PUCRS

ESCOLA POLITÉCNICA

PROGRAMA DE PÓS-GRADUAÇÃO

EM CIÊNCIA DA COMPUTAÇÃO

ANDERSON CAMARGO SANT'ANA

VULNERABILIDADES DE SEGURANÇA E CONTRAMEDIDAS EM PLATAFORMAS MPSOCS

Porto Alegre
2019

PÓS-GRADUAÇÃO - *STRICTO SENSU*

Pontifícia Universidade Católica
do Rio Grande do Sul

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

Vulnerabilidades de Segurança e Contramedidas em Plataformas MPSoC

Anderson Camargo Sant'Ana

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Ciência da Computação na Pontifícia Universidade Católica do Rio Grande do Sul.

Orientador: Fernando Gehm Moraes

Porto Alegre
2019

Ficha Catalográfica

S237v Sant'Ana, Anderson Camargo

Vulnerabilidades de segurança e contramedidas em plataformas MPSoC / Anderson Camargo Sant'Ana . – 2019.
82.
Dissertação (Mestrado) – Programa de Pós-Graduação em Ciência da Computação, PUCRS.

Orientador: Prof. Dr. Fernando Gehm Moraes.

1. Segurança. 2. Redes Dedicadas. 3. Firewalls. 4. Criptografia. 5. MPSoC. I. Moraes, Fernando Gehm. II. Título.

Elaborada pelo Sistema de Geração Automática de Ficha Catalográfica da PUCRS
com os dados fornecidos pelo(a) autor(a).
Bibliotecária responsável: Salete Maria Sartori CRB-10/1363

TERMO DE APRESENTAÇÃO

AGRADECIMENTOS

Gostaria de começar agradecendo a minha família pelo apoio e incentivo, vocês foram fundamentais em todas as etapas que eu passei para chegar aqui. Então, obrigado pai, mãe, irmão e irmã. Além destes, agradeço a minha namorada Jessica Santana pelo apoio e paciência.

Gostaria de agradecer também ao Professor Dr. Fernando Moraes pela orientação e dedicação durante o desenvolvimento desta Dissertação. Foi um ótimo orientador e professor, o levarei como um exemplo para minha vida acadêmica e profissional.

Agradeço aos colegas do GAPH pelo apoio técnico. Nominalmente, agradeço ao Marcelo Ruaro, por manter a HeMPS atualizada e dar o necessário suporte no desenvolvimento. Ainda, agradeço ao Guilherme Heck e Marcos Sartori por manterem a infraestrutura do laboratório sempre funcionando. Obrigado ao meu amigo Alzemiro pela sua amizade e esclarecimentos sobre plataforma. Finalmente, agradeço aos que colaboraram tecnicamente de forma direta para desenvolvimento do meu trabalho: Kevin Fiorentin, Bruno de Oliveira e Henrique Medina.

Agradeço ao Instituto Federal de Educação, Ciências e Tecnologia Sul-rio-grandense de Camaquã por me ajudar na realização do mestrado através das adequações de horário de trabalho, incentivo e amizade de seus colaboradores, e também ao Centro de Formação Profissional SENAI Ney Damasceno Ferreira pelo o incentivo a esta qualificação.

A presente Dissertação foi alcançado em cooperação com a Hewlett Packard Brasil LTDA e com recursos provenientes da Lei de Informática (Lei nº 8.248, de 1991).

VULNERABILIDADES DE SEGURANÇA E CONTRAMEDIDAS EM PLATAFORMAS MPSOC

RESUMO

Os sistemas computacionais tendem a adotar arquiteturas paralelas, utilizando sistemas multiprocessadores em chip (MPSoCs). Os MPSoCs são vulneráveis a ataques de software e hardware, como por exemplo, aplicações infectadas e Hardware Trojans podem ser instalados em um sistema MPSoC. Esses ataques podem ter o objetivo de obter acesso a dados sensíveis, interromper uma determinada aplicação ou até mesmo danificar o sistema fisicamente. A literatura apresenta técnicas de contramedidas: utilização de algoritmos de roteamento dedicados, criptografia, firewalls, filtros e zonas seguras. Essas abordagens apresentam um custo de hardware significativo (firewalls ou criptografia) ou são muito restritivas em relação ao uso de recursos MPSoC (zonas seguras). O objetivo desta Dissertação é implementar os ataques para expor as vulnerabilidades presentes em plataformas MPSoC e propor mecanismos de segurança leves para as vulnerabilidades encontradas. Aplicações instaladas em MPSoCs podem ser atacadas tanto no nível de hardware, como de software. Estes ataques comprometem a integridade dos dados processados pelo sistema. Dentre as vulnerabilidades estudadas destaca-se: (*i*) API de comunicação;(*ii*) sistema operacional; (*iii*) infraestrutura de rede; (*iv*) acesso a memória; (v) instalação de *hardware trojans*. Os mecanismos de segurança leves para MPSoCs apresentados neste trabalho utilizam quatro técnicas: isolamento espacial de aplicações; rede dedicada para enviar dados sensíveis; filtro de bloqueio de tráfego no firewall; criptografia. Esses mecanismos protegem o MPSoC contra os ataques de software mais comuns, como por exemplo, Negação de Serviço (DoS) e *man-in-the-middle*, e garante a confidencialidade e integridade das aplicações. Os resultados apresentam baixa sobrecarga de área e latência, os quais justificam-se pela segurança agregada ao MPSoC.

Palavras-chave: Segurança; Redes Dedicadas; Firewalls; Criptografia; MPSoC; Ataques.

SECURITY VULNERABILITIES AND COUNTERMEASURES ON MPSOC PLATFORMS

ABSTRACT

Computer systems tend to adopt parallel architectures, using Multi-Processor Systems-on-Chip (MPSoCs). MPSoCs are vulnerable to software and hardware attacks, such as infected applications and Hardware Trojans that can be installed on an MPSoC. The goal of these attacks can be the access sensitive data, disrupting a particular application, or even physically damaging the system. The literature presents several countermeasure techniques: the use of dedicated routing algorithms, encryption, firewalls, filters, and secure zones. These approaches have an important hardware cost (firewalls or encryption) or are very restrictive to the use of MPSoC resources (secure zones). The objective of this Dissertation is to implement the attacks to expose the vulnerabilities presented in MPSoC platforms and propose lightweight security mechanisms for the vulnerabilities. MPSoCs can be attacked both at the hardware and software levels. These attacks compromise the integrity of the data processed by the system. Among the vulnerabilities studied stands out: (*i*) communication API; (*ii*) operating system; (*iii*) network infrastructure; (*iv*) access to memory; (*v*) installation of Hardware Trojans. The lightweight security mechanisms for MPSoCs presented in this work adopt four techniques: spatial isolation of applications; dedicated network to send sensitive data; firewall to block malicious traffic using filters; cryptography. These mechanisms protect the MPSoC against the most common software attacks, such as Denial of Service (DoS) and man-in-the-middle, ensuring confidentiality and integrity to applications. Results show low area overload and latency, which are justified by the added security in MPSoCs.

Keywords: Security; Dedicated Networks; Firewalls; Cryptography; MPSoC; attacks.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ACRÔNIMOS

AES	Advanced Encryption System
API	Application Programming Interface
CI	Circuito Integrado
DoS	Denial of Service
DTA	Distributed Timing Attack
DTW	Dynamic time warping
E/S	Entrada/Saída
FN	False-Negative
FP	False-Positive
FW	Firewall
HeMPS	HERMES Multiprocessor System
HT	Hardware Trojan
IoT	Internet-of-Things
IP	Intellectual Property
MITM	Man-In-The-Middle
MP	Manager Processor
MPEG	Moving Picture Experts Group
MPSoC	Multiprocessor System-on-Chip
NI	Network Interface
NoC	Network-on-Chip
OS	Operating System
PE	Processing Element
PUFs	Physical Unclonable Functions
RTL	Register Transfer Level
SCA	Side-Channel Attack
SEA	Secured Environment Architecture
SHA	Secure Hash Algorithm
SoC	System-on-Chip
SP	Slave Processor
SU	Security Unit
YAML	YAML Ain’t Markup Language

SUMÁRIO

1 INTRODUÇÃO

Sistemas computacionais tendem a utilizar arquiteturas paralelas com sistemas multiprocessados em chip (MPSoCs) [WAS13]. Estes sistemas são compostos por uma grande quantidade de elementos de processamento (PEs) e elementos de armazenamento, os quais podem ser interconectados por uma rede intrachip de comunicação denominada NoC (do inglês, *Network-on-Chip*). Essa tendência na indústria de semicondutores para construção de MPSoCs é aplicada essencialmente para aplicações que requerem uma grande carga de processamento ou uma grande interconectividade de dispositivos. Exemplos de aplicações que demonstram a aplicabilidade de arquiteturas MPSoCs incluem:

- Processamento de sinais: codificação e compressão de vídeo em alta definição como MPEG4 (H264) ou vídeo 4K, transmissão digital de sinais, WiMAX, TV digital, telefonia 4G/5G, *wifi*, *bluetooth*, GSM, *software-defined radio* [BIS15];
- Processamento de imagem: renderização de imagem em tempo real, realidade aumentada, aplicações médicas como tomografia computadorizada [WAS13];
- Reconhecimento de padrões: aplicações em biometria como reconhecimento de face, voz, impressão digital, encontrar semelhanças e relações entre textos em linguagem natural, sequenciamento de DNA [LEE11];
- Internet das coisas (IoT) e plataformas móveis [GRE11].

Aplicações executadas em MPSoCs requerem técnicas de segurança para prevenção a possíveis ataques gerados por invasores. Estes ataques comprometem a integridade dos dados processados pelo sistema. Os invasores podem ter diferentes objetivos, desde a extração de informações do sistema até mesmo a desconfiguração ou danificação física do sistema.

Segundo Sepúlveda et al. [SEP14], os ataques baseados em software somam cerca de 80% dos ataques realizados em sistemas embarcados. Ataques de software podem ser oriundos de aplicações ou do sistema operacional infectados. Exemplos de ataques realizados a partir de software incluem, de forma não exaustiva: vírus, cavalos de Tróia, *worms*, negação de serviço, extração de informações sensíveis, *hijacking* e *spyware*.

No nível de hardware também ocorrem ataques. Exemplo de ataque muito relatado na literatura são os realizados por *hardware trojans* – HTs [TEH10] [KOC11]. HTs são módulos adicionados em alguma etapa do projeto, dado que a indústria de semicondutores desenvolve circuitos integrados (CIs) utilizando diversas empresas distribuídas em diferentes locais, da *design house* ao integrador do CI. Uma vez o HT inserido no CI, o mesmo pode ser ativado para obtenção de dados, degradação do desempenho ou a danificação física do CI [ALK08].

