI06]	Distribuída	Adapta a rota enquanto alinhamento horizontal ou vertical com o destino não é atingido.	Roteadores vizinhos	Grau de preenchimento de *buffers*	First-in-first-out
[HU04]	Distribuída	Adapta a rota quando da detecção de congestão.	Roteadores vizinhos	Grau de preenchimento de *buffers*	First-in-first-out
[DAN06]	Distribuída	Utiliza roteamento não-mínimo quando congestionamento é detectado.	Roteadores vizinhos	Grau de preenchimento de *buffers*	Prioridade (verifica qual roteador está mais congestionado)
[LOF08]	Distribuída	Roteadores monitoram o seu estado de congestionamento e enviam medições para roteadores críticos, que executam a adaptação.	Roteadores vizinhos	Grau de preenchimento de *buffers*	Round-robin
Proposta deste trabalho	**Na origem**	**Antes da transmissão de cada mensagem, o estado de congestionado do caminho é interpretado, sendo que o novo caminho deve evitar os roteadores congestionados**	**Roteadores pertencentes ao caminho do fluxo**	**Taxa de entrada de dados**	**Round-robin**

O fato da monitoração do estado de congestionamento apenas nos roteadores vizinhos não traz uma visão global do que acontece na rede, o que pode causar congestionamento em outros pontos mais distantes. A maior aplicabilidade da monitoração nos vizinhos ocorre em cenários onde os tráfegos possuem maior localidade. Tráfegos com menor localidade tendem a experimentar maior congestionamento, sendo necessária adaptação de acordo com o congestionamento no caminho da comunicação. A proposta de roteamento adaptativo apresentada neste trabalho observa regiões da rede que estão, ou estiveram congestionados, de modo a evitar pontos de congestionamento para os próximos fluxos de dados.

O fato do algoritmo de roteamento ser executado a cada fim de mensagem traz duas importantes vantagens. A primeira é a garantia de entrega em ordem dos pacotes, uma vez que pacotes da mesma mensagem percorrerão o mesmo caminho. A segunda vantagem é que a troca de caminhos levará em conta uma análise dos hops do caminho a cada pacote DATA recebido, o que evita a troca de caminhos motivada por pontos de congestionamento esporádicos. É necessário entretanto, não se utilizar mensagens muito longas para compor mensagens, uma vez que isso pode tornar o tempo de reação do algoritmo frente ao congestionamento proporcionalmente alto.

6. EXPERIMENTOS REALIZADOS

Este capítulo apresenta os experimentos realizados para avaliar os métodos aqui propostos para geração de tráfego, dimensionamento de *buffers* de Nis, monitoração de tráfego e roteamento adaptativo.

O método de geração de tráfego apresentado no Capítulo 2, foi publicado em [TED05] e [TED06]. A proposta de dimensionamento de *buffers* apresentada nesta tese no Capítulo 3, foi publicada em [TED08]. São aqui analisados as conseqüências causadas pelo correto dimensionamento de *buffers* de NIs de núcleos com restrições de QoS, que concorrem com tráfegos que também utilizam intensivamente a rede, e que causam jitter em fluxos QoS.

O segundo conjunto de experimentos inclui a avaliação da estrutura de monitoração e roteamento adaptativo proposta nos Capítulos 4 e 5 e publicada em [TED09] e [TED09a]. São apresentados resultados de latência, tempo de *flits* em *buffers* e reatividade a situações de congestionamento de fluxos QoS que concorrem com tráfegos com variados graus de localidade.

6.1 AVALIAÇÃO DO DIMENSIONAMENTO DE *D-BUFFERS*

O objetivo dos experimentos que são aqui apresentados é a verificação do comportamento de um cenário cujo tráfego possui requisitos de QoS. Tal fluxo compartilha recursos da rede com flutuação de carga, causando jitter nos dados recepcionados pelos IPs destinos.

A NoC Hermes [MOR04] é utilizada para executar os cenários de tráfego propostos e validar o método para dimensionamento de *buffers* apresentado. Os parâmetros de projeto da rede são: topologia malha 8x8, tamanho de *flits* de 16 bits; *buffers* internos com tamanho de 8 *flits*, controle de fluxo baseado em créditos, roteamento XY determinístico e 2 a 4 canais virtuais associados a cada canal físico. A freqüência empregada é de 50 MHz, o que corresponde a uma taxa de transmissão por canal de 800 Mbps.

Os experimentos empregam três modelos de tráfego sintéticos: sinais de controle, HTTP e HDTV. A Tabela 6.1 resume as características dos modelos de tráfego. Sinais de controle são utilizados como tráfego de ruído, sendo transmitidos utilizando uma distribuição espacial de tráfego complemento. Tráfego HTTP é utilizado para perturbar o *stream* de tempo real HDTV. O tráfego HDTV é caracterizado com requisitos temporais, sendo que *D-buffers* devem ser conectados aos Ips recebendo este tráfego, com o objetivo de garantir vazão sem violação de prazos, isto é, perda de dados por *starvation* ou *buffer* cheio.

A transmissão de sinais de controle é modelada com pacotes de 15 *flits*, os quais representam um tamanho típico para este tipo de aplicação. Estes pacotes são gerados continuamente no tempo, caracterizando um fluxo CBR. A taxa de injeção é de 16 Mbps, o que corresponde a 2% do consumo de largura de banda. O tráfego HTTP é modelado

utilizando a processo de injeção Pareto ON-OFF, sendo os pacotes gerados durante o período ON (atividade) a 160 Mbps. O modelo ON-OFF descreve o tráfego HDTV. Durante o período ON são gerados 10 pacotes a 200 Mbps, o que corresponde a 10 canais HDTV, consumindo assim 25% da largura de banda da NoC. Cada iniciador HDTV transmite 20 frames. Os cenários simulados empregam pacotes de tamanho fixo e tamanho baseado em traces obtidos de [FIT08].

Tabela 6.1 – Modelagem de tráfego para os experimentos conduzidos
(TP: Tamanho de Pacote; TI: Taxa de Injeção).

Aplicação	**Requisito**	**Características de tráfego**	
Sinais de controle	Sinalização	TP	15 *flits*
		TI	Taxa de injeção constante: 16 Mbps
100 sessões HTTP	Transferência de blocos em tempo real	TP	600 a 1500 *flits* (tamanhos fixos por experimento)
		TI	Pareto ON-OFF: 160 Mbps durante o período ON
10 canais HDTV	Stream de tempo real	TP	Fixo (1500 *flits*) e baseado em *traces* (1500 *flits* em média)
		TI	Transmissão de 10 fluxos HDTV simultâneos por *frame*. Taxa durante o período do frame: 200 Mbps (carga de 25%).

A Figura 6.1 ilustra a distribuição de tráfego espacial utilizada nos experimentos, com dois iniciadores de HDTV (M1-M2), onde o fluxo HDTV oriundo de M1 é perturbado por 3 fluxos HTTP. A distribuição complemento empregada pelo tráfego de ruído aumenta o volume de comunicação na biseção da rede. Os fluxos HDTV e HTTP são posicionados próximo ao centro da rede, de modo a maximizar a influência do tráfego de ruído nestes fluxos. O número de fluxos em competição deve ser superior ao número de canais virtuais, de modo que se possa avaliar o impacto da concorrência na latência e no *jitter* para o fluxo HDTV considerado.

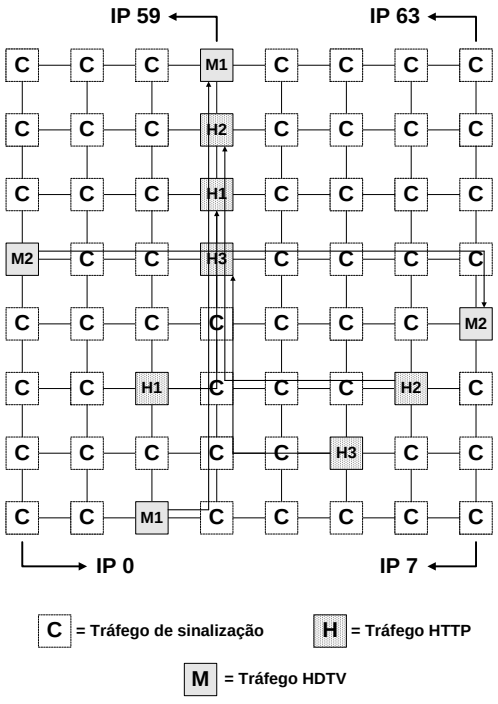


Figura 6.1 – Distribuição de tráfego espacial, sendo o fluxo HDTV M1 perturbado por 3 fluxos HTTP.

6.1.1 DIMENSIONAMENTO DE *BUFFERS* COM PACOTES DE TAMANHO FIXO E VARIÁVEL

Este cenário modela fluxos HDTV utilizando o modelo ON-OFF, para pacotes de tamanho fixo e variável. No cenário com pacotes de tamanho fixo, os mesmos possuem tamanho de 1500 *flits*. *Traces* coletados em [FIT08] são utilizados para definir o cenário de pacotes de tamanho variável. Pacotes de *traces* possuem um tamanho médio de 1500 *flits*, variando entre 15 a 6200 *flits*. A Tabela 6.2 apresenta resultados relativos aos parâmetros *threshold* e tamanho para o *D-buffer*, assim como a latência, considerando a ocorrência ou não de empacotamento, concorrência entre fluxos e número de canais virtuais. O IP destino considerado é o de número 59.

Tabela 6.2 – Valores de *threshold* e tamanho de *D-buffer* para o IP 59.

Linha	Modelo de tráfego	Empacotamento?	Concorrência?	Dois canais virtuais			Quatro canais virtuais		
				Bthreshold (ciclos)	Bsize (*flits*)	Latência (ciclos)	Bthreshold (ciclos)	Bsize (*flits*)	Latência (ciclos)
1	Sintético (pacotes de tamanho fixo)	Não	Não	0	9	22547	0	9	22550
2		Não	Sim	0	9	24794	0	10	22552
3		Sim	Não	0	1125	2300	0	1125	2300
4		Sim	Sim	2664	1817	5059	0	1150	2407
5	Traces (pacotes de tamanho variável)	Não	Não	0	9	21501	0	9	21502
6		Não	Sim	20	704	24112	4	11	21301
7		Sim	Não	16048	4650	14193	16048	4650	14914
8		Sim	Sim	14520	4654	14192	16048	4650	14915

6.1.1.1 IMPACTO DO EMPACOTAMENTO

Quando o processo de empacotamento de dados não é adotado, não há uma necessidade significativa de *slots* de *buffers*, porque praticamente não há diferença entre a taxa original de dados transmitida e a que chega ao IP destino. A adoção de empacotamento reduz significativamente os valores de latência e da contenção na rede, ao custo de aumento no tamanho no *D-buffer* (*Bsize*). O tamanho do *D-buffer* é proporcional à taxa de injeção e ao tamanho máximo de pacote. O experimento adota uma taxa de injeção de 25% da largura de banda do canal, significando que um *flit* deve ser consumido a cada 4 ciclos. Portanto, para cada *flit* consumido, 3 *flits* permanecem armazenados no *D-buffer*.

Multiplicando-se os tamanhos máximos de pacotes (1500 para tráfego sintético e 6200 para tráfego baseado em traces) por 0.75 (percentual de ociosidade do tráfego gerado), resulta em um *D-buffer* de tamanho 1125 e 4650 para tráfego sintético e baseado em traces, respectivamente. Este valores podem ser observados nas linhas 3 e 7 da Tabela 6.2. Para pacotes de tamanho fixo ou quando não há concorrência entre fluxos, o valor de *threshold* é zero. Isto significa que *flits* podem ser consumidos assim que chegam ao *D-buffer* do IP destino.

Quando pacotes de tamanho variável são utilizados em conjunto com o processo de empacotamento, os valores de *threshold* são maiores que zero (linha 7-8). A Figura 6.2 mostra a chegada de pacotes no *D-buffer* e sendo consumidos pela aplicação,

considerando um valor de *threshold* igual a zero. O intervalo entre pacotes é proporcional ao tamanho do pacote. Esta lacuna corresponde à condição de *starvation*, que ocorre quando o IP não possui dados a consumir. O valor de *threshold* resolve este problema, uma vez que ele define a quantidade de tempo a esperar antes do início do consumo de dados. Este valor é alto porque uma alta variabilidade de tamanhos de pacotes é apresentada pelos *traces* utilizados.

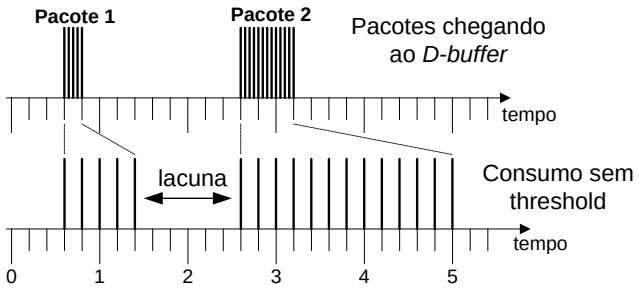


Figura 6.2 – Consumo de dados sem *threshold*, causando *starvation*.