Os ataques relacionados acima, tanto de software quanto de hardware, podem ter a finalidade de encontrar uma chave de criptografia utilizada no MPSoC para obter o acesso a dados sensíveis. O acesso à informação criptografada é possível, pois os dados criptografados trafegam na infraestrutura de comunicação ou encontra-se armazenados na memória. Uma técnica para obtenção de chaves é o ataque de canal lateral (SCA, do inglês *side channel attack*) [KAR01].

Devido aos diversos tipos de ataques reportados na literatura, ter um sistema resiliente a ataques torna-se cada vez mais necessário para a indústria de sistemas eletrônicos. Por consequência, surgiram diversas técnicas que visam a aumentar o nível de segurança dos CIs. Dentre as técnicas utilizadas em nível de hardware destacam-se: zonas seguras [CAI17], criptografia [ISA13] e *firewalls* [FER15].

A criação de zonas seguras é uma técnica que protege as aplicações através da reserva de recursos de comunicação e/ou computação. No nível de computação, evita-se que os processadores executem tarefas de aplicações distintas, evitando-se que aplicações maliciosas acessem a memória local para extração de informação. No nível de comunicação adotam-se duas técnicas principais. A primeira é a criação de zonas seguras disjuntas, onde a comunicação é protegida por criptografia [SEP17][FER16]. A segunda técnica é a criação de uma zona segura contínua (de formato retangular), com bloqueio de tráfego externo [CAI17]. Esta segunda opção é melhor, em termos de segurança, pois não requer criptografia e os fluxos não são perturbados por fluxos externos. Assim, a criação de zonas seguras, com bloqueio de tráfego de dados, torna o sistema imune aos ataques que visam explorar vulnerabilidades no nível da comunicação. A desvantagem desta segunda opção reside no fato de se criar regiões onde não é possível acessar os recursos do MPSoC (chamadas pela literatura de regiões opacas [CAI17]), o que pode comprometer a execução das demais aplicações.

A maioria das propostas encontradas na literatura visando segurança em MPSoCs utilizam criptografia. A criptografia provê integridade e confiabilidade para informações sensíveis, ou seja, informações que precisam ser processadas de modo seguro, assim como sua autenticação. Assim, através da criptografia cria-se canais seguros de comunicação [ISA13]. Observar que estes dados criptografados compartilham o meio de comunicação com outros fluxos de dados. Estes fluxos de dados podem ser maliciosos, buscando acesso aos dados através de ataques SCA.

Uma outra forma de agregar segurança em MPSoC é a utilização de *firewalls* para filtrar os dados de entrada e saída de acordo com a política de segurança implementada [FER15]. Esses *firewalls* são alocados em lugares estratégicos, para proteger os PEs de diversas formas de ataques.

1.1 Motivação

Ao analisar os trabalhos desenvolvidos na área de segurança para MPSoCs, há diversas propostas de mecanismos para prover um maior grau de segurança ao sistema. Entretanto, poucos trabalhos avaliam de forma conjunta, a eficiência e descrição dos ataques propostos e a avaliação das contramedidas propostas.

A motivação do presente trabalho baseia-se na necessidade de realizar propostas de mecanismos de segurança próximos à realidade de MPSoCs. Para isto, deve-se executar o seguinte conjunto de ações: (*i*) avaliação de vulnerabilidades; (*ii*) realização de ataques para comprovação destas vulnerabilidades; (*iii*) proposta de contramedidas.

1.2 Objetivos

O objetivo estratégico desse trabalho compreende a avaliação de vulnerabilidades em MPSoCs, tanto no nível de software quando no nível de hardware, propondo contramedidas para evitar diferentes tipos ataques.

Os objetivos específicos desse trabalho também incluem:

- Domínio da plataforma de referência, a qual inclui *firewalls* e criptografia AES (*Advanced Encryption Standard*) [OLI18a];
- Domínio de técnicas de ataques em MPSoCs;
- Domínio de técnicas de segurança em MPSoCs;
- Avaliar vulnerabilidade no nível de aplicação, *kernel* (sistema operacional) e físico (processadores e NoC);
- Executar ataques de DoS, spoofing, Hardware Trojan e acesso à memória a plataforma de referência;
- Propor contramedidas para eliminar ou mitigar os ataques;
- Avaliar as contramedidas propostas.

1.3 Metodologia

A presente Dissertação tem por base de desenvolvimento a plataforma MPSoC desenvolvida por Bruno Oliveira [OLI18a], modelada em linguagem VHDL, a qual inclui *firewalls* e criptografia AES dos dados que trafegam na NoC. Esta plataforma é modificada ao longo do trabalho, em função dos ataques realizados e das contramedidas propostas.

1.4 Estrutura do Documento e Contribuições

A Figura 1 ilustra a interdependência entre os Capítulos desta Dissertação. O Capítulo 2 contém a fundamentação teórica necessária para a compreensão dos conceitos

relacionados à segurança e ataques no contexto deste trabalho. Utilizando estes conceitos, apresenta-se no Capítulo 3, o estado-da-arte relacionado à segurança em sistemas MPSoCs, posicionando qualitativamente a presente Dissertação em relação aos trabalhos avaliados. Na sequência, Capítulo 4, apresenta-se o MPSoC de referência. Em função dos tipos de ataques e vulnerabilidades apresentados no Capítulo 2, mostra-se no Capítulo 5 que estes são factíveis de serem realizados em MPSoCs, através da simulação dos mesmos na plataforma de referência. Para mitigar os ataques realizados, apresenta-se no Capítulo 6 a arquitetura SEA (*Secured Environment Architecture*) [OLI18a], a qual é baseada na arquitetura de referência, tendo como contramedidas mecanismos de segurança relacionados às vulnerabilidades exploradas no Capítulo anterior. O Capítulo 7 apresenta os resultados quantitativos da plataforma SEA. O Capítulo 8 apresenta as conclusões desta Dissertação, publicações e trabalhos futuros.

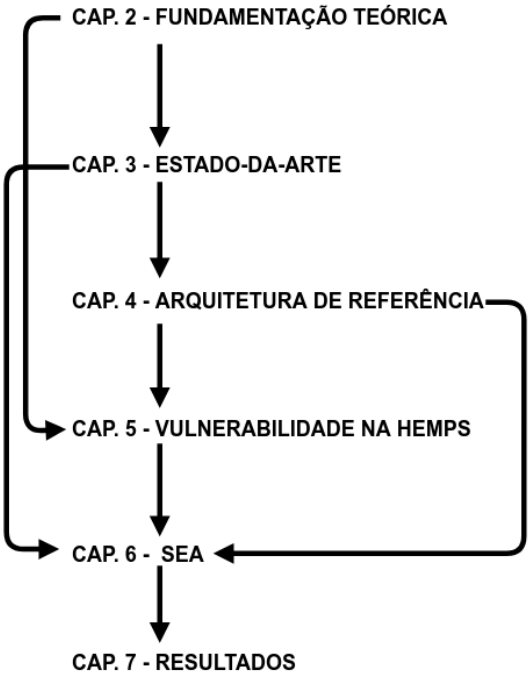


Figura 1: Estrutura dos Capítulos da presente Dissertação (fonte: Autor).

Destacam-se abaixo a principais ***contribuições*** do autor no presente trabalho:

- Implementação do ataque *man-in-the-middle* na plataforma de referência;
- Ataques de DoS;
- Acesso indevido à memória;
- Criação de políticas de segurança;
- Implementação de rede dedicada (HNoC) na plataforma SEA;
- Protocolo de comunicação da rede com os firewalls;
- Distribuição das chaves utilizando a HnoC;
- Construção do filtro de segurança inserido no firewall.

2 FUNDAMENTAÇÃO TEÓRICA

Este Capítulo tem por objetivo expor os conceitos que serão mencionados ao longo da Dissertação. A Seção 2.1 apresenta os princípios de segurança desejáveis em um sistema seguro. A Seção 2.2 elenca os tipos de ataques realizados em MPSoCs. A seção 2.3 correlaciona os princípios de segurança como os tipos de ataques, e quais são tratados na presente Dissertação.

2.1 Definições de Segurança

De acordo com Ramachandran em seu livro "*Designing Security Architecture Solutions*" [RAM02], há sete princípios de segurança que são geralmente aceitos como a base de uma boa solução de segurança em um sistema. Estes princípios de segurança com suas respectivas definições são descritos a seguir:

- *Autenticação*: é o processo de estabelecer a validade de uma identidade reivindicada. O responsável por gerar um pedido de acesso para um dispositivo protegido, uma sessão ou transação, deve apresentar suas credenciais que comprovem sua identidade.
- *Autorização*: é o processo de determinar se uma entidade válida pode ter o acesso a um recurso seguro baseado em atributos, predicados ou contexto.
- *Integridade*: é a prevenção da modificação ou destruição de um recurso/informação por um usuário ou entidade não autorizada.
- *Disponibilidade*: é a proteção de recursos/informações contra ameaças de negação de serviço (do inglês, *Denial of Service (DoS)*) que podem impactar na disponibilidade do sistema.
- *Confiabilidade*: é a propriedade de não divulgação de informações para usuários, entidades ou processos não autorizados.
- *Auditoria*: é a propriedade de registrar todas as atividades do sistema em níveis suficientes para reconstrução de eventos.
- *Não-repúdio*: A impedimento de qualquer participante em uma comunicação ou transação, negando sua respectiva função na interação, uma vez que ela esteja completa.

Além desses sete principais princípios de segurança, os desenvolvedores de sistemas devem fornecer apoio a outros princípios que melhorem a facilidade de uso, manutenibilidade, facilidade de manutenção, e administração de segurança do sistema.