6.1.1.2 IMPACTO DA CONCORRÊNCIA

Para o modelo de tráfego sintético, o tráfego concorrente leva a um considerável aumento nos valores de *threshold* e tamanho do *D-buffer* (comparar linhas 3-4 da Tabela 6.2). Aumentando o número de canais virtuais, o efeito da concorrência é minimizado. Para tráfego baseado em *traces*, o impacto da concorrência entre fluxos é maior quando o processo de empacotamento é utilizado (linhas 5-6).

A concorrência entre fluxos não afeta o dimensionamento do *D-buffer*, considerando que o tamanho do *buffer* é suficiente para armazenar dados sem esperar pelo período de threshold (linhas 7-8). O aumento no número de canais virtuais não influencia no dimensionamento do *D-buffer*, pela mesma razão: o *buffer* é dimensionado de acordo com o maior pacote. A concorrência afeta principalmente o valor de *threshold* para ambos os modelos de tráfego. O tamanho do *D-buffer* é bastante afetado pelo tamanho do maior pacote. A latência é minimizada com o emprego do empacotamento com tamanhos de pacote fixos, como ocorre por exemplo em redes ATM.

6.1.1.3 IMPACTO DO ROTEAMENTO/ARBITRAGEM

O impacto do roteamento e arbitragem é muito pequeno, se comparado com as outras fontes de jitter. Este efeito é causado pelo maior tamanho dos pacotes HDTV (1500 a 6200 *flits*), comparado à quantidade de ciclos de relógio necessários para roteamento/arbitragem (em torno de 7).

Este impacto pode ser notado nas linhas 1-2 e 5-6, quando o processo de empacotamento não é adotado. O tamanho de *D-buffer* obtido corresponde ao número de ciclos para processar um pacote em um roteador. Nota-se na linha 6 a redução do tamanho do *D-buffer*, quando 4 canais virtuais são utilizados.

6.1.2 OCUPAÇÃO DE *BUFFER*

A Figura 6.3 mostra o número de *slots* de *buffer* preenchidos (eixo *x*) *versus* o número de ciclos de relógio que estes slots são preenchidos durante a execução de uma simulação (eixo *y*), quando pacotes de tamanho fixo para tráfego HDTV e dois canais virtuais são utilizados. Na ausência de concorrência, o *buffer* é utilizado de maneira homogênea no tempo. Este comportamento muda com tráfego concorrente. Quando um dado pacote é bloqueado, ele possivelmente pode ser enviado com um intervalo até o próximo pacote reduzido, requerendo desta forma espaço em *buffer* para evitar perdas de dados (devido a violações de prazos). Comparando os tamanhos de *D-buffer* (Tabela 6.2-linhas 3 e 4 da coluna 6), a diferença é 692, correspondendo ao deslocamento à direita na curva. Estes 692 slots em *buffer* extras são requeridos para absorver o *jitter* introduzido pela concorrência entre fluxos.

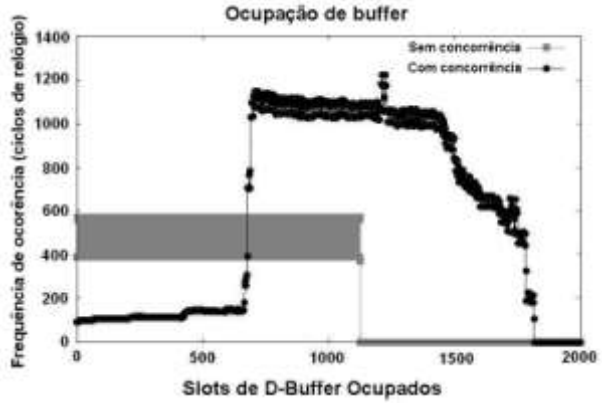


Figura 6.3 – Ocupação de *D-buffer* sendo o fluxo HDTV composto por pacotes de tamanho fixo de 1500 *flits*, sem (cinza) e com (preto) concorrência de pacotes HTTP de 1500 *flits*.

A Figura 6.4 apresenta a utilização de *buffer* para um fluxo que possui tamanhos de pacotes variáveis. Como mostrado na Figura 6.3, fluxos com tamanhos de pacote fixos utilizam o *buffer* de uma maneira uniforme. A Figura 6.4 mostra um comportamento distinto, considerando que a ocupação em *buffer* é de em média 70% durante simulação da aplicação. Isto é induzido por pacotes maiores, os quais devem ser armazenados com o objetivo de serem consumidos na taxa da aplicação.

Figura 6.4 – Ocupação do *D-buffer* sendo o tráfego HDTV modelado com tamanhos de pacote variáveis, sem (cinza) e com (preto) concorrência de pacotes HTTP de 1500 *flits*.

6.1.3 VIOLAÇÃO DE PRAZOS

Esta seção avalia o percentual de violações de prazos devido à redução do tamanho do *D-buffer*. Violações de prazos estão relacionadas à *flits* que não foram consumidos devido a problemas de *starvation* ou *buffer* cheio. A Figura 6.5 apresenta a latência e o percentual de violações de prazos para pacotes de tamanho fixo, considerando o processo de empacotamento e 3 fluxos HTTP perturbando o fluxo QoS, em função da dimensão do *D-buffer*. Se não há a utilização do *D-buffer*, é observado um percentual de prazos violados em 68%, devido ao *jitter*. A redução de 40% do tamanho do *buffer* e do *threshold* acarreta uma importante redução na latência, para um pequeno número de prazos violados (5%). Este gráfico mostra que: (*i*) se um pequeno percentual de violações de prazos é permitido, o *buffer* pode ser reduzido, minimizando área e latência; (*ii*) as violações de prazos aumentam consideravelmente com a redução do tamanho do *D-buffer*.

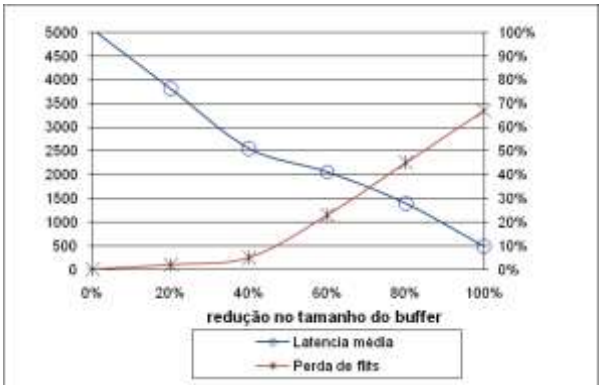


Figura 6.5 – Latência e perda de *flits*, para tráfego HDTV modelado com pacotes de tamanho fixo, perturbado por 3 fluxos HTTP (pacotes de 1500 *flits*).

A Figura 6.6 apresenta um experimento similar em relação ao da Figura 6.5, utilizando pacotes com tamanho variável. Apesar de apresentar maiores valores de latência, a diminuição da mesma possui o mesmo comportamento em ambos os experimentos. Comparando a Figura 6.6 com a Figura 6.5, a principal diferença observada é o crescimento do número de violações de prazos. Quando o tamanho do *D-buffer* é reduzido em 60%, o tráfego de pacotes com tamanho fixo possui 22% de violações de prazos, enquanto que em relação a pacotes de tamanho variável, ocorrem 10% de violações de prazos.

Figura 6.6 – Latência e perda de *flits* para tráfego HDTV baseado em traces, perturbado por 3 fluxos HTTP (pacotes de 1500 *flits*).

6.1.4 CONCLUSÕES SOBRE OS EXPERIMENTOS COM *D-BUFFERS*

O primeiro conjunto de experimentos, apresentado na Seção 6.1 demonstrou o efeito da concorrência entre fluxos no *jitter*, tamanho de *buffer* e ocupação em *buffer*, mesmo quando canais virtuais são utilizados. As curvas apresentadas em Figura 6.3 e Figura 6.4 foram obtidas simulando a rede após o dimensionamento do *D-buffer*. O número de pacotes perdidos foi zero, com uma taxa de consumo constante, validando o método de dimensionamento de *buffers* apresentado no Capítulo 3.

A segunda parte de experimentos, apresentada na Seção 6.1.3 demonstrou a influência da redução do *D-buffer* nas latências médias dos pacotes dos fluxos HDTV. Observou-se que a adoção de pacotes fixos reduz a latência de transmissão de dados e a necessidade de *D-buffers* maiores, apresentando, no entanto, maior sensibilidade à redução de *D-buffer*, apresentando maiores violações de prazos em comparação com a utilização de *traces*.

No contexto desta tese, o dimensionamento e a utilização de *D-buffers* constituem fatores importantes para a execução correta de aplicações com requisitos de QoS. Através da detecção de tendência de violações de prazos pode-se notificar a rede de modo a corrigir o encaminhamento do fluxo que está sob efeito de *jitter*. O desenvolvimento de mecanismos para tratar congestionamento em NoCs traz como possibilidade a diminuição da necessidade de *buffers* maiores nas Interfaces de Rede. Como visto nos experimentos com *traces*, a redução no tamanho do *D-buffer* em 60% produziu 10% de violações de prazos, o que pode ser diminuído com a adoção de tratamento de fluxos em tempo de execução.

6.2 AVALIAÇÃO DO MÉTODO DE MONITORAÇÃO E ADAPTAÇÃO PROPOSTO

O conjunto de experimentos aqui apresentado utiliza a NoC assíncrona que faz parte da plataforma MAGALI [CLE09]. Os métodos foram implementados durante o período de estágio sanduíche, utilizando a linguagem SystemC TLM. Entre as principais características desta arquitetura de comunicação estão: (*i*) roteamento na origem; (*ii*) método de chaveamento de pacotes wormhole; (*iii*) tamanho de *flit* igual a 32 bits; (*iv*) controle de fluxo baseado em créditos; (*v*) freqüência de operação de 500Mhz.

Nos experimentos conduzidos, cada roteador monitora a quantidade de tempo durante a qual os *flits* esperam nos *buffers* de entrada antes de serem encaminhados. Denomina-se este tempo de *tempo de flit*. A latência de rede de um *flit* corresponde à soma de todos os *tempos de flit* em seu caminho. No lado do receptor de tráfego QoS, ocorre a monitoração periódica de seu TCT para detectar congestionamento no caminho (como mostrado na Figura 4.19). Um primeiro experimento, sem tráfego de perturbação, foi executado e um tempo de *flit* de 1 ciclo de relógio foi obtido em cada *hop*. Para os cenários com tráfego de perturbação, um roteador com *flits* esperando mais que dois ciclos para serem encaminhados é considerado congestionado. Portanto, um valor maior que 2 ciclos de relógio é o parâmetro de *threshold* utilizado para sinalizar um roteador congestionado.

6.2.1 CENÁRIO 1

Neste primeiro cenário, três tipos de geradores e receptores de tráfego são definidos, como mostrado na Tabela 6.3 (para geradores) e Tabela 6.4 (para receptores). Fluxos de tráfego QoS são aqueles que possuem requisitos temporais, sendo seus receptores aqueles que executam o método de roteamento dinâmico apresentado no Capítulo 4. Os tráfegos de perturbação são aqueles que produzem áreas de *hot-spot*, causando variações nos fluxos QoS. Um tráfego ruído é o terceiro tipo de tráfego gerado, não produzindo consideráveis perturbações em outros fluxos que trafegam na rede.

Tabela 6.3 – Tipos de fluxos gerados.

Rótulo	Descrição do nodo
$\mathbf{Q_{Gfn}}$	Gerador de tráfego QoS do fluxo de número *fn*.
$\mathbf{D_{Gfn}}$	Gerador de tráfego de perturbação do fluxo número *fn*.
$\mathbf{N_{Gfn}}$	Gerador de tráfego de ruído do fluxo número *fn*.

Tabela 6.4 – Tipos de fluxos recebidos.

Label	Descrição do nodo
$\mathbf{Q_{Rfn}}$	Receptor de tráfego QoS do fluxo de número *fn*.
$\mathbf{D_{Rfn}}$	Receptor de tráfego de perturbação do fluxo número *fn*.
$\mathbf{N_{Rfn}}$	Receptor de tráfego de ruído do fluxo número *fn*.

A Figura 6.7 ilustra a distribuição de tráfego espacial para os fluxos simulados. O núcleo *CPU*, localizado no centro da rede, ativa cada núcleo para o início da transmissão/recepção de dados. Os nodos rotulados como R indicam um roteador com um núcleo gerador/receptor conectado. Considerando que o roteamento dos pacotes começa adotando o algoritmo XY, é possível observar áreas de *hot-spot*, onde um maior número de fluxos compete por recursos, sendo potenciais pontos de congestionamento. Portanto, os fluxos QoS devem evitar esta área no caso de congestionamento. Na Figura 6.7, a área de *hot-spot* é destacada.

Cada gerador de tráfego QoS envia 500 pacotes de 16 *flits*. A taxa média de injeção de pacotes é de 1.2 Gbps. Os geradores de tráfego de ruído injetam 125 pacotes de 16 *flits*, em uma taxa que varia entre 1.5 Gbps e 2.0 Gbps. Tráfegos de perturbação são gerados em taxas que variam entre 1 Gbps e 1.4 Gbps. São gerados 265 pacotes relativos ao tráfego de perturbação, possuindo cada pacote 16 *flits*.

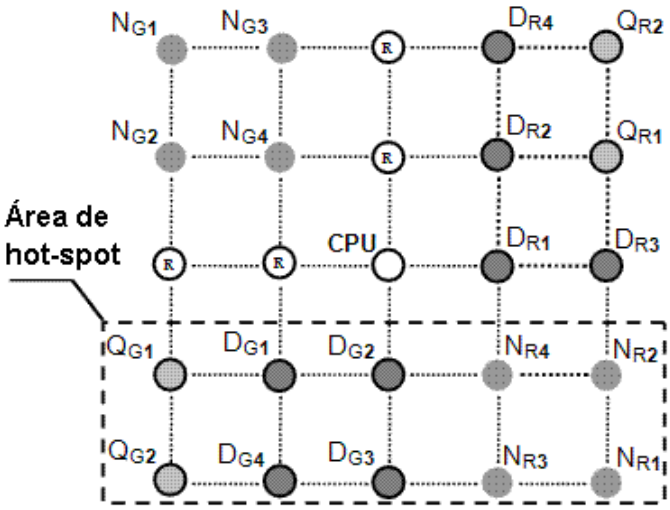


Figura 6.7 – Distribuição espacial de tráfego adotada.

6.2.1.1 AVALIAÇÃO DE LATÊNCIA E TEMPO DE *FLITS*

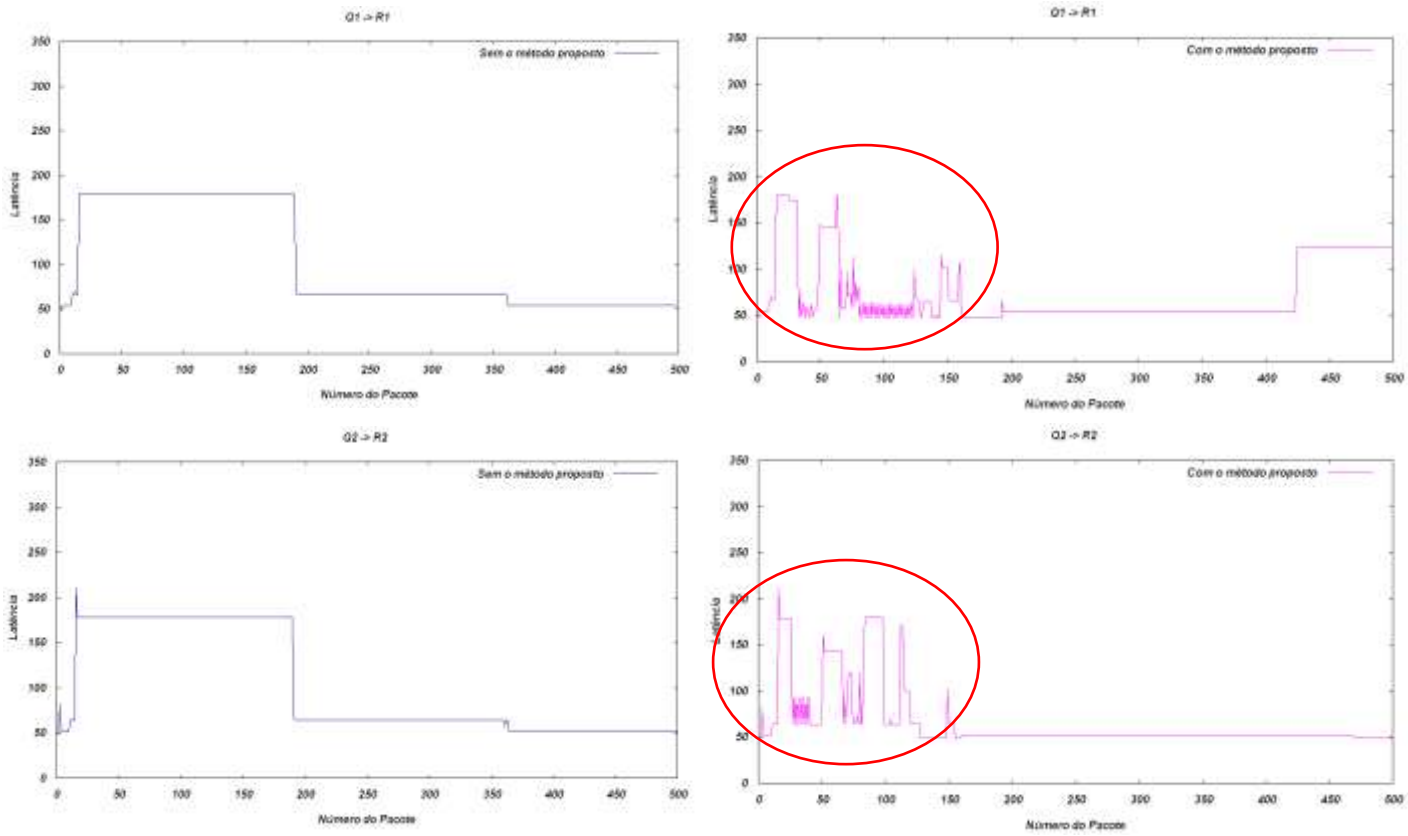
O parâmetro de desempenho adotado para avaliação é a latência. Ela é obtida para cada pacote, sendo o tempo gasto entre a transmissão e a recepção do pacote no nodo destino. Esta análise considera a latência apenas para os fluxos QoS, gerados pelos nodos Q_{G1} e Q_{G2} (Figura 6.7) A Tabela 6.5 apresenta os valores de latência média obtidos e seu desvio padrão, em ciclos de relógio, para o cenário definidos.

Tabela 6.5 – Valores de latência de pacotes fluxos QoS, em ciclos de relógio (M: Média; DP: Desvio Padrão).

Fluxo	Sem o método proposto	Com o método proposto	Redução (-) ou aumento (+)?
$Q_{G1} \rightarrow Q_{R1}$ (7 hops)	M: 100.95 DP: 57.31	M: 65.9 DP: 34.5	M: - 34.7% DP: **- 39.8%**
$Q_{G2} \rightarrow Q_{R2}$ (9 hops)	M: 103.11 DP: 57.2	M: 73.80 DP: 35.32	M: - 28.4% DP: - 38.3%

Os resultados apresentados mostram que ambos os fluxos QoS têm reduzidas as suas latências médias nos três cenários simulados. A redução na latência média é similar para os dois fluxos QoS. Entretanto, o desvio padrão apresentado por Q_{G1} apresenta a maior redução. Isto é justificado pelas possibilidades fora da área de *hot-spot* que podem ser exploradas ao se rotear este fluxo, onde um menor nível de concorrência é encontrado.

Os gráficos da Figura 6.8 mostram as latências para cada pacote nos dois fluxos QoS, onde se pode comparar o que acontece com os fluxos quando o método é aplicado (gráficos em (b)) ou não (gráficos em (a)). A primeira linha de gráficos corresponde ao fluxo Q1→R1 e a segunda linha corresponde ao fluxo Q2→R2. Nota-se para ambos os fluxos que a detecção do congestionamento ocorreu no inicio do fluxo de perturbação. A ocorrência de outros fluxos e mesmo a concorrência entre fluxos QoS leva às instabilidades destacadas na figura.



(a) (b)

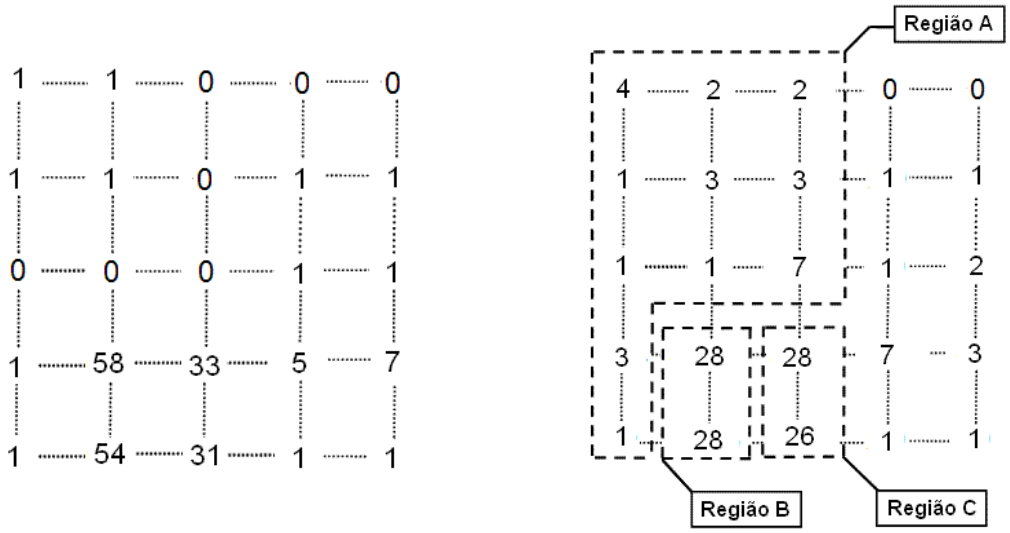
Figura 6.8 – Latências obtidas para pacotes gerados com requisitos de QoS (em ciclos de relógio).

A segunda métrica que foi avaliada é o tempo gasto pelos *flits* nos roteadores. Para isto, três regiões foram definidas: A, B e C. Cada região limita um grupo de roteadores com valores similares de tempo médio de espera dos *flits*. Estas regiões são ilustradas na Figura 6.9.

A Tabela 6.6 apresenta os valores médios de tempos de *flit* para as regiões apresentadas na Figura 6.9. A região B apresenta a maior redução comparada às outras regiões. Este fato é esperado, considerando que as mudanças no caminho de roteamento envolvem principalmente estas regiões, que é próxima de Q_{G1} e Q_{G2}. Os roteadores que pertencem à região A oferecem caminhos alternativos para os pacotes que passam através de regiões congestionadas, causadas pelo tráfego de perturbação, que não foram previamente utilizadas pelos fluxos gerados por Q_{G1} e Q_{G2}, como mostrado na Figura 6.9. Este fato justifica o aumento de valores de tempos de *flit* nesta região.

Tabela 6.6 – Tempos de *flit* em roteadores para as regiões especificadas.

Região	Com o método proposto	Sem o método proposto	Redução (-) ou aumento (+)?
A	1	2.5	+ 150%
B	56	28	- 50%
C	32	27	- 16%

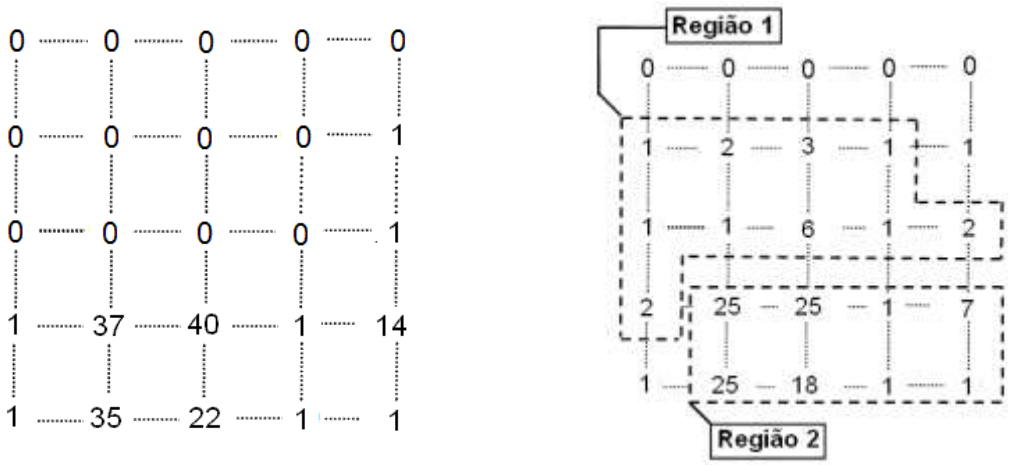


(a) valores obtidos quando o método de roteamento proposto não é adotado

(b) valores obtidos quando o método de roteamento proposto é adotado

Figura 6.9 – Tempo médio de pacotes, para cada região da NoC, para todos os fluxos (em ciclos de relógio).

Especificamente para os fluxos QoS, é ainda possível definir regiões de acordo com os tempos de espera de *flits* em roteadores (Figura 6.10). Notar que a ocupação da região 1 ocorre quando o método de roteamento dinâmico é adotado. A Tabela 6.7 apresenta os valores médios de tempos de *flit* nas regiões 1 e 2. É mostrado que este fato leva a um aumento de ocupação em 100%, comparado a valores obtidos quando somente o roteamento XY é adotado. A região 2 tem sua utilização reduzida em 39%.