Para garantir algumas premissas destacadas acima, os desenvolvedores de sistemas criam políticas de segurança para agregar segurança ao sistema. É importante ressaltar que não é possível classificar sistemas como seguros e não seguros, pois o termo

segurança não é absoluto. Na literatura de segurança em sistemas embarcados, podemos destacar algumas técnicas para agregar segurança as quais são: roteamento adaptativo, zonas seguras, módulos em hardware para agregar segurança, rede de comunicação dedicada à segurança, firewalls e monitores de tráfego:

- *Roteamento adaptativo*: esta técnica altera o caminho de roteamento quando uma situação anômala [SEP15] é detectada, garantindo a *disponibilidade* do sistema;
- *Zonas seguras*: esta técnica visa a criação de zonas isoladas [CAI17][FER16] que protegem a comunicação e computação de dados sensíveis, garantindo a *integridade* das aplicações;
- *Módulos em hardware*: esta técnica visa a criação de hardwares específicos para agregar segurança a MPSoCs [PRA17][CHA18]. Estes módulos visam inibir ataques no nível de hardware e software. Cada módulo é projetado de acordo com o ataque a ser inibido, como por exemplo, os módulos de criptografia;
- *Rede de comunicação dedicada à segurança*: esta técnica visa a construção de uma arquitetura de rede para inibir uma variedade de ataques [REI16][KIN17], tais como ataques de canal lateral (SCA – do inglês *side channel attack)*.
- *Firewalls*: esta técnica de segurança é utilizada para controlar as conexões de entrada e saída dos elementos de processamento (PE) de acordo com a política de segurança implementada [FER15].

2.2 Tipos de Ataques

De acordo com Avizienis et al. [AVI04] os invasores (do inglês, *intruders*) de um sistema, podem serem definidos como entidades maliciosas (seres humanos ou outros sistemas) que tentam exceder (superar) qualquer autoridade, com o intuito de alterar ou interromper um serviço do sistema. Os invasores podem alterar a funcionalidade ou desempenho do sistema visando acessar informações sensíveis do mesmo.

Os invasores podem realizar ataques visando diferentes componentes do sistema. Por exemplo, o processador, a memória, a infraestrutura de comunicação (barramento ou NoC). Há diferentes tipos de ataques reportados na literatura que visam a aquisição de dados, a degradação do desempenho de uma aplicação, ou até mesmo provocar danos físico permanente ao dispositivo.

Ataques de negação de serviço (DoS) são projetados para que usuários de um sistema não acessem ou utilizem o sistema de maneira satisfatória. Os ataques DoS geralmente interrompem o serviço de uma rede ou de um núcleo de processamento, de modo que prejudique a utilização de recursos do sistema, pois o desempenho é seriamente afetado [HAN05]. De acordo com Hansman et al. [HAN05], o ataque DoS pode ser classificado em 3 tipos: ataques baseados em *host*, rede e ataques distribuídos.

- Baseados em *host*: ataque DoS baseados em *host* são projetados para usar recursos em um computador. Recursos como tempo de CPU e uso de memória são os alvos mais comuns.
- Baseados em rede: este tipo de ataque visa atacar os recursos de uma rede de comunicação. Este ataque utiliza pacotes de comunicação para inundar a rede de dados na tentativa de interromper o seu funcionamento. Para o sucesso deste ataque, os pacotes enviados pelo intruso devem ter uma quantidade maior de pacotes que a rede pode manipular. Se o intruso está atacando a rede, os pacotes enviados devem ser suficientes para que a largura de banda deixada para usuários legítimos seja severamente reduzida.
- Distribuídos: Os ataques DDoS (*distributed DoS*) funcionam usando um grande número de *hosts* para direcionar um ataque simultâneo a um ou diversos alvos. Um número de nodos mestres é usado para controlar um número maior de nodos *daemos*[1], os quais lançam o ataque ao alvo.

A Figura 2 mostra o processo de um ataque DDoS. No primeiro instante, o intruso envia comandos aos nodos principais (*master nodes*) para iniciar o ataque. Os nodos principais, em seguida, ordenam que todos os nodos *daemon* sob seu comando iniciem o ataque. Finalmente, os nodos *daemon* atacam o alvo simultaneamente, causando uma negação de serviço (DoS).

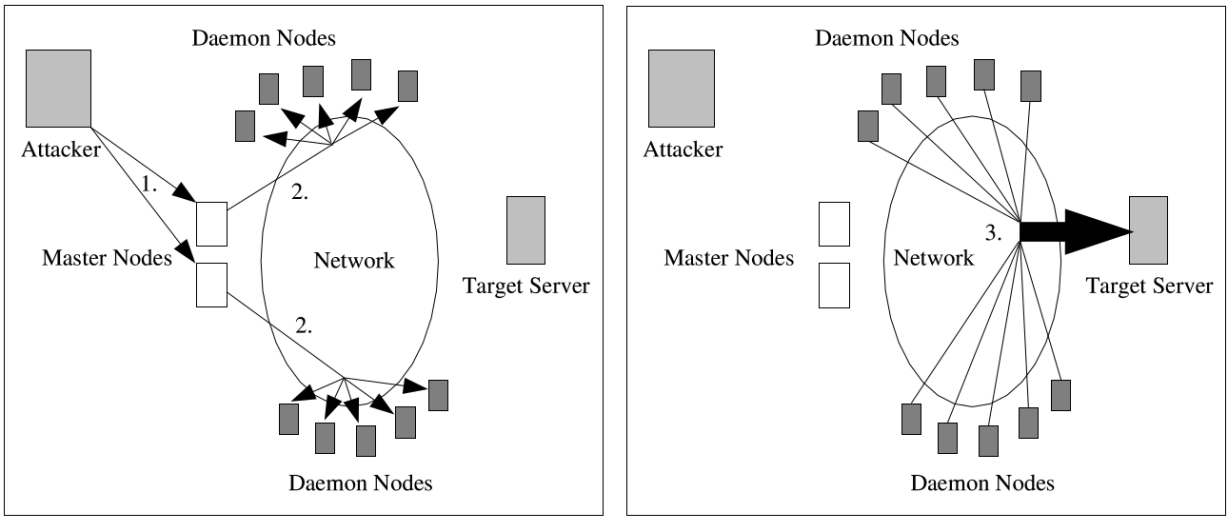


Figura 2: Mecanismo de execução de ataque DDoS [HAN05].

Agosta [AGO16] et al. apresentam uma variedade de ataques denominados ataques de canal lateral (do inglês, *Side-Channel Attacks (*SCA*)*). Estes ataques se caracterizam em obter informações confidenciais de forma indireta. Técnicas de ataques SCA incluem, por exemplo, avaliação da assinatura da energia consumida no sistema, a análise do tempo

[1] Em sistemas operacionais multitarefa, um daemon é um programa de computador que executa como um processo em plano de fundo, fonte: wikipédia

de execução de determinada aplicação, e a avaliação da irradiação eletromagnética emitida pelo sistema [AGO16].

Ataques baseados em tempo (do inglês, *Timing Attacks* (TA)) são ataques do tipo SCA e se caracterizam por injetar tráfego para colidir com o tráfego de informações sensíveis e obter informações, como por exemplo, a chave de criptografia utilizada para criptografar mensagens de um fluxo de comunicação [REI16]. Pode-se implementar uma versão de ataque distribuída, conhecido como DTA (do inglês, *Distributed Timing Attack* (DTA)). Este ataque ocorre quando múltiplos núcleos de processamento são infectados, podendo trabalhar coordenadamente para implementar o DTA.

Outro ataque relatado no estado da arte é *Time-Driven-Attack* [SEP15]. Os ataques *time driven* são gerados por processos maliciosos instalados no sistema. Estes processos podem inferir o padrão das informações sensíveis através da degradação do desempenho, pela inserção de tráfego através do processo malicioso. Isso ocorre devido a colisão dos tráfegos do invasor e da informação sensível. A Figura 3 mostra um cenário de ataque, onde S é o processo que envia informações sensíveis, D é o processo de destino, e A é o processo malicioso injetando dados para realizar o ataque *time-driven*.

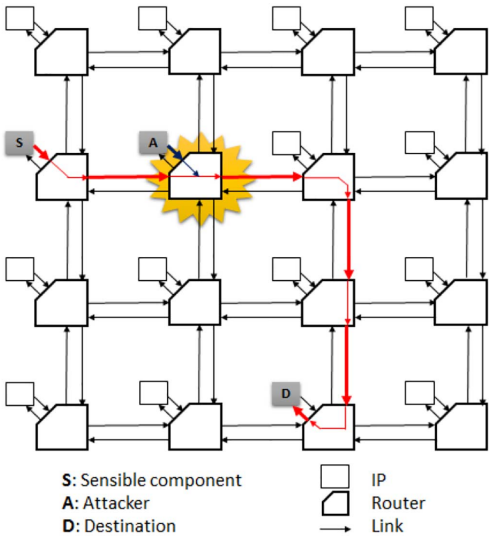


Figura 3: Exemplo de um cenário de ataque baseado na técnica *time-driven* [SEP15].

De acordo com Conti et al. [CON16] o ataque *man-in-the-middle* (MITM) é um dos ataques mais conhecidos em segurança de computadores, representando uma das maiores preocupações dos profissionais de segurança. O MITM visa obter dados que trafegam entre duas vítimas (computadores, PEs), comprometendo a confidencialidade e integridade dos dados. No ataque MITM, o cenário mais comum envolve: dois nodos (vítimas) e um intruso (representado como *attacker*). O intruso tem acesso ao canal de comunicação entre os dois nodos e pode manipular suas mensagens. O ataque MITM pode ser visualizado como mostrado na Figura 4. Como exemplo do ataque, as vítimas tentam iniciar uma comunicação segura enviando chaves públicas entre si (mensagens M1 e M2). O atacante intercepta M1 e M2, e como retorno envia a sua chave pública às vítimas

(mensagens M3 e M4). Na sequência, a vítima 1 (do inglês, *victim 1*) encripta a sua mensagem através da chave pública do atacante e a envia à vítima 2 (mensagem M5). O atacante intercepta a M5, e a descriptografa, utilizando uma chave privada conhecida. Em seguida, o atacante criptografa o texto simples pela chave pública da vítima 2 e o envia para a vítima 2 (mensagem M6).

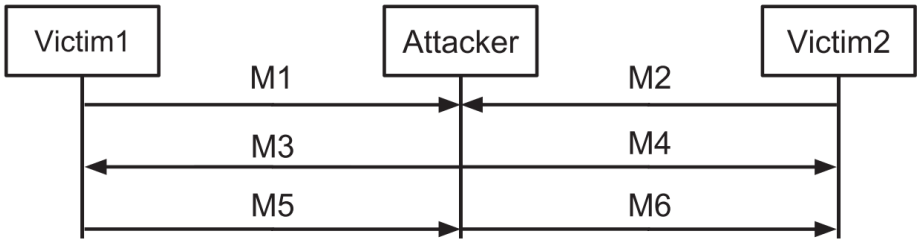


Figura 4:Troca de mensagens em um típico ataque MITM [CON16].

Como resultado, o intruso convenceu ambas as vítimas de que elas usam um canal seguro, mas na realidade ele tem acesso a todas as mensagens criptografadas.

2.3 Correlação de propriedades de Segurança e Ataques

A Tabela 1 apresenta uma relação entre os tipos de ataques relatados na Dissertação e os princípios de segurança que serão afetados pelos ataques.

Tabela 1: Relação de ataques e segurança em MPSoCs.

Tipos de Ataques	Princípios de Segurança						
	Auten.	Autoriz.	Integri.	Dispon.	Conf.	Audit.	Não-rep.
DoS				M			
Man-in-the-middle	T	T	T		T		T
Acesso a memória		T	T				
SCA					T		

Entre a relação do tipo de ataque e o princípio de segurança, são utilizadas marcações para mostrar que o presente trabalho trata o ataque com a seguinte legenda:

- Mitigado (M) – Significa que reduz a ocorrência do ataque, mas não o elimina.
- Tratado (T) - Significa que foram eliminadas todas as ocorrências de ataques encontradas.

Durante o trabalho foram abordadas técnicas para garantir as propriedades de segurança abordadas neste Capítulo. Para garantir a propriedade de autenticação utilizaremos uma identificação, denominada *App_ID, p*ara cada aplicação que é alocada no sistema, onde verificaremos esta identificação nos pacotes que entrarem e saírem dos PEs.

Na propriedade de autorização, cada tarefa tem autorização de se comunicar apenas pelas tarefas que compõe sua aplicação. São descartados os pacotes oriundos de tarefas que tentarem se comunicar com tarefas de aplicações diferentes.

Na comunicação entre tarefas, os pacotes possuem um *flit* denominado *App_ID*. Este *flit* é inserido pela interface de rede (mecanismo este detalhado posteriormente), garantindo assim a integridade do pacote. Nesse sentido, consideramos como elementos seguros o PE que gerencia o sistema, e a rede dedicada que configura as interfaces de rede.

A propriedade de disponibilidade é afetada por ataque do tipo DoS. Nesse sentido, o sistema evita o DoS através de um filtro, o qual evita a inundação de pacotes corrompidos. A confiabilidade [RAM02] é garantida pela política de segurança que restringe que o fluxo de comunicação existe apenas entre tarefas de uma mesma aplicação.

O presente Dissertação não abordará e não tratará as propriedades de segurança relacionadas a auditoria e não-repúdio.

3 ESTADO-DA-ARTE

Este Capítulo apresenta propostas relacionadas à segurança em NoCs e em MPSoCs, assim como a descrição de ataques relacionados a estes sistemas.

3.1 Noc-Based Protection for SoC Time-Driven Attacks

O objetivo do trabalho em [SEP15] é modificar o árbitro e a lógica de roteamento dos roteadores presentes em NoCs, com o objetivo de evitar ataques de *temporização* (do inglês, *time driven attacks*) por invasores instalados no MPSoC.

Como solução, são propostos dois mecanismos: arbitragem randômica e roteamento adaptativo. A arbitragem randômica possui dois modos de operação. O primeiro modo ocorre na presença de contenção, onde um número sorteado aleatoriamente define o fluxo que será atendido. O segundo modo de operação acontece quando não há contenção. Neste caso é determinado que o pacote será atendido após um atraso definido por um gerador pseudoaleatório. Estes dois modos garantem que não haja nenhuma relação do desempenho do tráfego do invasor com o tráfego da rede, assim impossibilitando a extração de informações a partir da análise do comportamento da rede.

No roteamento adaptativo o objetivo é evitar que dois tráfegos se cruzem, evitando assim que o invasor extraia informações a partir da degradação do desempenho de seu tráfego. Um exemplo é o roteamento *West-First*.

A Figura 5 mostra a vazão para diferentes taxas de injeção de dados provenientes do processo malicioso instalado no sistema para diferentes tipos de técnicas aplicadas ao roteamento. Nota-se que a vazão utilizando as técnicas propostas, roteamento adaptativo e arbitragem randômica, mantiveram-se constantes, ou seja, inibiram a interação do tráfego do invasor com o tráfego de informação sensível. Assim, o invasor não poderá tirar conclusões através do monitoramento, informações estas relacionadas ao tráfego sensível, pois a vazão do processo malicioso não foi afetada.

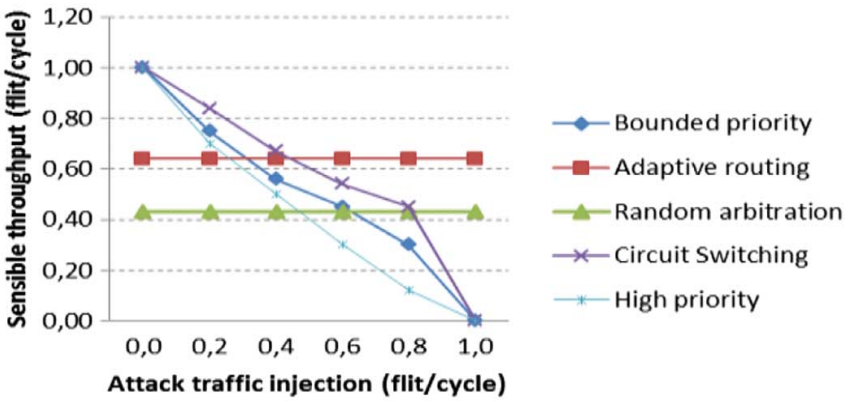


Figura 5: Valores de desempenho para o tráfego sensível sobre diferentes taxas de injeção do tráfego do invasor [SEP15].

3.2 Activation of Secure Zones in Many-core Systems with Dynamic Rerouting

Caimi et al. [CAI17] propõe proteção simultânea da comunicação e da computação contra ataques em sistemas *many-core*, através da definição de regiões seguras opacas (OSZ, do inglês *Opaque Secure Zones*) em tempo de execução. O isolamento das OSZs é realizado por *wrappers*, assim reduzindo o custo de área e latência comparado aos outros sistemas de segurança intrachip, como por exemplo o uso de *firewalls* e módulos de criptografia.

A arquitetura do sistema proposto, mostrada na Figura 6, contém duas redes de comunicação: a rede de dados e a rede de controle. A rede de dados é responsável por somente transportar mensagens de dados das aplicações, e a rede de controle transporta mensagens de controle. As duas redes possuem *wrappers*, que atuam para o isolamento da OSZ.

Os *wrapper* bloqueiam a entrada e saída de dados de ambas redes. A técnica proposta visa mitigar as seguintes vulnerabilidades: negação de serviço, ataque baseado em tempo (do inglês, *timing attack)*, mascaramento da identidade utilizando endereços de remetentes falsificados (do inglês, *spoofing)*, e também o comprometimento da confiabilidade e da integridade dos dados. Os autores assumem que nenhum recurso é compartilhado dentro da OSZ e que esta região compreende uma forma retangular. Desta forma, a região segura torna-se imune às vulnerabilidades mencionadas.

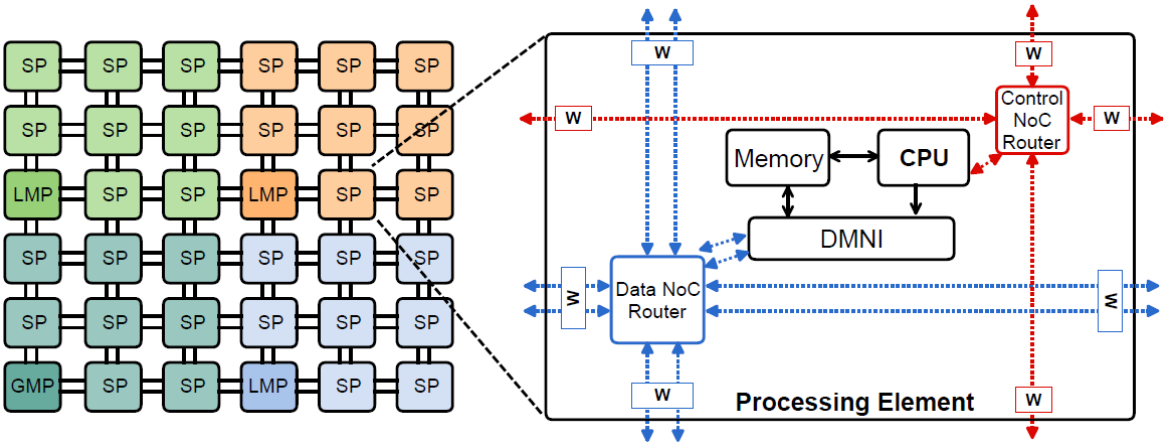


Figura 6: Arquitetura do sistema. *Wrappers* (W) são adicionados para o controle dos sinais que conectam as NoCs, habilitando o isolamento das portas individuais [CAI17].

Ao implementar o protocolo de OSZ, os resultados mostraram um tempo de 1,2 µs para iniciar uma OSZ em um cenário com 25 PEs. Esta baixa latência é o custo para se inserir aplicações com requisitos de segurança na plataforma. Outra análise realizada diz respeito ao impacto das OSZs sobre o desempenho de aplicações que não requerem segurança em sua execução. Na avaliação realizada, as aplicações tiveram um aumento no tempo de execução de 2,56% para o pior caso. Este aumento de tempo é devido a

reconfiguração dos caminhos de comunicação das tarefas para evitar que atravessam a região segura. As Figura 7(a) e a Figura 7(b) mostram, respectivamente, as aplicações MPEG e DTW, sendo executadas fora da zona segura. O tráfego de comunicação dessas tarefas não atravessa a zona segura devido ao processo de re-roteamento [WAC17].

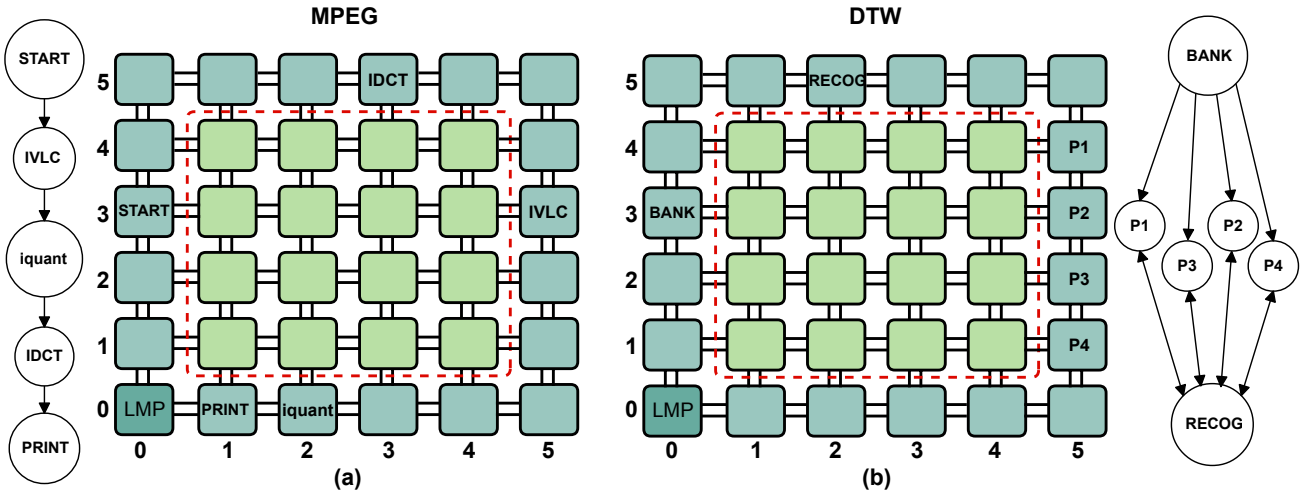


Figura 7: Grafo de tarefas e mapeamento. Aplicações: (a) MPEG e (b) DTW [CAI17].