(a) valores obtidos quando o método de roteamento proposto não é adotado

(b) valores obtidos quando o método de roteamento proposto é adotado

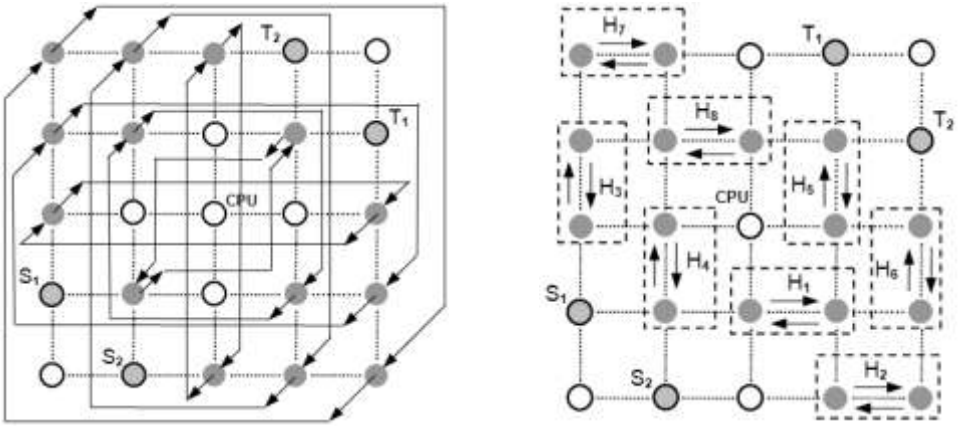
Figura 6.10 – Tempos de *flit* em roteadores para os fluxos QoS. As regiões 1 e 2 são destacadas.

Tabela 6.7 – Tempos de *flit* em regiões utilizadas por fluxos QoS.

Região	Sem o método proposto	Com o método proposto	Redução (-) ou adição (+)?
1	1	2	+ 100%
2	19	13	- 32%

6.2.2 CENÁRIO 2

A Figura 6.11 ilustra as distribuições espaciais de tráfego. A CPU, que é responsável pelo controle dos geradores de tráfego, é posicionada no centro da rede, a exemplo do cenário anterior. Os fluxos QoS, gerados pelos nodos S, executam o método de roteamento proposto. Os destinos de tráfego QoS, rotulados como T, analisam a condição de congestionamento dos caminhos sendo utilizados, e geram os pacotes do tipo ALARM. Em ambas as distribuições de tráfego, os fluxos QoS são S1→T1 e S2→T2, e os roteadores sombreados geram os fluxo concorrentes.



(a) (b)

Figura 6.11 – Distribuições de tráfego espacial, complemento e *hot-spot* (maior localidade).

Na distribuição complemento (Figura 6.11(a)), os fluxos que perturbam o tráfego QoS (denominados aqui *de perturbação*) possuem um mínimo de 5 e um máximo de 9 hops de distância entre origem e destino. Ainda que a biseção da rede seja a região com maior volume de tráfego, o padrão complemento distribui a carga uniformemente na rede. No padrão *hot-spot* (Figura 6.11(b)), os fluxos de perturbação são rotulados com a letra H, estando posicionados em oito diferentes regiões da rede. A distância entre cada par origem destino rotulado como H é de apenas dois hops, resultando em um maior grau de localidade em comparação à distribuição complemento.

Inicialmente, todos os fluxos executam o algoritmo de roteamento XY. Estes fluxos, controlados pela CPU, iniciam em momentos distintos da simulação, com o objetivo produzir um ambiente com tráfego dinâmico. Três cenários de simulação são aplicados em cada distribuição de tráfego, que se diferenciam pelo tamanho da mensagem, como detalhado na Tabela 6.8.

Tabela 6.8 – Cenários de tráfego.

Tipo de tráfego	Descrição	Tamanho do pacote (número de *flits*)	Tamanho da mensagem (número de pacotes)	Numero de pacotes
QoS	Tráfego com requisitos de QoS	16	8, 16 e 32	500
De perturbação	Fluxos de tráfego que perturbam os fluxos QoS	16	16	250

6.2.2.1 AVALIAÇÃO DE LATÊNCIA E TEMPO DE *FLITS*

A avaliação de latência considera apenas os pacotes dos fluxos QoS S1→T1 e S2→T2. A Tabela 6.9 apresenta as latências médias obtidas, assim como os valores de desvio padrão, em ciclos de relógio. O método de roteamento proposto não reduz as latências para a distribuição complemento. Isto é explicado pelo fato do tráfego ser uniformemente distribuído na rede, levando à falta de caminhos a serem explorados pelo roteamento adaptativo.

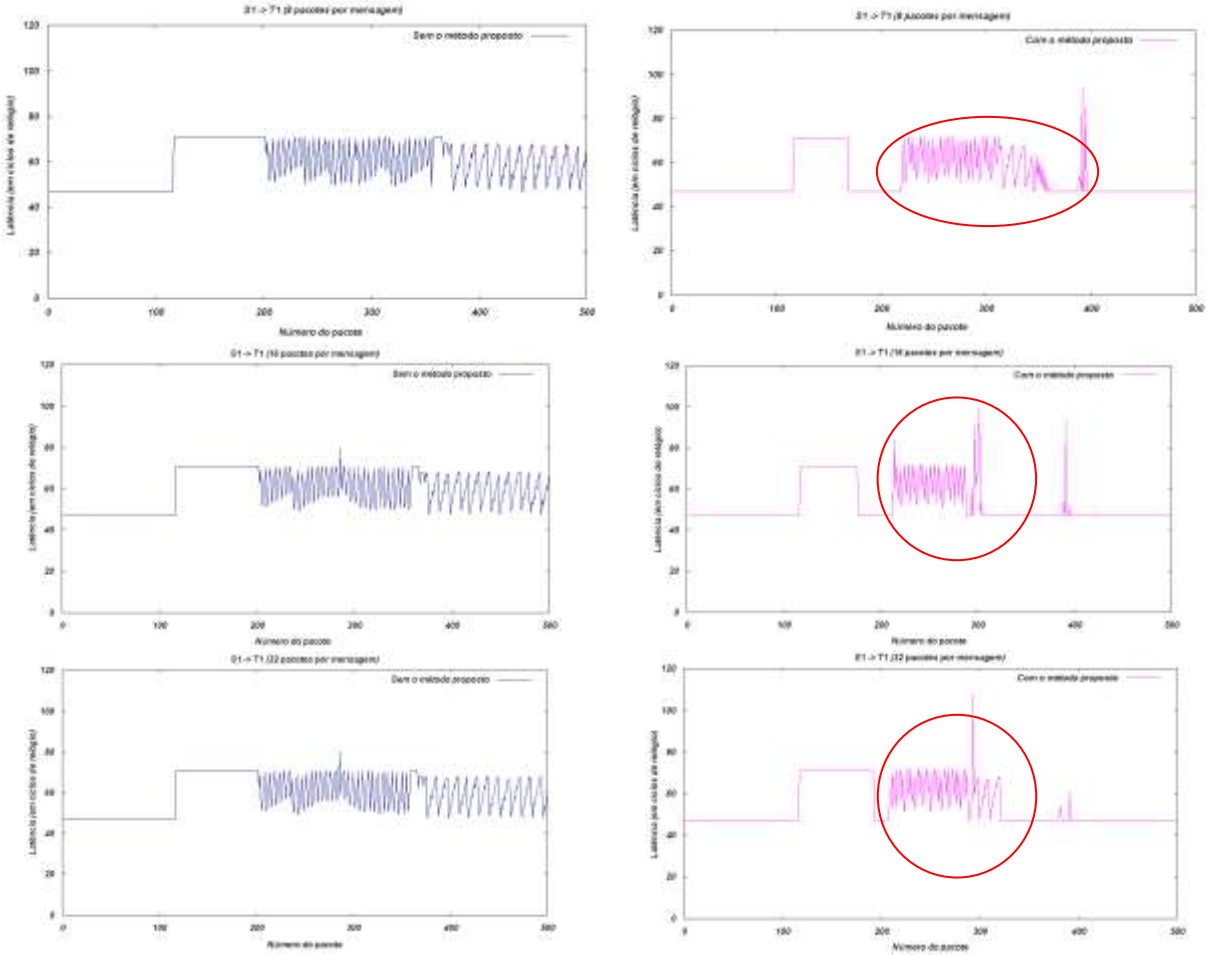
Tabela 6.9 – Resultados de latência de pacotes para fluxos QoS, em ciclos de relógio (M: Média, DP: Desvio Padrão).

Flow	Tamanho das mensagens para fluxos QoS (em número de pacotes)	***Distribuição de tráfego complemento***				***Distribuição de tráfego Hot-spot***			
		Sem o método proposto		**Com o método proposto**		Sem o método proposto		**Com o método proposto**	
		M	DP	**M**	**DP**	M	DP	**M**	**DP**
S1 → T1	8	65.9	19.3	**63.6 (-3.5%)**	**20.5**	59.2	9.5	**53.6 (-9.5%)**	**9.7**
	16	66.2	19.3	**63.8 (-3.6%)**	**19.9**	59.2	9.5	**52.8 (-10.8%)**	**10.0**
	32	65,7	19.2	**64.4 (-1.9%)**	**21.9**	59.3	9.5	**54.2 (-8.6%)**	**10.3**
S2 → T2	8	47.1	1.4	**48.6 (+3.2%)**	**5.2**	58.4	11.5	**51.1 (-12.5%)**	**8.9**
	16	47.2	1.9	**48.7 (+3.2%)**	**5.1**	57.9	11.2	**51.2 (-11.6%)**	**9.1**
	32	47.3	2.6	**48.3 (+2.1%)**	**4.5**	57.9	11.2	**51.8 (-10.5%)**	**9.5**

Por outro lado, com o cenário *hot-spot* é possível observar que a latência média reduz em média 10% e o desvio padrão reduz em 7%. Neste cenário, o congestionamento em regiões específicas da rede são introduzidos, e o método proposto detectou e evitou estas regiões.

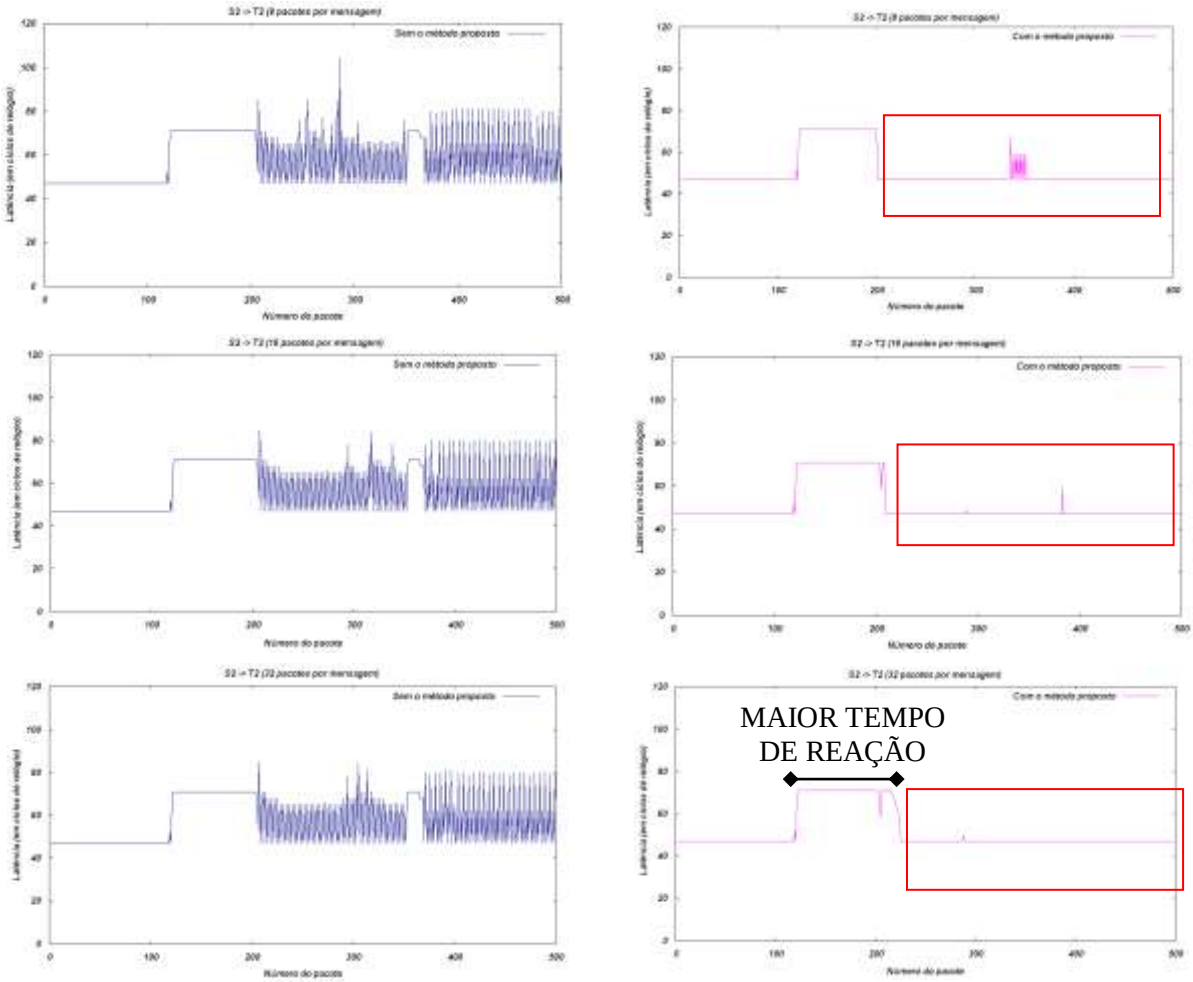
As Figura 6.12 e 6.13 apresentam as latências ocorridas durante as entregas de pacotes para os fluxos QoS considerados, de acordo com os três tamanhos de mensagens definidos, sendo que na primeira linha as mensagens QoS tem 8 pacotes, na segunda linha 16 pacotes e na terceira linha 16 pacotes. O padrão de tráfego é o *hot-spot*. A Figura 6.12 apresenta as latências obtidas para o fluxo S1→T1. Em todos os cenários observa-se a necessidade de haver mais de uma troca de caminho, pela dificuldade do algoritmo em detectar o congestionamento, fato expressado pela variabilidade das latências, o que ocasiona *jitter* (destaque na Figura).