3.3 A Security Aware Routing Approach for NoC-based MPSoCs

Fernandes et al. [FER16] apresentam um algoritmo de roteamento para NoCs objetivando implementar zonas seguras. Diferentemente da técnica apresentada anteriormente, OSZs, esta técnica é realizada em tempo de projeto e não durante a execução das aplicações. Esta técnica visa evitar dois tipos de ataques: ataques baseados em tempo (do inglês, *timing attacks*) e negação de serviço (DoS, do inglês *Denial of Service*). Ataques baseados em tempo se caracterizam por injetar tráfego para colidir com o tráfego de informações sensíveis. Ataques do tipo DoS são caracterizados por degradar o desempenho do sistema através da sobrecarga na arquitetura de comunicação. Ambos os ataques interferem na NoC usando o mesmo princípio (colisões e interferência de tráfegos sensíveis). Zonas seguras são apresentadas como alternativa para isolação de tráfegos e computação sensíveis, com intuito de proteger o sistema.

Conforme a Figura 8, o trabalho considera três cenários diferentes de comunicação:

i. Comunicação total dentro da zona (FIZ): a origem e o destino estão na mesma zona e o caminho de roteamento também está dentro da zona, como por exemplo, IP1 e IP2.

ii. Comunicação parcial dentro da zona (PIZ): a origem e o destino estão na mesma zona, porém, o caminho de roteamento não está contido numa única zona, como por exemplo, IP3 e IP4.

iii. Comunicação entre zonas: a origem e o destino estão alocados em diferentes zonas, como por exemplo, IP5 e IP6.

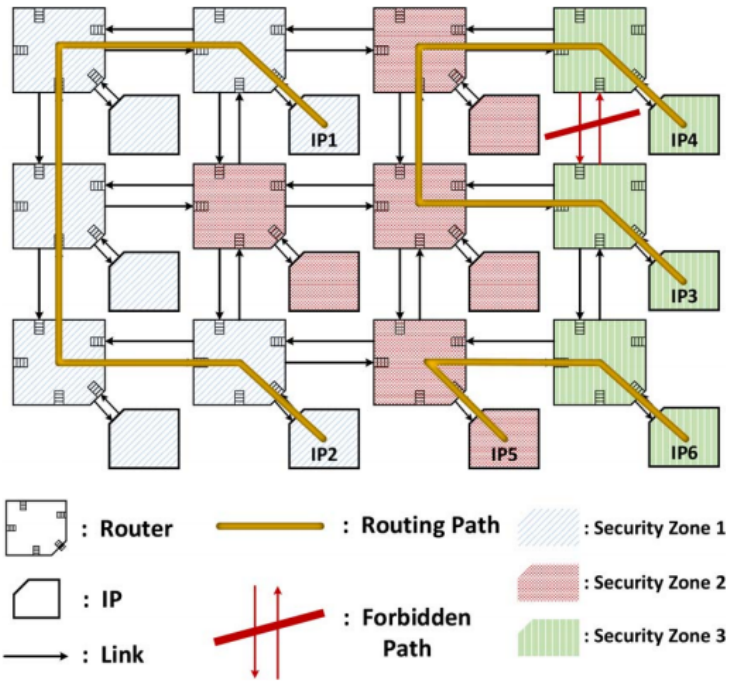


Figura 8: Cenário de comunicação com zonas seguras [FER16].

O trabalho adota dois algoritmos de roteamento: o roteamento baseado em segmentos (SBR, do inglês *Segment-based Routing*) e o roteamento baseado em regiões (RBR, do inglês *Region-based Routing*). O SBR é responsável por evitar *deadlock*s, enquanto o RBR calcula os caminhos.

O algoritmo SBR é composto de duas fases: computação de segmentos e definição de restrições de roteamento. Na primeira fase o SBR divide a NoC em segmentos (de fato os segmentos correspondem a regiões), incluindo roteadores e enlaces. O SBR busca minimizar o número de elementos por segmento. Na sequência, o SBR define as restrições de trocas de direção (*turns*).

O algoritmo RBR considera as restrições de *turns*, calculadas pelo SBR, para encontrar caminhos entre todas as origens e destinos no NoC. Como resultado, define-se uma tabela de roteamento para cada roteador. A principal vantagem do RBR é que uma única entrada na tabela de roteamento pode representar um caminho para mais de um destino, o que pode reduzir significativamente o tamanho destas tabelas.

A proposta do trabalho é o algoritmo SBR *Security Zone Awareness* (SBR- SZA), uma heurística que considera as características dos requisitos de segurança das aplicações. Este método, SBR- SZA, utiliza dois conceitos: zonas seguras em conjunto com o algoritmo de roteamento. O SBR-SZA usa o algoritmo SBR visando calcular rotas privilegiando zonas do tipo FIZ, ou seja, diminuindo as conexões entre zonas.

O conjunto de benchmarks NASA *Numeral Arodynamic Simulation* (NAS) foi utilizado para mapear 5 aplicações em uma NoC 13x13. Cada aplicação contém 32 tarefas. Dado que pacotes que atravessam zonas seguras requerem criptografia, o experimento avaliou

o impacto que a técnica de criptografia pode induzir a latência, conforme mostra a Figura 9. Métodos de criptografia leve, como HIGH induzem uma penalidade inferior a 1% na latência, enquanto o AES-128 pode dobrar a latência. Esta Figura também mostra que o SBR-SZA aumenta a latência em relação ao SBR.

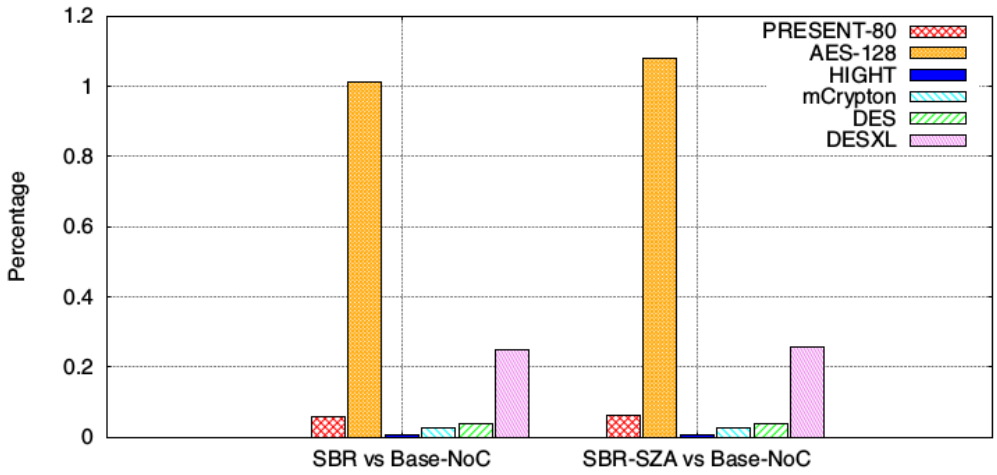


Figura 9: Variação média da latência utilizando os benchmarks NAS com diferentes técnicas de criptografia [FER16].

3.4 Fort-NoCs: Mitigating the Threat of a Compromised NoC

Ancajas et al. [ANC14] apresentam um método realizado em tempo de execução, visando proteção contra ataques realizados por Hardware Trojans (HTs). Os Autores criaram um cenário de ataque em uma NoC através da instanciação de um HT no roteador.

No roteador existem *p* portas, com seus respectivos canais virtuais. Um HT duplica os pacotes que chegam da porta local do roteador, oriundos da interface de rede (NI), podendo inserir estes em cada uma das demais portas, como mostra a Figura 10. O HT observa os dados oriundos da NI, verificando os mesmos para disparar um ataque. O HT opera como uma máquina de estados, com 3 estados:

- *Inatividade*: o HT está inativo, aguardando o dado que o dispare (*trigger*) para iniciar o processo de ataque.
- *Espera*: o HT está ativado, aguardando os comandos de um processo malicioso.
- *Vazamento*: o HT passa a enviar os dados sensíveis direcionados à porta local para um determinado processo malicioso que receberá estes dados.

A avaliação de potência e área ao inserir o HT na NoC foi obtida através da síntese do roteador descrito no nível RTL, com tecnologia TSMC de 45nm. Os resultados mostraram um aumento de 4,62% na área e 0,28% na potência dissipada. Para o resultado de latência foi utilizado o simulador GEM5. A Figura 11 ilustra a função de distribuição de probabilidade correspondente ao aumento da latência média versus o número de possíveis caminhos utilizado pelo HT (*theft paths*). Em média, 83% dos caminhos obtidos pelo HT

não impactam na latência (0%), 14% aumentam a latência entre 0 a 5%, e apenas 0,25% dos caminhos aumentam a latência em 10%. Dado este resultado, é perceptível que uma NoC comprometida representa uma ameaça importante à segurança do sistema.

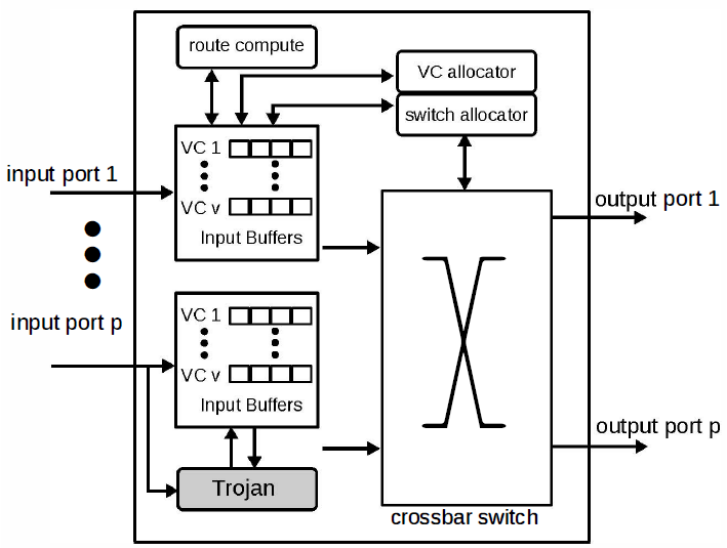


Figura 10: NoC danificada com Hardware Trojan [ANC14].