Por outro lado, o fluxo S2→T2 (Figura 6.13) apresenta melhor desempenho, visto que pelo tráfego concorrente ser mais intenso, o algoritmo de roteamento adaptativo possui melhores condições de detectar congestionamento (comparar com a Figura 6.12). Desta forma, o caminho sem congestionamento foi escolhido com melhor precisão, levando as latências médias a partir do pacote de número 200 ao seu valor mínimo.



(a) (b)

Figura 6.12 – Latências obtidas no fluxo gerado pelo núcleo S1 e recebido pelo núcleo T1.



(a) (b)

Figura 6.13 – Latências obtidas no fluxo gerado pelo núcleo S1 e recebido pelo núcleo T1.

6.2.2.2 REAÇÃO A EVENTOS DE CONGESTIONAMENTO

O *hot-spot* é um cenário que permite uma melhor exploração de caminhos. A reação do mecanismo de roteamento é a quantidade de pacotes enviados com um nível indesejado de QoS. Esta métrica é avaliada de acordo com o número de pacotes em cada mensagem, que é a granularidade de análise deste algoritmo. Considerando o fluxo S2→ T2, o pior caso é para mensagens de QoS com 32 pacotes (comparar na Figura 6.12(a) os resultados entre mensagens de 8 e 32 pacotes). Isto mostra que mensagens maiores levam a um tratamento tardio dos eventos de congestionamento i.e. a reação do algoritmo a eventos de congestionamento levam um tempo maior.

Mensagens com menor número de pacotes atingem a latência mínima no pacote de número 200. Entretanto, alguns valores maiores de latência são observados nos pacotes 330 a 350. O uso de mensagens maiores, leva a um tempo de reação maior. Como o número de eventos monitorados também aumenta em mensagens maiores, o algoritmo é habilitado a encontrar um caminho menos congestionado, eliminando o ruído observado quando da utilização de mensagens com 32 pacotes.

6.2.3 CENÁRIO 3

Este terceiro cenário tem como objetivo obter resultados com a NoC Hermes-PLR, de modo a realizar a sua validação. Para fins de avaliação será considerada a latência de pacotes do fluxo QoS, que é aquele cujos transmissor e receptor executam o método de roteamento adaptativo apresentado nesta tese. A distribuição de tráfego espacial é ilustrada na Figura 6.14, em uma rede malha 5x5. Oito fluxos geram tráfego para perturbar o fluxo QoS, sendo rotulados como *T* os transmissores de perturbação e como *R* os receptores de perturbação.

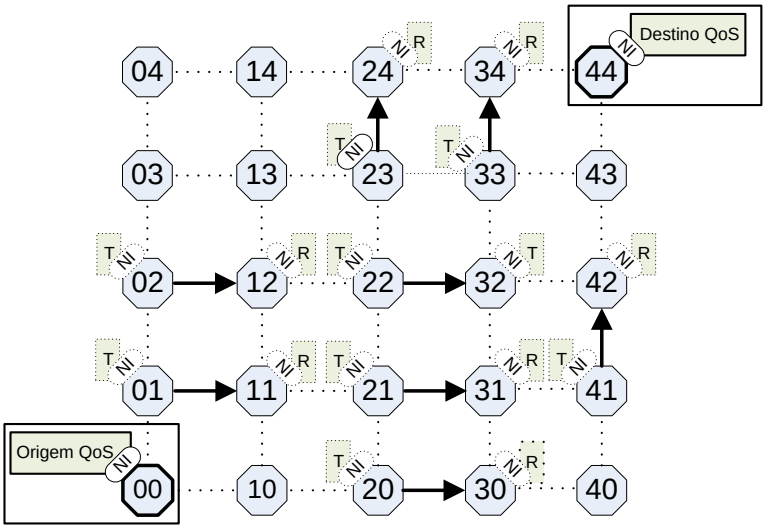


Figura 6.14 – Distribuição espacial de tráfego.

Todos os geradores de tráfego transmitem 1200 pacotes na taxa de 240 Mbps, que corresponde a uma carga de 20%. Cada gerador de tráfego BE transmite três mensagens com 400 pacotes, sendo que cada o intervalo entre mensagens é escolhido de maneira aleatória, de modo a produzir um tráfego global com dinamicidade na ocorrência de *hot-spots*. O gerador de tráfego QoS transmite mensagens compostas por 20, 80 e 160 pacotes para cada cenário simulado.

Os monitores de tráfego conectados aos roteadores capturam a quantidade de *flits* de entrada em uma janela de 1000 ciclos de relógio. Considerando como parâmetro de QoS uma carga de 20%, uma quantidade registrada igual ou acima de de 200 *flits* indica que o roteador monitorado está congestionado.

A Tabela 6.10 apresenta os resultados obtidos para latência média e desvio padrão. Também são apresentados o número de alterações de caminho executadas quando congestionamento é detectado, quando mensagens de 20.80 e 160 pacotes são utilizadas. Para efeitos de comparação, são também resultados com a utilização do roteamento XY (Seção 5.2) e *oblivious* (Seção 5.3). O roteamento XY é determinístico, sendo que todos os pacotes percorrem o mesmo caminho da fonte até o destino. No roteamento *oblivious*, alterações de caminho são realizadas para cada pacote, sem qualquer conhecimento do estado de congestionamento da rede.

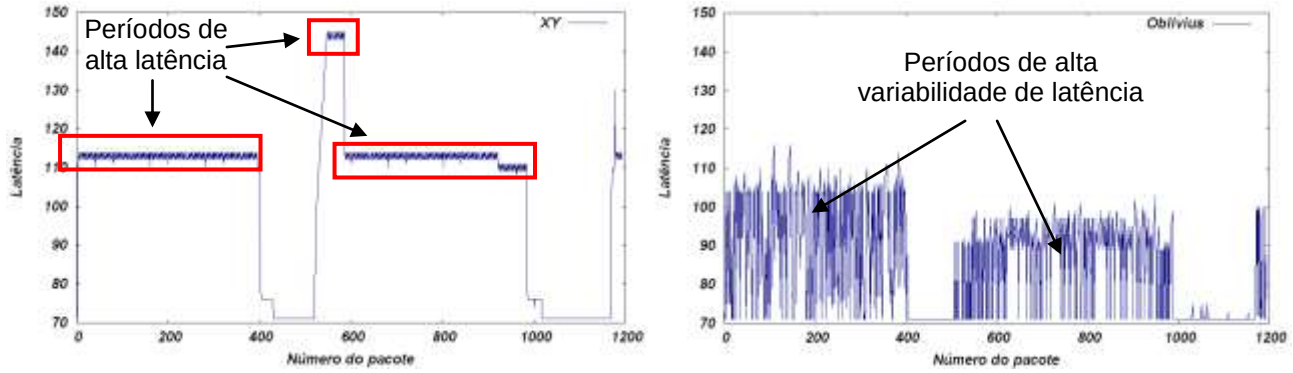
6.2.3.1 AVALIAÇÃO DE DESEMPENHO

Os resultados a seguir mostram as latências ocorridas para os pacotes do fluxo QoS. De acordo com os resultados obtidos e apresentados na Tabela 6.10, é possível observar o compromisso existente entre o tamanho da mensagem, a latência média e o percentual de pacotes com latência mínima. Quando mensagens curtas são utilizadas, há um número maior de avaliações de caminhos para toda a simulação. Isto significa que cada roteador é pouco avaliado durante a transmissão de uma mensagem, visto que são poucos os eventos de monitoração. Mensagens maiores, no entanto, levam a um tratamento mais tardio do congestionamento, realizado ao final da transmissão de cada mensagem. O tempo de reação de uma monitoração com a utilização de mensagens maiores é, portanto, mais lento.

Tabela 6.10 – Resultados para fluxos QoS.

Método aplicado	Latência média	Desvio padrão	Modificações de caminho	% de pacotes com latência mínima
Sem monitoração (roteamento XY)	103.4	19.4	0	25%
Sem monitoração (roteamento oblivious)	85.4	13.3	1200	42%
*Com monitoração (mensagens com **20** pacotes)*	86.5	17.6	42	43%
Com monitoração (mensagens com 80 pacotes)	***81.3***	***15.4***	***13***	***66%***
*Com monitoração (mensagens com **160** pacotes)*	92.7	22.8	7	45%

A utilização do algoritmo XY resultou, como esperado, maiores valores de latência (Figura 6.15(a)), considerando que não é possível alterar o caminho quando o congestionamento ocorre. Por outro lado, a utilização do algoritmo de roteamento *oblivious* resulta em valores de latência média comparáveis com os obtidos com a utilização de monitoração, além de apresentarem grande variabilidade (Figura 6.15(b)). Um grande número de alterações de caminho acontece quando da adoção desta abordagem, o que pode levar a perturbações nos outros fluxos que executam sobre a NoC.



(a) (b)

Figura 6.15 – Latências obtidas quando é utilizado os algoritmos (a)XY e (b)*oblivious*.

Utilizando o método proposto de monitoração e roteamento adaptativo na origem, o melhor resultado foi obtido quando mensagens com 80 pacotes foram transmitidas (Figura 6.16). Observam-se neste caso, períodos maiores de latência mínima em comparação com a utilização de mensagens de 20 (Figura 6.17) e 160 pacotes (Figura 6.18). O tempo gasto para o envio de 80 mensagens mostrou para este cenário, uma análise do caminho satisfatória, considerando o baixo número de alterações de caminho executadas.

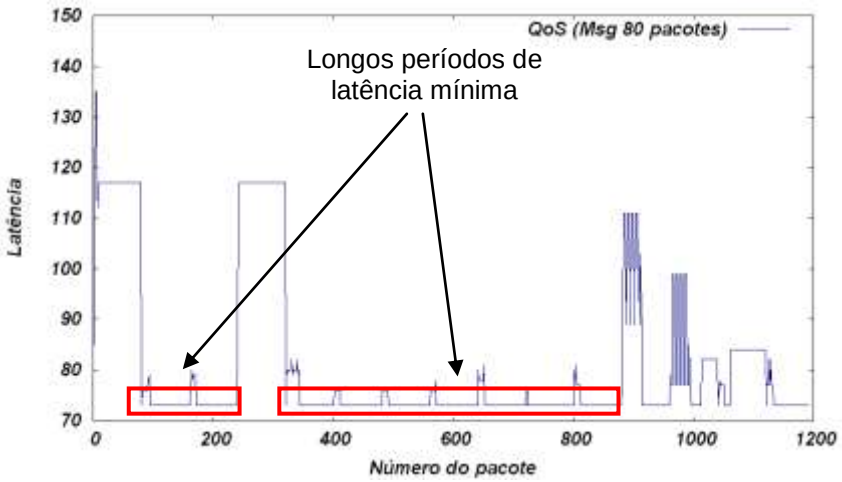


Figura 6.16 – Latências obtidas quando mensagens com 80 pacotes são transmitidas.

Quando mensagens de 20 pacotes são empregadas, não há uma precisão satisfatória na análise do caminho. A Figura 6.17 ilustra períodos curtos de alta latência alternando com aqueles que apresentam latência mínima. Este comportamento ilustra a instabilidade quando se adotam mensagens com poucos pacotes, onde são reduzidas as quantidades de análises em cada roteador.

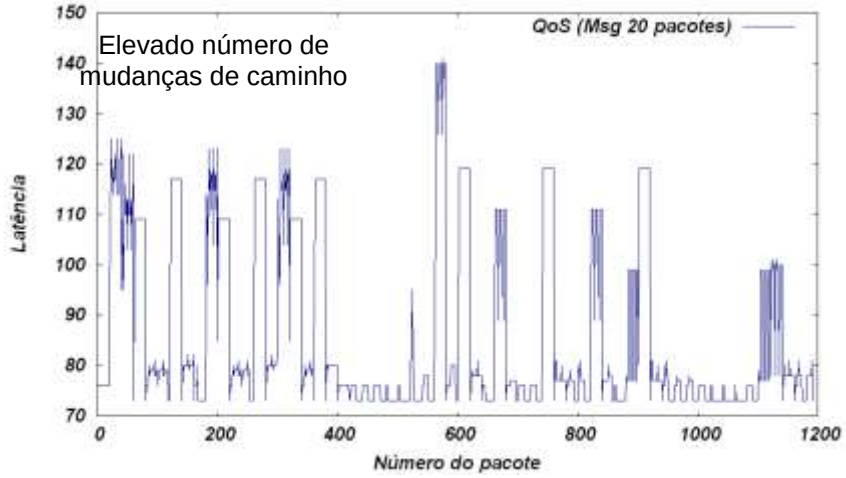


Figura 6.17 – Latências obtidas quando mensagens com 20 pacotes são transmitidas.