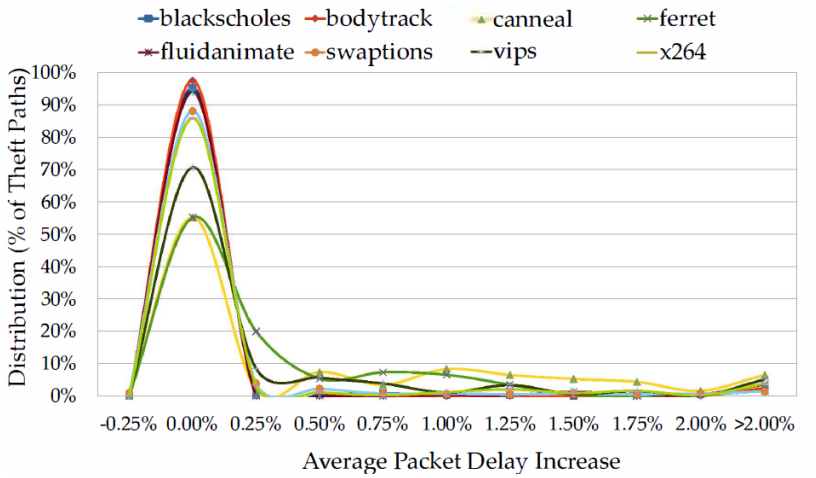


Figura 11: Aumento da latência média versus o número de possíveis caminhos utilizado pelo HT (theft paths). [ANC14].

Os Autores propõem a NoC Fort-NoC, que possui um mecanismo de segurança organizado em três camadas. Essas medidas de segurança são inseridas na interface de rede (NI) da NoC.

O primeiro nível é o *Data Scrambling* (DS), o qual embaralha os dados antes de injetá-lo na NoC. Consequentemente, a chave de ativação do HT será confundida, bloqueando assim sua ativação. Além de impedir ativações, também torna os dados vazados incompreensíveis para o atacante. Este processo de embaralhamento é realizado com criptográfica baseada em portas XOR.

A segunda camada é a certificação de pacotes (PC, do inglês *Packet Certification*) que vincula uma etiqueta (do inglês *tag*) criptografada no fim do pacote antes de injetá-lo na NoC. Esta etiqueta é verificada no destino para confirmar a autenticidade do pacote. O

objetivo do PC é fornecer uma segunda camada de segurança, caso a anterior seja violada. Embora a técnica anterior seja eficaz na prevenção de ativações do HT, este ainda pode ser ativado utilizando outros métodos. Por exemplo, a literatura reporta outras formas de ativar os HT, com gatilhos (do inglês, *triggers*) térmicos ou gatilhos baseados em tempo.

A terceira camada é denominada de ofuscação (*node obfuscation*). Esta terceira técnica consiste em migrar periodicamente as tarefas, de tal forma a impedir os ataques realizados pelos HT.

A avaliação da área adicional para as técnicas PC e DS correspondem a um aumento de 0,34% e 9,57%, respectivamente. A potência consumida se mantém constante para a técnica PC e para a técnica DS, um aumento de 5,8%. A penalidade de desempenho observada nos experimentos foi de 5,9%.

3.5 Runtime Mitigation of Illegal Packet Request Attacks in Networks-on-Chip

O artigo apresentado por Prasad et al. [PRA17] propõe uma arquitetura segura para roteadores, denominada SeRA. Essa arquitetura também visa proteger a NoC contra HTs, implementados no *buffer* do roteador. Em um MPSoC, o processamento dos núcleos permanece ocioso por algum tempo durante o mapeamento das aplicações. Neste tempo de ociosidade, os PEs não injetam pacotes na rede. Durante esse tempo, a fila local, ou seja, a fila que corresponde ao núcleo de processamento, não requisita o acesso para à rede. Os HTs podem desenvolver um ataque DoS, ou seja, escolhem este período de mapeamento da aplicação para gerar um envio de pacote ilegal para a lógica de controle do roteador. Os autores denominam este ataque como *illegal packet request attack* (IPRA).

A Figura 12(a) apresenta o aumento do tempo de execução da aplicação *raytrace* e o aumento de área do roteador, à medida com que aumenta o número de HT implementados no roteador. Uma característica importante dos HTs é o baixo aumento de área quando inserido no roteador, tornando-os praticamente indetectáveis. O aumento da área foi de 2,18% para 20 HTs implementados. O aumento de área e potência consumida pelos HTs são considerados muito pequenos, requerendo poucas portas XOR para causar um ataque. O cenário de ataque proposto é apresentado pela Figura 12(b), a qual apresenta a arquitetura de um roteador com um Hardware Trojan (HT).

O artigo assume que o ataque por HT é direcionado para o cabeçalho do pacote, nos campos de origem, destino e a identificador do *flit*. Como contramedida, foi proposto uma unidade de segurança (SU, do inglês Security Unit), o qual foi projetada para inibir o ataque e restaurar a segurança da comunicação. Sua principal função é validar o cabeçalho do pacote, o qual contém: a origem pacote, destino pacote e o identificador do *flit*.

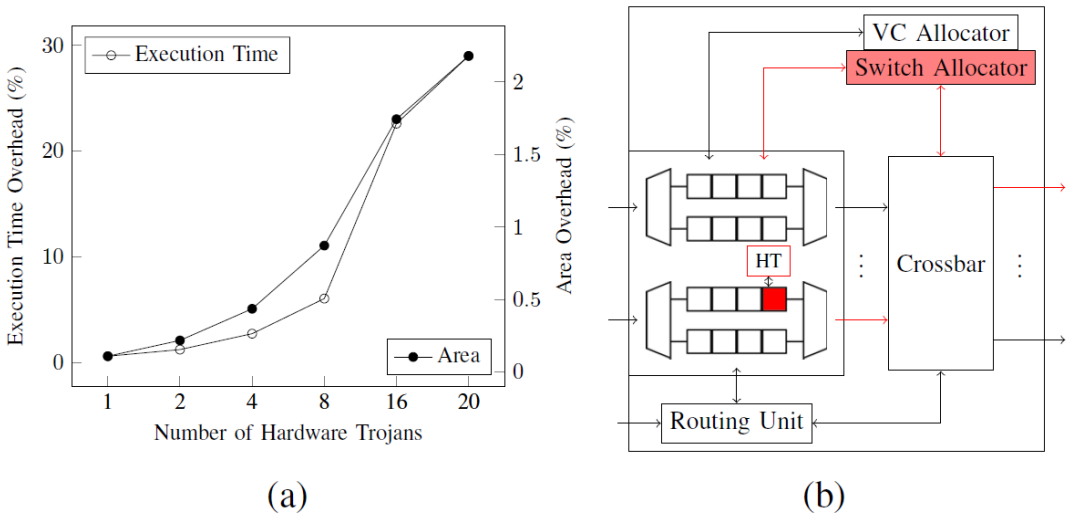


Figura 12: (a) Tempo de execução e aumento da área devido a inserção de HTs. (b) arquitetura de um roteador com HT [PRA17].

A Figura 13(a) apresenta a lógica do *SU* para uma NoC do tipo *mesh* 2x2 e a arquitetura proposta SeRA. A Figura 13(b) apresenta o posicionamento do módulo *SU* no roteador. Esta lógica monitora os *buffers,* sinalizando a detecção de ameaças. Ao detectar-se uma ameaça, posições dos *buffers* são isoladas. Isso pode degradar o desempenho da rede, já que o número de posições nos buffers é reduzido.

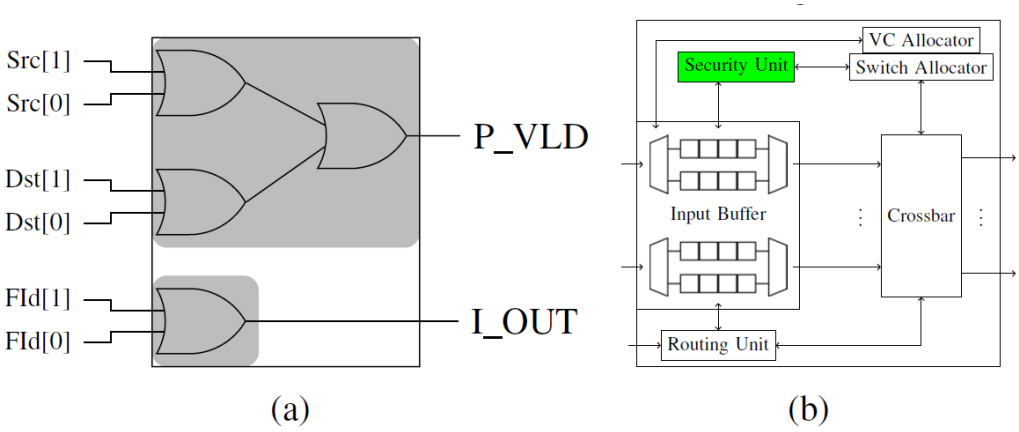


Figura 13: (a) Lógica da unidade de segurança (SU). (b) A proposta da arquitetura segura do roteador [PRA17].

O roteador foi modelado em VHDL, para uma frequência de 1GHz e tecnologia de 90 nm. O aumento da área e potência correspondeu a 1,69% e 0,63%, respectivamente, quando comparado a um roteador de referência.

A proposta de arquitetura do roteador seguro (SeRA) reduziu o tempo de execução, e a energia consumida das aplicações, quando comparados os cenários sob ataque - Figura 14.