Por outro lado, quando mensagens com 160 mensagens são empregadas (Figura 6.18), o tempo de reação é comparativamente alto, levando pacotes a serem transmitidos em caminhos congestionados por um período de tempo maior. A mudança para um caminho também congestionado é custosa, devido ao fato da grande quantidade de pacotes da nova mensagem ser obrigada a trafegar no mesmo.

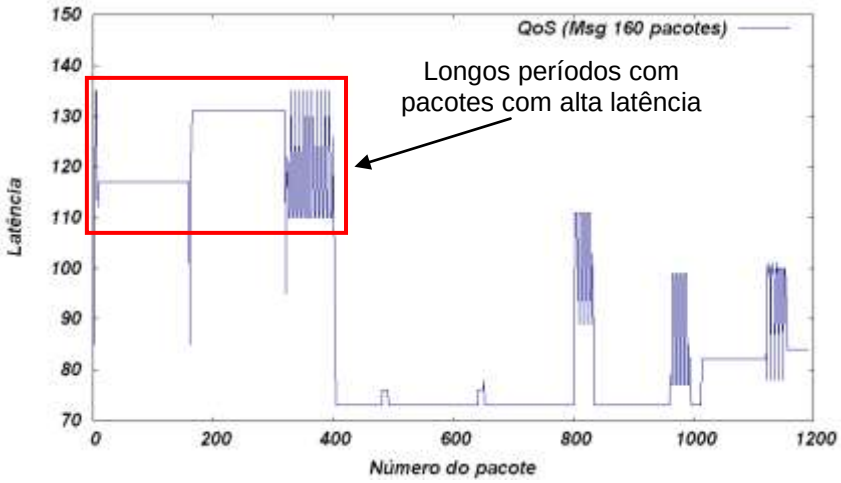


Figura 6.18 – Latências obtidas quando mensagens com 160 pacotes são transmitidas.

6.2.4 CONCLUSÕES SOBRE OS EXPERIMENTOS COM ROTEAMENTO ADAPTATIVO

O conjunto de experimentos aqui apresentado validou o método para roteamento adaptativo proposto nos Capítulos 4 e 5. Foram considerados parâmetros importantes para a avaliação de algoritmos de roteamento. A latência foi o parâmetro de desempenho adotado. Tráfegos com pouca localidade espacial tendem a não produzir pontos de congestionamento, eliminando a necessidade de um roteamento que considere o caminho do fluxo. Para pontos esporadicamente congestionados, algoritmos distribuídos, como Dyad e DyXY podem ser mais apropriados.

No entanto, para tráfegos localizados, ocorre a produção de pontos quentes na rede. Considerando ainda o seu aparecimento em várias regiões da rede, torna-se necessário um mecanismo para verificar o congestionamento sob um ponto de vista global. A adoção de roteamento na origem com visão de múltiplos pontos de congestionamento torna possível a descoberta de caminhos livres de congestionamento, o que se pôde observar nos resultados obtidos. Ainda, uma melhor exploração da rede foi observada, onde os tráfego QoS puderam trafegar com melhor desempenho.

Observou-se ainda, que há um compromisso entre o tamanho de mensagens e a reatividade do algoritmo frente a eventos de congestionamento. Isso porque em cada pacote de uma mensagem é verificado o congestionamento em um ponto da rede. Logo, quanto maior o número de pacotes em uma mensagem, mais vezes os pontos do caminho de um fluxo QoS podem ser observados. Mensagens curtas levam a uma reatividade mais rápida, porém, menos precisa, pelo fato de haver uma amostra pequena do congestionamento, o que pode levar o algoritmo de roteamento a calcular um caminho com algum grau de congestionamento. Mensagens maiores levam mais tempo para serem avaliadas, mas levam a um caminho com menos congestionamento, pelo fato das medições sobre os pontos monitorados ser mais precisa.

Desta forma, conclui-se que a estratégia de roteamento é adequada quando fluxos QoS concorrem com fluxos com alto grau de localidade. Além disso, observou-se que há um compromisso entre o tamanho de mensagens (que expressa número de pontos

monitorados e quantas vezes estes pontos são monitorados) e a precisão do algoritmo de roteamento.

7. CONCLUSÃO E TRABALHOS FUTUROS

Este Capítulo apresenta as contribuições da Tese. Posteriormente, são apresentadas as publicações obtidas, que tem como foco os métodos desenvolvidos. Finalmente, são apresentadas as conclusões e trabalhos que podem ser desenvolvidos.

7.1 CONTRIBUIÇÕES DO TRABALHO

Dentre as contribuições do trabalho podem-se citar as seguintes como principais:

Método para dimensionamento de *D-buffers* – A primeira contribuição do trabalho é um método para dimensionamento de *buffers* em interfaces de rede em NoCs (denominados *D-buffers*). A função deste *buffer* é desacoplar os efeitos do empacotamento de dados e do tráfego da rede nas taxas originais da aplicação. Estes dados chegam ao IP destino com jitter, sendo que o mesmo pode trazer dois problemas na aplicação-alvo: (*i*) a aplicação não conseguir consumir dados do *D-buffer*, pelo fato do mesmo estar vazio; (*ii*) os dados da rede não poderem ser inseridos no *buffer*, pelo mesmo estar cheio. O método de dimensionamento de *D-buffers* ocorre em tempo de projeto, e considera traces de tráfego na saída de IPs transmissores e entradas de IPs receptores de tráfego. Este trabalho diferencia-se da maior parte dos demais trabalhos relacionados, onde os *buffers* dimensionados são aqueles conectados às portas de entrada dos roteadores da NoC.

Método para estabelecimento de sessão de monitoração de tráfego QoS – A segunda contribuição do trabalho consiste de um protocolo para gerenciamento de monitoração de eventos que possam vir a degradar o desempenho de fluxos QoS. Tal protocolo é implementado através da transmissão de pacotes para abertura e fechamento de sessão, configuração de tabelas para armazenamento de informações de tráfego e o aproveitamento de pacotes de dados da aplicação para transmissão de informações monitoradas. Através destes pacotes as interfaces de transmissão e recepção têm condições de verificar o estado de congestionamento do tráfego pertence ao caminho que os dados percorrem na rede. Esta visão de congestionamento difere-se da maior parte dos trabalhos do estado da arte, onde o congestionamento tem uma visão local, considerando o estado de congestionamento de roteadores vizinhos.

Algoritmo de roteamento adaptativo na origem – A terceira contribuição do trabalho trata de um algoritmo de roteamento adaptativo na origem, que escolhe rotas para pacotes com base no que acontece no caminho de fluxos QoS. Propostas de algoritmos adaptativos distribuídos são maioria nos trabalhos relacionados a roteamento. Estes algoritmos apresentam, no entanto, uma visão local do congestionamento, limitando-se a poucos vizinhos. Verificou-se que o algoritmo é eficiente para desviar de fluxos concorrentes com baixa localidade, que ocasionam *hot-spots*. Foi ainda realizada um analise da influência do tamanho de mensagens na rapidez com que o algoritmo reage ao congestionamento e também na precisão de se encontrar pontos de congestionamento.

7.2 PUBLICAÇÕES

As publicações abaixo contribuíram para a definição e desenvolvimento dos métodos apresentados nesta tese.

TEDESCO, Leonel Pablo; MELLO, Aline Vieira de; GIACOMET, Leonardo; CALAZANS, Ney Laert Vilar; MORAES, Fernando Gehm.
Application Driven Traffic Modeling for NoCs.
In: SBCCI, 2006, pp. 62-67.

TEDESCO, Leonel Pablo; CALAZANS, Ney Laert Vilar; MORAES, Fernando Gehm.
Buffer Sizing for QoS Flows in Wormhole Packet Switching NoCs.
In: SBCCI, 2007, pp. 99-104.

TEDESCO, Leonel Pablo ; CALAZANS, Ney Laert Vilar; MORAES, Fernando Gehm .
Buffer Sizing for Multimedia Flows in Packet-Switching NoCs.
JICS. Journal of Integrated Circuits and Systems, vol. 3, pp. 46-56, 2008.

TEDESCO, Leonel Pablo; CLERMIDY, Fabien; MORAES, Fernando Gehm.
A Path-Load Based Adaptive Routing Algorithm for Networks-on-Chip.
In: SBCCI, 2009, pp. 141-146.

TEDESCO, Leonel Pablo; CLERMIDY, Fabien; MORAES, Fernando Gehm.
A monitoring and adaptive routing mechanism for QoS traffic on mesh NoC architectures.
In: CODES+ISSS, 2009, pp. 109-117.

TEDESCO, Leonel Pablo; CLERMIDY, Fabien; MORAES, Fernando Gehm.
Adaptive Congestion-Based Source Routing Algorithm for Mesh NoCs.
Submetido para: JPDC, Journal of Parallel and Distributed Computing, 2010.

TEDESCO, Leonel Pablo; MORAES, Fernando Gehm.
Implementation and Evaluation of a Congestion Aware Routing Algorithm for Networks-on-Chip.
Submetido para: SBCCI, 2010.

7.3 CONCLUSÕES

O número de funcionalidades que são continuamente acrescidas aos sistemas embarcados faz com o tráfego em chip torne-se mais complexo e imprevisível. Desta forma, faz-se necessário o emprego de estruturas de comunicação em chip que possam tratar dessa imprevisibilidade, de modo a garantir os serviços para tráfego com requisitos rígidos de comunicação. O emprego de NoCs como meio de comunicação para as próximas gerações de MPSoCs é uma realidade, considerando o grande investimento em projetos que utilizam este tipo de estrutura, realizado por fabricantes de processadores [INT10a][TIL10].

Este trabalho apresentou dois métodos para adaptação da rede aos requisitos de QoS em comunicações apresentadas por aplicações que executam em MPSoCs: dimensionamento de *buffers* de desacoplamento (*D-buffers*) e roteamento adaptativo.

O dimensionamento de *D-buffers* é um método empregado em tempo de projeto, que tem por objetivo minimizar ou ainda eliminar a ocorrência de *jitter* em dados que são recebidos por IPs que possuem de QoS. Diversos impactos foram observados após a realização dos experimentos envolvendo o dimensionamento de *D-buffers*. A análise da utilização de *D-buffers* mostrou que tráfegos QoS com taxa constante de transmissão utilizam o *buffer* de maneira uniforme. Entretanto, tráfegos gerados a partir de *traces* têm maior variabilidade, exigindo *D-buffers* maiores com *threshold* diferente de zero. Esta

observação demonstra a necessidade dos *D-buffers* para aplicações reais. A violação de prazos também foi estudada, verificando-se que a eliminação de *slots* de *D-buffers* afeta com menor intensidade tráfegos com taxas variáveis (como encontrado em *traces*). Alguns pacotes de maior tamanho, encontrados em *traces*, são minoria, podendo ser descartados se isso for tolerável pela aplicação.

O segundo método de adaptação, empregado em tempo de execução, é um algoritmo de roteamento adaptativo na origem. Este algoritmo foi desenvolvido no CEA-LETI durante o período de estágio sanduíche, sendo incorporado à NoC Hermes ao final do doutorado, criando-se a estrutura Hermes-PLR. Este algoritmo tem como principal característica a visualização de eventos de congestionamento no caminho do fluxo, diferentemente do que acontece na maior parte dos trabalhos do estado da arte, onde as decisões de roteamento são baseadas em decisões locais, considerando apenas o estado de congestionamento em roteadores vizinhos.

O suporte para este algoritmo é oferecido por um protocolo de gerenciamento de sessão de monitoração de tráfego, monitoração distribuída, e tabelas que armazenam o estado dos roteadores. Após a abertura de uma sessão de monitoração, mensagens compostas de um ou mais pacotes coletam informações sobre o estado da rede, alimentando uma tabela no destino do tráfego, que é analisada a cada final de mensagem. Se um determinado estado de congestionamento crítico é atingido, a NI envia um pacote que alerta a origem do fluxo que o caminho está congestionado. Com base neste pacote, o algoritmo escolhe um caminho livre de congestionamento. Uma mensagem só é transmitida após a chegada de um pacote de alarme, garantindo a ordem de entrega dos pacotes, o que normalmente não é tratado em roteamento adaptativo distribuído.