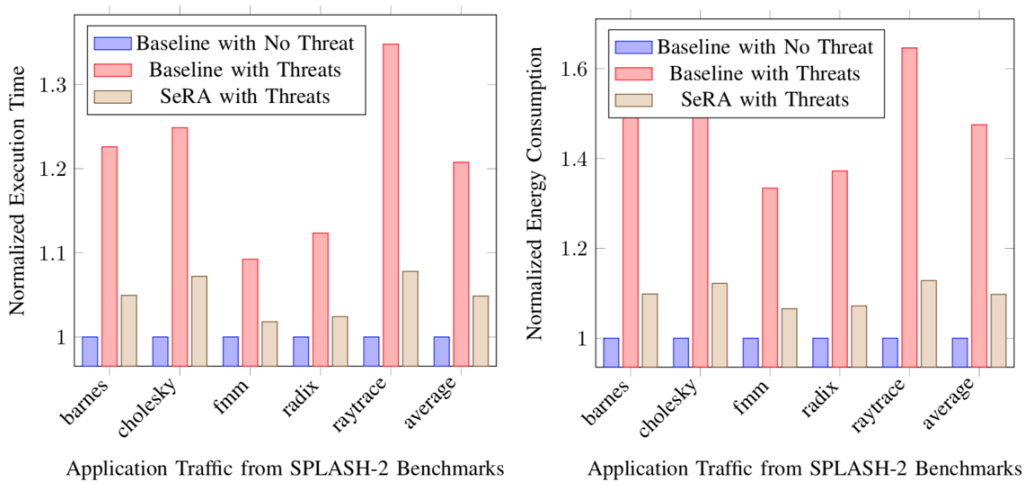


Figura 14: Comparação de desempenho do SeRA com o roteador de referência [PRA17].

3.6 Gossip NoC - Avoiding Timing Side-Channel Attacks through Traffic Management

O trabalho proposto por Reinbrecht et al. [REI16] apresenta a execução de um ataque baseado em tempo de forma distribuída (DTA, do inglês *distributed timing attack*) em MPSoC. O trabalho também é um aperfeiçoamento de arquitetura NoC para segurança, chamada de Gossip NoC. Quando múltiplos núcleos de processamento são infectados por ataques baseados em tempo (do inglês, *timing attack*), eles podem trabalhar coordenadamente para implementar o DTA. Há 2 papéis que os processos maliciosos podem assumir: injetores e observadores. Os injetores têm o papel de injetar dados com uma alta taxa de injeção, com o intuito de congestionar o caminho atacado. Os observadores têm a função de injetar dados com uma baixa taxa de injeção e observar o atraso resultante de seus pacotes, monitorando assim o tráfego do caminho atacado.

O cenário de ataque apresentado pela Figura 15 mostra um ataque a um MPSoC com injetores (IP14 e IP9) e 1 observador (IP8). A aplicação sensível, mapeada no IP 12, inicia a transferência de dados utilizando criptografia. Este IP acessa a memória compartilhada, IP11, fluxo destacado com linhas tracejadas vermelhas. Quando ocorrer um *miss* nesta memória compartilhada é criado um novo fluxo de comunicação para acessar a memória principal, IP0, fluxo destacado com linhas tracejadas azuis. Os processos maliciosos irão atacar o fluxo de comunicação do IP4, com o objetivo de congestionar a rede. Os autores apresentam um processo de calibração dos dados injetados pelos injetores com o intuito reconhecer um pacote sensível trafegando na rede. Portanto, qualquer pacote que trafega junto ao tráfego do invasor, causa saturação na rede, sendo assim detectado pelo observador. O conhecimento do comportamento da rede é uma das suposições que o artigo assume.

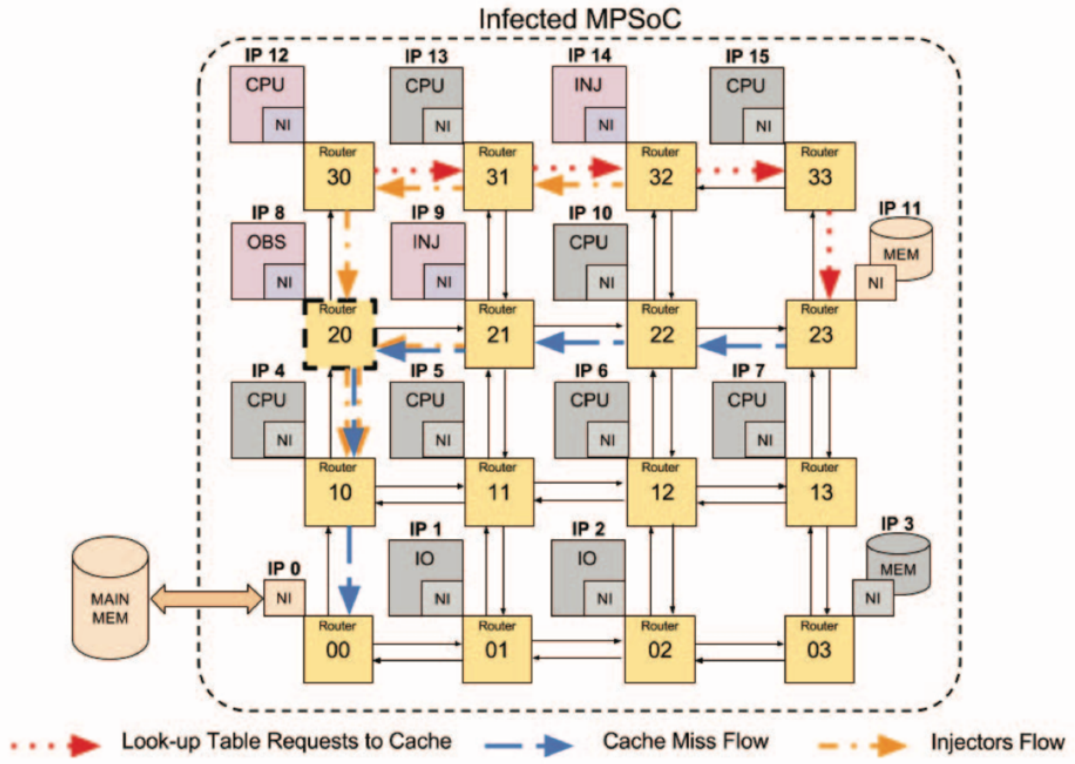


Figura 15: Sistema MPSoC executando uma aplicação sensível, depois do estágio de infecção por injetores e observadores [REI16].

Para uma rede resiliente ao ataque DTA, é proposto um aperfeiçoamento na arquitetura chamado de *Gossip NoC*, o qual possui um monitor de tráfego (o qual envia mensagens de monitoramento) e técnicas de contramedidas implementadas no roteador.

A Figura 16 apresenta a arquitetura do roteador proposta, onde:

- Bloco de entrada: controla o estado interno do roteador de acordo com os sinais de saída dos roteadores vizinhos (*output Gossip*). Quando o número de mensagens *Gossip* recebidas dos roteadores vizinhos supera um dado limiar, um ataque é confirmado. Como resultado, o roteamento dos pacotes é modificado.
- Lógica Gossip: altera a forma de roteamento. Sob ataque, o tráfego é roteado de acordo com o algoritmo YX. Caso contrário, o roteador adota roteamento XY.
- Gerador Gossip: monitora a vazão do tráfego. Quando exceder um limite de vazão, um sinal indicando um possível ataque é ativado e transmitido para todos roteadores vizinhos. Isso configura uma mensagem *Gossip*.

O funcionamento da arquitetura Gossip consiste em alterar o roteamento na presença de um ataque. Portanto, a técnica inibe a ação de um observador (invasor) mapeado no caminho de comunicação. Entretanto, o invasor obterá sucesso se instalar dois observadores nas duas rotas de roteamento.

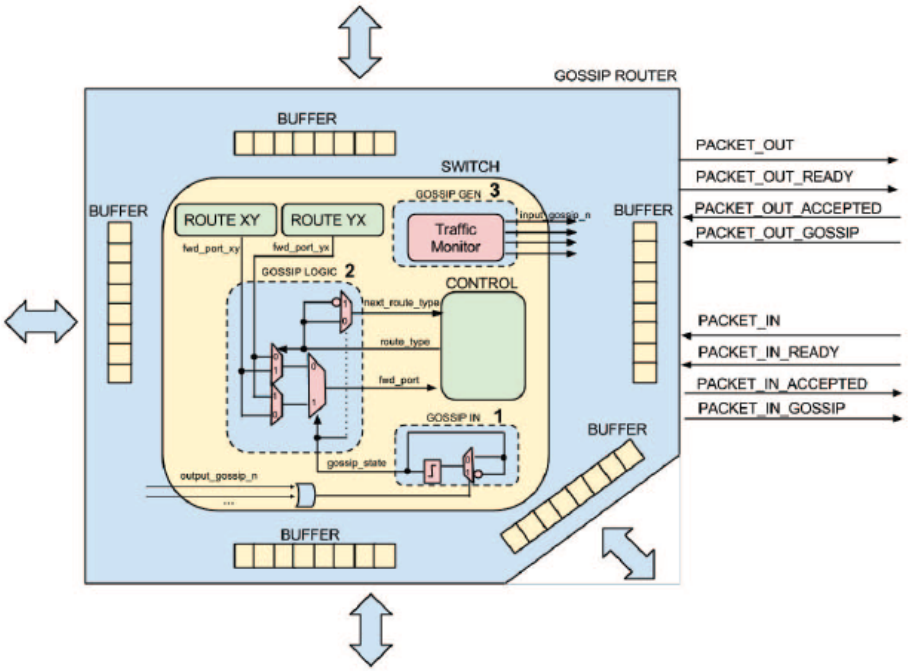


Figura 16: Arquitetura do roteador Gossip: (1) Bloco de entrada Gossip; (2) Lógica do Gossip; (3) Gerador Gossip [REI16]

A Tabela 2 apresenta os resultados obtidos pelo observador (invasor) sobre um ataque DTA. Eficiência corresponde à quantidade de dados sensíveis identificados corretamente pelo ataque. Taxa de sucesso corresponde à porcentagem de amostras acima do limiar, ou seja, correspondendo a dados sensíveis. FP (Falso-Positivo) significa amostras identificados erroneamente, ou seja, aquelas que o invasor não obteve sucesso. FN (Falso-Negativo) representa parte da informação invisível para o invasor. A segunda linha da Tabela mostra que o ataque DTA sobre a rede desprotegida tem alta eficiência, e os dados observados correspondem de fato aos dados sensíveis a serem atacados (alta taxa de sucesso e reduzido número de falsos positivos). Por outro lado, observa-se pela tabela que a técnica *Gossip* reduziu significativamente a eficiência do ataque DTA, e que a maioria dos dados observados são falsos positivos (FP) ou seja, das medidas executadas pelo observador, 93,43% ele não conseguiu identificar os dados sensíveis.

Tabela 2: Resultados de um ataque para o roteador padrão e para o Gossip.

	Eficiência	**Taxa de Sucesso**	**FP**	**FN**
DTA - Unprotected NoC	59,63%	96,92%	3,07%	40,36%
DTA - Gossip	6,62%	6,56%	93,43%	93,37%

A Tabela 3 apresenta resultados de síntese para um roteador padrão e para o roteador Gossip proposto por uma tecnologia de 65 nm a 1 GHz. Os resultados mostram um aumento 21,16% de área e um aumento de 16,2% de potência consumida, sendo considerados baixos em relação ao aumento de segurança oferecido ao sistema.

Tabela 3: Resultados de Síntese para um roteador padrão e o Gossip.

	NoC de referência	Gossip NoC	Overhead
Área (μm²)	2632	3189	21,16%
Potência (mW)	2,073	2,409	16,2%

3.7 A Non-Intrusive and Reconfigurable Access Control to Secure NoCs

Fernandes et al. [FER15] apresentam a inserção de *firewalls* para garantir a proteção contra invasores na rede. A arquitetura dos *firewalls* tem por objetivo filtrar dados de saída e entrada dos PEs.

Os *firewalls* são previamente configurados com regras de permissão para envio e recepção de dados. Cada *firewall* controla a recepção e a transmissão de pacotes. Por exemplo, na transmissão de um pacote o *firewall* verifica a origem e o destino do pacote. Havendo uma entrada para esta dupla nas regras de transmissão do *firewall*, o pacote pode ser injetado na rede. Na recepção o processo é semelhante, com verificação da dupla destino-origem, e entrada nas regras de recepção do *firewall*. A latência para este processo de recepção/transmissão é de 3 ciclos de relógio. A Figura 17 ilustra onde o *firewall* é inserido no sistema.

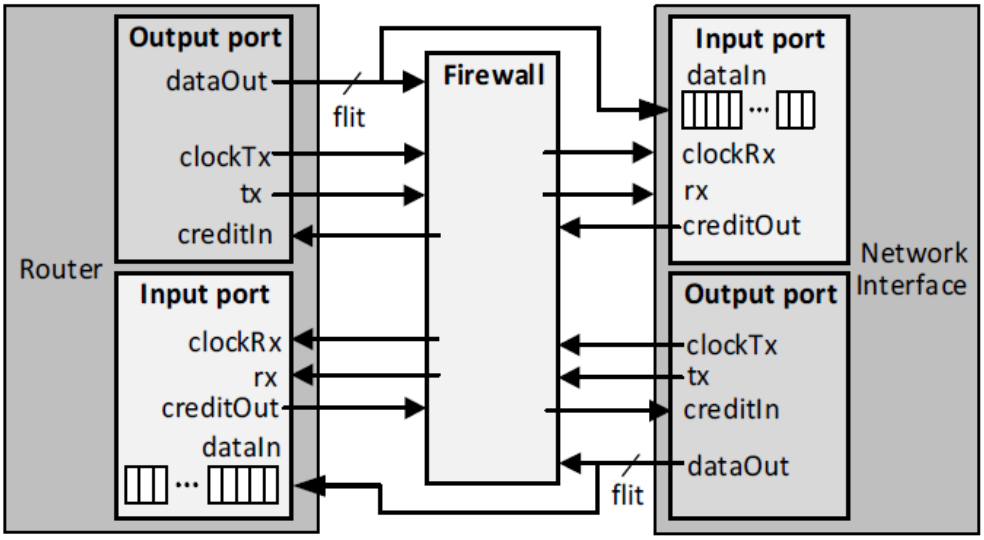


Figura 17: Interface da porta da NoC e interconexão do *firewall* [FER15].

Os *firewalls* são interconectados por um caminho Hamiltoniano, conectando os *firewalls* em série. Esta rede é separada da rede de dados, impedindo o acesso para extração de dados da mesma. Assim garante-se a integridade, confiabilidade e proteção dos pacotes desta rede. A Figura 18 mostra a conexão dos *firewalls* pelo caminho Hamiltoniano.

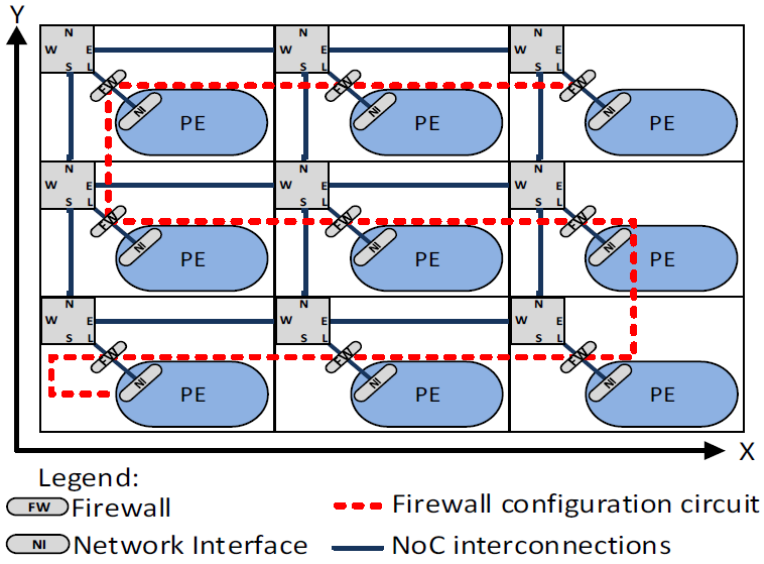


Figura 18: Localização dos *firewalls* em uma NoC do tipo mesh 3x3 [FER15].

Os Autores avaliaram o controle de acesso, tempo de configuração dos *firewalls*, a área consumida e a escalabilidade da proposta. Sobre o controle de acesso, os *firewalls* exerceram seu papel corretamente, descartando pacotes oriundos de outros remetentes e concedendo acesso somente para pacotes previsto nas regras armazenadas no *firewall*.

O tempo de configuração dos *firewalls* foi analisado para o cenário de pior caso, onde a origem e o destino têm a maior distância na rede. Para uma rede de 4x4 o tempo de configuração corresponde a aproximadamente 720 ciclos de relógio. Na avaliação do aumento da área devido a inserção de firewalls, foi estimado que cada *firewall* aumenta cerca de 13,91% a área do roteador e quando somados ao sistema em uma configuração de 4x4 aumentam cerca de 12,4%. Sobre a escalabilidade do sistema com os firewalls, foi analisado diversos tamanhos de NoCs, onde se concluiu que o aumento de área, em média, foi de 13% em NoCs variando a dimensão de 3x3 até 8x8.

3.8 Enabling Secure MPSoC Dynamic Operation through Protected Communication

Azad et al. [AZA18] apresentam uma infraestrutura de segurança baseada na utilização de *firewalls* e mecanismos de autenticação leve para agregar segurança em um MPSoC. O trabalho também analisa o custo área e desempenho agregados ao sistema. No MPSoC, a infraestrutura de segurança do trabalho é composta por um gerente de segurança e um conjunto de interfaces de segurança que agrega o *firewall*.

O gerente de segurança é responsável por implantar políticas de segurança ao sistema, o qual permite a entrada de pacotes ou o descarte de pacotes dependendo das políticas de acesso estabelecidas. Caso o pacote for descartado um alarme é disparado. O *firewall* possui duas tabelas:

- Tabela do remetente (do inglês, *initiator table*): responsável por verificar os direitos da comunicação do remetente;
- Tabela do destinatário (do inglês, *target table*): responsável por verificar os direitos da comunicação do pacote recebido do destinatário.

A Figura 19 mostra o diagrama de blocos da arquitetura proposta para os *firewalls* alocados na NI (Network Interface), em laranja são os blocos agregados na NI de referência. O gerente de segurança está encarregado de emitir comandos de reconfiguração para as NIs. Ao receber comandos de reconfiguração, a NI verifica a autenticidade e direitos e inicia o processo de reconfiguração (armazena novos valores na tabela do *firewall*).

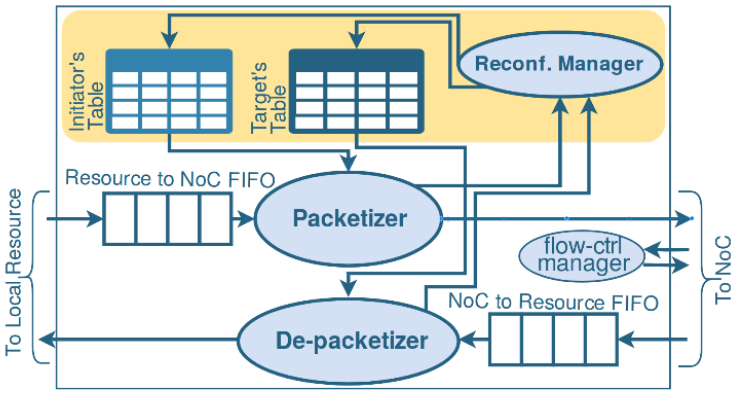


Figura 19: Arquitetura do *firewall* proposto [AZA18].

Foi realizada uma análise de área no nível RTL com a tecnologia CMOS TSMC de 40 nm com objetivo de comparar uma NI de referência (sem qualquer mecanismo de segurança) com a NI com firewall para verificação dos custos ao agregar segurança ao sistema, conforme mostra a Figura 20. Houve um aumento na área de 139% NI com firewalls instalados, onde cada firewall possui 15 entradas. Adicionando o firewall a latência do caminho crítico aumentou 6,46%.

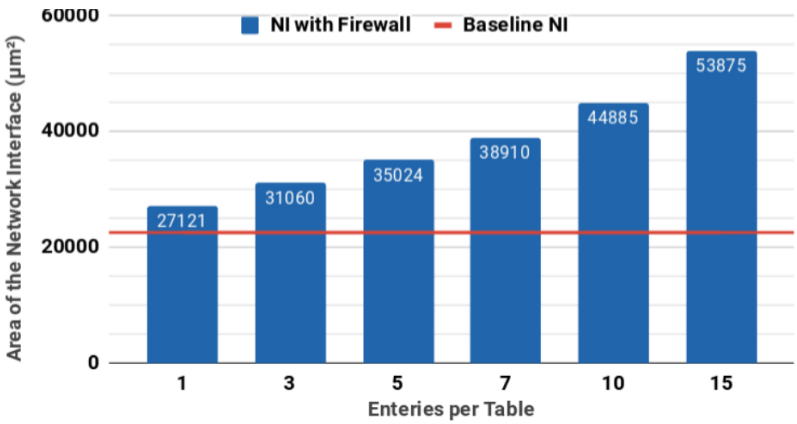