Os experimentos conduzidos demonstram a eficácia do método quando tráfegos QoS competem com tráfegos altamente localizados, que geram regiões de *hot-spot*. Estes tráfegos ocorrem em sistemas reais, como por exemplo, nos enlaces próximos a CPU de controle da plataforma Magali, ou em enlaces próximos a memórias compartilhadas. O encontro de caminhos livres leva a redução da latência dos pacotes QoS ao seu valor mínimo. Outro benefício é o balanceamento da carga na rede, onde há menor ocorrência de concorrência entre os fluxos. O custo para isso, no entanto, é o tempo de reatividade do algoritmo, que depende do tamanho da mensagem. Os resultados obtidos mostram uma relação entre a reatividade do algoritmo frente a eventos de congestionamento e o tamanho da mensagem. Mensagens maiores demoram mais a encontrar um caminho, mas quando o mesmo é encontrado, a latência atinge valores mínimos até o final de sua transmissão. Mensagens menores reagem de maneira mais rápida ao congestionamento, mas geralmente não encontram o melhor caminho de imediato.

O algoritmo de roteamento implementado atende aos objetivos propostos na Seção 1.3, provando que roteamento adaptativo na origem com monitoração distribuído do tráfego pode ser um eficiente método para ser empregado em redes intra-chip para atender aos requisitos das aplicações com restrições de qualidade de serviço.

Finalmente, os métodos de dimensionamento de *buffers* e roteamento com monitoração distribuída são genéricos. O dimensionamento de *buffers* é genérico, pois é executado externamente à rede, na interface de rede, não dependendo de uma NoC específica. O roteamento é também genérico, tendo como principais restrições ser na origem e ser adaptativo. A partir destas duas restrições, pode-se empregá-lo com outras topologias ou com outros algoritmos de roteamento, diferentes do WF-EF.

7.4 TRABALHOS FUTUROS

Elenca-se como sugestões de trabalhos futuros:

Definição de tamanho ideal de mensagens para fluxos QoS. Durante a transmissão de uma mensagem, cada pacote captura informações de congestionamento de um roteador específico. A transmissão de mensagens com muitos pacotes faz com que um número maior de amostras de congestionamento sejam capturadas e analisadas, aumentando a precisão da monitoração. Entretanto, a reatividade pode ser comprometida, uma vez que um novo caminho só é adotado a partir da transmissão da próxima mensagem do fluxo. Um método para encontrar um tamanho de mensagens que apresente reatividade rápida e precisa é a primeira sugestão para trabalho futuro. Pode-se também avaliar a alteração do cabeçalho do pacote para capturar mais de uma informação de monitoração ao longo do caminho.

Definição de pontos estratégicos para monitoração. A estratégia de monitoração apresentada considera todos os hops que fazem parte do caminho de um fluxo QoS. Uma alternativa a este método é a seleção de pontos específicos de monitoração, que podem ser escolhidos em tempo de execução. Este método pode fazer com que a precisão da monitoração aumente, podendo diminuir também o número de pacotes que compõem as mensagens.

Avaliação de novas métricas de monitoração e níveis de congestionamento. Neste trabalho foram considerados o tempo de *flits* em roteadores e a taxa de entrada dos *flits*. A maior parte do estado da arte considera a ocupação de *buffers* de entrada. Outros trabalhos consideram a percentual de conexões estabelecidas. A terceira sugestão para trabalhos futuros é o estudo de outras métricas para avaliação de congestionamento e seus respectivos níveis, que devem ser obedecidos de forma a atender aos requisitos da aplicação-alvo.

Adoção de abordagens mistas de roteamento. De acordo com os experimentos realizados com distribuições de tráfego com baixa localidade, o algoritmo de roteamento adaptativo na origem não trouxe redução significativa de latência. Sugere-se como trabalho futuro a criação de um método que capture as propriedades do tráfego da aplicação alvo e decida para cada par origem-destino o tipo de roteamento que ele deve executar.

REFERÊNCIAS BIBLIOGRÁFICAS

[ALF08] Al Faruque, M.; Ebi, T.; Henkel, J. “ROAdNoC: Runtime Observability for an Adaptive”. In: Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD’08), 2008, pp. 543-548.

[AND98] Andersen, A. T.; Nielsen, B. F. “A Markovian Approach for Modeling Packet Traffic with Long-range Dependence”. IEEE Journal on Selected Areas in Communications, vol. 16-5, Junho 1998, pp. 719-732.

[ASC08] Ascia, G.; Catania, V.; Palesi, M.; Patti, D. “Implementation and Analysis of a New Selection Strategy for Adaptative Routing in Networks-on-Chip”. IEEE Transactions on Computers, vol. 57-6, Junho 2008, pp. 809-820.

[BAR07] Barcelos, D.; Brião, E.; Wagner, F. “A Hybrid Memory Organization to Enhance Task Migration”. In: Symposium on Integrated Circuits and Systems Design (SBCCI’07), 2007, pp. 282-287.

[BEN01] Benini, L.; Micheli, G. “Powering Networks on Chip”. In: International Symposium on System Synthesis (ISSS’01), 2001, pp. 33-38.

[BEN02] Benini, L.; Micheli, G. “Network on Chips: A New SoC Paradigm”. IEEE Computer, vol. 35-1, Janeiro 2002, pp. 70-78.

[BER04] Bertozzi, D.; Benini, L. “Xpipes: A Network-on-chip Architecture for Gigascale Systems-on-Chip”. IEEE Circuits and Systems Magazine, vol. 4-2, Setembro 2004, pp. 18-31.

[BER05] Bertozzi, D.; Jalabert, A.; Murali, S.; Tamhankar, R.; Stergiou, S.; Benini, L.; De Micheli, G. “NoC Synthesis Flow for Customized Domain Specific Multiprocessor Systems-on-Chip”. IEEE Transactions on Parallel and Distributed Systems, vol. 16-2, Fevereiro 2005, pp. 113-129.

[BER95] Beran, J.; Sherman, R.; Taqqu, M.; Willinger, W. “Long-Range Dependence in Variable-Bit-Rate Video Traffic”. IEEE Transactions on Communications, vol. 43-2/3/4, Fev-Mar-Abr 1995, pp. 1566-1579.

[BOR88] Borkar, S.; Cohn, R.; Cox, G.; Gleason, S.; Gross, T.; Kung, H.; Lam, M.; Moore, B.; Peterson, C.; Pieper, J.; Rankin, L.; Tseng, P.; Sutton, J.; Urbanski, J.; Webb, J. “iWarp: an Integrated Solution to High-Speed Parallel Computing”. In: Proceedings of the Supercomputing ’88, 1988, pp. 330-339.

[BOL04] Bolotin E.; Cidon, I.; Ginosar, R.; Kolodny, A. “QnoC: QoS Architecture and Design Process for Network on Chip”. Journal of Systems Architecture: The EUROMICRO Journal, vol. 50-(2-3), Fevereiro 2004, pp. 105-128.

[BOL05] Boland, J.; Thibeault, C.; Zilic, Z. “Using Matlab and Simulink in a SystemC verification environment”. In: DVCON’05, 2005, 7p.

[BOU06] Bouhraoua, A; Elrabaa, M. E. “A High-Throughput Network-on-Chip Architecture for Systems-on-Chip Interconnect”. In: International Symposium on System-on-Chip (SOC’06), 2006, pp. 127-130.

[BRA07] van den Brand, J.; Ciordas, C.; Goossens, K.; Basten, T. “Congestion-Controlled Best-Effort Communication for Networks-on-Chip”. In: Proceedings of the Design, Automation and Test in Europe (DATE’07), 2007, pp. 1-6.

[CAR08] Carara, E.; Calazans, N.; Moraes, F. “A New Router Architecture for High-Performance Intrachip Networks”. Journal Integrated Circuits and Systems, vol. 3-1, Março 2008, pp. 23-31.

[CAR09] Carro, L.; Beck, A. “Adaptability: the Key for Future Embedded Systems”. Capturado em: http://esweek09.inrialpes.fr/tutorials/tutorial2.shtml, Outubro 2009.

[CAR09a] Carvalho, E. “Mapeamento Dinâmico de Tarefas em MPSoCs Heterogêneos baseados em NoCs”. Tese de Doutorado,PPGCC-FACIN-PUCRS, 2009, 168 p.

[CHA04] Chandra, V.; Xu, A.; Schmit, H.; Pileggi, L. “An interconnect channel design methodology for high performance integrated circuits”. In: Proceedings of the Design, Automation and Test in Europe (DATE’04), 2004, pp. 1138-1143.

[CHI00] Chiu, G. “The Odd-even Turn Model for Adaptive Routing”. IEEE Transactions on Parallel and Distributed Systems, vol. 11-7, Julho 2000, pp. 729-738.

[CIO04] Ciordas, C.; Basten, T.; Radulescu, A.; Goossens, K.; Meerbergen, J. “An Event-based Network-on-Chip Monitoring Service”. In: 9th IEEE International High-Level Design Validation and Test Workshop (HLDVT’04), 2004, pp. 149-154.

[CIO06] Ciordas, C.; Goossens, K.; Basten, T. “NoC Monitoring: Impact on the Design Flow”. In: Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS’06), 2006, pp. 1981-1984.

[CLE09] Clermidy, F.; Lemaire, R.; Popon, X.; Ktenas, D.; Thonnart, Y. “An Open and Reconfigurable Platform for 4G Telecommunication: Concepts and Application”. In: 12th Euromicro Conference on Digital System Design, Architectures, Methods and Tools (DSD’09), 2009, pp. 449 – 456.

[COE06] Coenen, M.; Murali, S.; Radulescu, A.; Goossens, K.; De Micheli, G. “A Buffer-Sizing Algorithm for Networks on Chip using TDMA and Credit-Based End-to-End Flow Control”. In: Proceedings of the 4th International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS’06), 2006, pp. 130-135.

[DAL01] Dally, W.; Towles, B. “Route Packets, Not Wires: On Chip Interconnection Networks”. In: Proceedings of the Annual Conference on Design Automation (DAC’01), 2001, pp. 684-689.

[DAL04] Dally, W. J.; Towles, B. “Principles and Practices of Interconnection Networks”. Elsevier, San Francisco. 2004, 550p.

[DAL06] Dally, W. “DRAFT Final Report: Workshop on On- and Off-Chip Networks for Multi-Core Systems”. Capturado em: http://www.ece.ucdavis.edu/~ocin06/OCIN_Workshop_Final_Report.pdf, 2006.

[DAN06] Daneshtalab, M.; Afzali-Kusha, A.; Mohammadi, S. “Minimizing Hot Spots in NoCs through a Dynamic Routing Algorithm based on Input and Output Selections”. In: International Symposium on System-on-Chip (SOC’06), 2006, pp. 49-52.

[DAV05] Davis, J.; Richardson, S.; Charitsis, C.; Olukotun, K. “A Chip Prototyping Substrate: the Flexible Architecture for Simulation and Testing (FAST)”. ACM SIGARCH Computer Architecture News, vol. 33-4, 2005, pp. 34-43.

[DIC08] Dick, R. “Embedded System Synthesis Benchmarks Suites (e3s)”. Capturado em: http://ziyang.eecs.northwestern.edu/~dickrp/e3s/, 2008.

[DUA03] Duato, J.; Yalamanchili, S.; Ni, L. “Interconnection Networks – An Engineering Approach”. Elsevier, San Francisco. 2003, 600p.

[DUM09] Dumitriu, V.; Khan, G. “Throughput-Oriented NoC Topology Generation and Analysis for High Performance SoCs”. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 17-10, Outubro 2009, pp. 1433-1446.

[FIT08] Fitzek, S.; Reisslein, M. “MPEG-4 and H.263 Video Traces for Network Performance Evaluation”. Capturado em: http://www-tkn.ee.tu-berlin.de/research/trace/trace.html, 2008.

[GEN05] Genko, N.; Atienza, D.; De Micheli, G.; Mendias, J.; Hermida, R.; Catthoor, F. “A Complete Network-On-Chip Emulation Framework”. In: Design Automation and Test in Europe (DATE’05), 2005, pp. 246-251.

[GHA09] Ghani, T. “Challenges and Innovations in Nano-CMOS Transistor Scaling”. Capturado em: http://download.intel.com/technology/silicon/Neikei_Presentation_2009_Tahir_Ghani.pdf, 2009.

[GLA94] Glass, C; Ni, L. “The Turn Model for Adaptive Routing”. Journal of the ACM, vol. 41-5, Setembro 1994, pp. 874-902.

[GOO02] Goossens, K.; van Meerbergen, J.; Peeters, A.; Wielage, P. “Networks on Silicon: Combining Best-Effort and Guaranteed Services”. In: Design Automation and Test in Europe (DATE’02), 2002, pp. 423-425.

[GOO05] Goossens, K.; Dielissen, J.; Radulescu, A. “Æthereal Network on Chip: Concepts, Architectures, and Implementations”. IEEE Design & Test of Computers, vol. 22-5, Set-Out 2005, pp. 414-421.

[GRE07] Grecu, C.; Ivanov, A.; Pande, R.; Jantsch, A.; Salminen, E.; Ogras, U.; Marculescu, R. “Towards Open Network-on-Chip Benchmarks”. In: 1st International Symposium on Networks-on-Chip (NOCS’07), 2007.

[GUE00] Guerrier, P.; Greiner, A. “A Generic Architecture for on Chip Packet-Switched Interconnections”. In: Proceedings of the Conference on Design, Automation and Test in Europe (DATE’00), 2000, pp.250-256.

[HEG05] Hegedûs, Á.; Maggio, G. M.; Kocarev, L. “A ns-2 Simulator Utilizing Chaotic Maps for Network-on-Chip Traffic Analysis”. In: IEEE International Symposium on Circuits and Systems (ISCAS’05), 2005, pp. 3375-3378.

[HU03] Hu, J.; Marculescu, R. “Exploiting the Routing Flexibility for Energy/Performance Aware Mapping of Regular NoC Architectures”. In: Design, Automation and Test in Europe (DATE’03), 2003, pp. 688-693.

[HU04] Hu, J.; Marculescu, R. “DyAD: Smart Routing for Network-on-Chip”. In: Proceedings of the Annual Conference on Design Automation (DAC’04), 2004, pp. 260-263.

[HU06] Hu, J.; Ogras, U.; Marculescu, R. “System-Level Buffer Allocation for Application-Specific Networks-on-Chip Router Design”. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 25-12, Dezembro 2006, pp. 2919-2933.

[IBM10] IBM Corporation. “The Cell project at IBM Research”. Capturado em: http://www.research.ibm.com/cell/, 2010.

[IEE10] “Moore’s Law Meets Its Match”. Capturado em: http://spectrum.ieee.org/computing/hardware/moores-law-meets-its-match, 2010.

[INT10] Intel Corporation. “World’s First 2-Billion Transistor Microprocessor”. Capturado em: http://www.intel.com/technology/architecture-silicon/2billion.htm?iid=tech_micro+2b, 2010.

[INT10a] Intel Corporation. “Intel’s Teraflops Research Chip”. Capturado em: http://download.intel.com/pressroom/kits/Teraflops/Teraflops_Research_Chip_Overview.pdf, 2010.

[ITR09] ITRS. “2009 International Technology Roadmap for Semiconductors–System Drivers”. Capturado em: http://www.itrs.net/Links/2009ITRS/2009Chapters_2009Tables/2009_SysDrivers.pdf, 2009.

[JER05] Jerraya, A.; Tenhunen, H.; Wolf, W. “Guest Editors’ Introduction: Multiprocessor Systems-on-Chips”. IEEE Computer, vol. 38-7, Julho 2005. pp. 36-40.

[JES89] Jesshope, C.; Miller, P.; Yantchev, J. “High Performance Communications in Processor Networks”. In: Proceedings of the 16th International Symposium on Computer Architecture (ISCA’89), 1989, pp. 150-157.

[KIM07] Kim, K.; Kim, D.; Lee, D.; Yoo, H. “Cost-efficient Network-on-Chip Design Using Traffic Monitoring System”. In: Design, Automation and Test in Europe Conference & Exhibition (DATE’07), 2007, 2p.

[LAN09] Lan, Y.; Lo, S.; Lin, Y.; Hu, Y.; Chen, S. “BiNoC: A Bidirectional NoC Architecture with Dynamic Self-reconfigurable Channel”. In: 3rd International Symposium on Networks-on-Chip (NOCS’09), 2009, pp. 266-275.

[LEE09] Lee, S.; Bagherzadeh, N. “A Variable Frequency Link for a Power-Aware Network-On-Chip (NoC)”. Integration, the VLSI Journal, vol. 42-4, Fevereiro 2009, pp. 479-485.

[LEL94] Leland, W. E.; Taqqu, M. S.; Willinger, W.; Wilson, D. V. “On the Self-similar Nature of Ethernet Traffic”. IEEE/ACM Transactions on Networking, vol. 2-1, Fevereiro 1994, pp. 1-15.

[LET10] LETI – Laboratoire d’Electronique et de Technologie de l’Information. Capturado em: http://www.leti.fr, Abril 2010.

[LI06] Li, M.; Zeng, Q.; Jone, W. “DyXY – A Proximity Congestion-Aware Deadlock-Free Dynamic Routing Method for Networks on Chip”. In: Design Automation Conference (DAC’06), 2006, pp. 849-852.

[LIN91] Linder, D.; Harden, J. “An Adaptive and Fault Tolerant Wormhole Routing Strategy for K-ary N-cubes”. IEEE Transaction on Computers, vol. 40-1, Janeiro 1991, pp. 2-12.

[LOF08] Lotfi-Kamran, P.; Daneshtalab, M.; Lucas, C.; Navabi, Z. “BARP-A Dynamic Routing Protocol for Balanced Distribution of Traffic in NoCs”. In: Design, Automation and Test in Europe Conference & Exhibition (DATE’08), 2008, pp. 1408-1413.

[LOG04] Loghi, M.; Angiolini, F.; Bertozzi, D.; Benini, L.; Zafalon, R. “Analyzing On-Chip Communication in a MPSoC Environment”. In: Design, Automation and Test in Europe Conference & Exhibition (DATE’04), 2004, pp. 752 – 757.

[MAN06] Manolache, S.; Eles, P.; Peng, Z. “Buffer Space Optimisation with Communication Synthesis and Traffic Shaping for NoCs”. In: Design, Automation and Test in Europe Conference & Exhibition (DATE’06), 2006, pp. 1-6.

[MAR01] Martin, G.; Chang, H. “System-on-Chip Design”. In: 4th International Conference on ASIC, 2001, pp.12-17.

[MAR05] Marcon, C.; Calazans, N.; Moraes, F.; Susin, A.; Reis, I.; Hessel, F. “Exploring NoC Mapping Strategies: An Energy and Timing Aware Technique”. In: Proceedings of the Conference on Design, Automation and Test in Europe (DATE’05), 2005, pp. 502-507.

[MAR05a] Marescaux, T.; Rangevall, A.; Nollet, V.; Bartic, A.; Corporaal, H. “Distributed Congestion Control for Packet Switched Networks on Chip”. In: Proceedings of the International Conference ParCo (ParCo’05), 2005, pp. 761-768.

[MAR10] Marczak, S. “Implementação de uma Infra-Estrutura de Monitoramento para Avaliação de Plataformas MPSoC Baseada em NoC”. Dissertação de Mestrado, PPGCC-FACIN-PUCRS. 2010. 63p. Disponível em: http://www.inf.pucrs.br/~moraes/my_pubs/papers/dissertacao_samuel.pdf.

[MAT08] “The Mathworks – MATLAB and Simulink for Technical Computing”. Capturado em: http://www.mathworks.com/, 2008.

[MEL06] Mello, A. “Qualidade de Serviço em Redes Intra-Chip – Implementação e Avaliação sobre a Rede Hermes”. Dissertação de Mestrado, PPGCC- FACIN-PUCRS. 2006. 138p. Disponível em: http://www.inf.pucrs.br/~moraes/my_pubs/papers/dissertacao_aline.pdf.

[MEL07] Mello, A.; Calazans, N.; Moraes, F. “Rate-based Scheduling Policy for QoS Flows in Networks on Chip”. In: IFIP International Conference on Very Large Scale Integration (VLSI-SoC’07), 2007, pp. 140-145.

[MIN04] Min, G.; Ould-Khaoua, M. “A Performance Model for Wormhole-Switched Interconnection Networks under Self-Similar Traffic”. IEEE Transactions on Computers, vol. 53-5, Maio 2004, pp. 601-613.

[MOO65] Moore, G. “Cramming More Components Onto Integrated Circuits”. Electronics, vol. 38-8, Abril 1965, pp.114-117.

[MOR04] Moraes, F.; Calazans, N.; Mello, A.; Moller, L.; Ost, L. “HERMES: an Infrastructure for Low Area Overhead Packet-switching Networks on Chip”. Integration, the VLSI Journal, Amsterdam, vol. 38-1, Outubro 2004, pp. 69-93.

[MOR10] Moreno, E. “Mapeamento e Adaptação de Rotas de Comunicação em Redes em Chip”. Tese de Doutorado, PPGCC-FACIN-PUCRS. 2010. 173p.

[MUR05] Murali, S.; Benini, L.; de Micheli, G. “Mapping and Physical Planning of Networks-on-Chip Architectures with Quality-of-Service Guarantees”. In: Asia and South Pacific Design Automation Conference (ASP-DAC’05), 2005, pp. 27-32.

[NIC06] Nicopoulos, C. A.; Dongkook, P.; Jongman, K.; Vijaykrishnan, N.; Yousif, M. S.; Das, C. R. “ViChar: a Dynamic Virtual Channel Regulator for Network-on-Chip Routers”. In: 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO’06), 2006, pp. 333-346.

[OGR06] Ogras, U.; Marculescu, R. “It's a Small World After All: NoC Performance Pptimization via Long-range Link Insertion”. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 14-7, Julho 2006, pp. 693-706.

[OGR08] Ogras, U. Y.; Marculescu, R. “Analysis and Optimization of Prediction-Based Flow Control in Networks-on-Chip”. ACM Transactions on Design Automation of Electronic Systems, vol. 13-1, Janeiro 2008, Article 11, 28p.

[PAN05] Pande, P.; Grecu, C.; Jones, M.; Ivanov, A.; Saleh, R. “Performance Evaluation and Design Trade-Offs for Network-on-Chip Interconnect Architectures”. IEEE Transactions on Computers, vol. 54-8, Agosto 2005, pp. 1025-1040.

[PAS08] Pasricha, S.; Dutt, N. “On-Chip Communication Architectures”. Morgan Kaufmann, Burlington. 2008, 544p.

[SAB07] Sabry, M.; El-Kharashi, M.; Bedor, H. “A New Dynamic Routing Algorithm for Networks-on-Chips”. In: IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PacRim’07), 2007, pp. 346-349.

[SAL05] Salminen, E.; Kangas, T.; Riihimaki, J.; Hamalainen, T. D. “Requirements for Network-on-Chip Benchmarking”. In: 23rd NORCHIP Conference, 2005, pp. 82-85.

[SAN05] Santi, S.; Lin, B.; Kocarev, L.; Maggio, G.; Rovatti, R.; Setti, G. “On the Impact of Traffic Statistics on Quality of Service for Networks on Chip”. In: IEEE International Symposium on Circuits and Systems (ISCAS’05), 2005, pp. 2349 - 2352.

[SCH07] Scherrer, A.; Fraboulet, A.; Risset, T. “Long-Range Dependence and On-chip Processor Traffic”. In: Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC’07), 2007, pp. 111-120.

[SET06] Sethuraman, B.; Vemuri, R. “optiMap: a Tool for Automated Generation of NoC Architectures using Multi-port Routers for FPGAs”. In: Design, Automation and Test in Europe (DATE’06), 2006, pp. 947-952.

[SOC08] “Soclib simulation environment”. Capturado em: http://soclib.lip6.fr/, 2008.

[SOT06] Soteriou, V.; Wang, H.; Peh, L. “A Statistical Traffic Model for On-Chip Interconnection Networks”. In: 14th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS’06), 2006, pp. 104-116.

[TED05] Tedesco, L. “Uma proposta para Geração de Tráfego e Avaliação de Desempenho para NoCs”. Dissertação de Mestrado, PPGCC-FACIN-PUCRS. Dezembro 2005. 126p. Disponível em http://www.inf.pucrs.br/~moraes/my_pubs/papers/dissertacao_leonel.pdf

[TED06] Tedesco, L.; Calazans, N.; Moraes, F. “Application Driven Traffic Modeling for NoCs”. In: Symposium on Integrated Circuits and Systems Design (SBCCI’06), 2006, pp. 62-67.

[TED08] Tedesco, L.; Calazans, N.; Moraes, F. “Buffer Sizing for Multimedia Flows in Packet-Switching NoCs”. Journal Integrated Circuits and Systems, vol. 3-1, Março 2008, pp. 46-56.

[TED09] Tedesco, L.; Clermidy, Fabien; Moraes, F. “A Path-Load Based Adaptive Routing Algorithm for Networks-on-Chip”. In: Symposium on Integrated Circuits and Systems Design (SBCCI’09), 2009, pp. 141-146.

[TED09a] Tedesco, L.; Clermidy, Fabien; Moraes, F. “A Monitoring and Adaptive Routing Mechanism for QoS Traffic on Mesh NoC architectures”. In: Proceedings of the 7th International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS’09), 2009, pp. 109-117.

[TIL10] Tilera Corporation. “TILE-Gx Processor Family”. Product Brief. Capturado em: http://www.tilera.com/pdf/Product_Brief_TILE-Gx.php, 2010.

[VAR04] Varatkar, G. V.; Marculescu, R. “On-Chip Traffic Modeling and Synthesis for MPEG-2 Video Applications”. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 12-1, Janeiro 2004, pp. 108-119.

[WOS07] Woszezenki, C. “Alocação de Tarefas e Comunicação entre Tarefas em MPSoCs”. Dissertação de Mestrado, PPGCC-FACIN-PUCRS. 2007. 121p. Disponível em: http://www.inf.pucrs.br/~moraes/my_pubs/papers/dissertacao_cris.pdf.

[XIL10] Xilinx Inc. “Xilinx Next Generation 28 nm FPGA Technology Overview”. Capturado em: http://www.xilinx.com/support/documentation/white_papers/wp312_Next_Gen_28_nm_Overview.pdf, Abril 2010.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo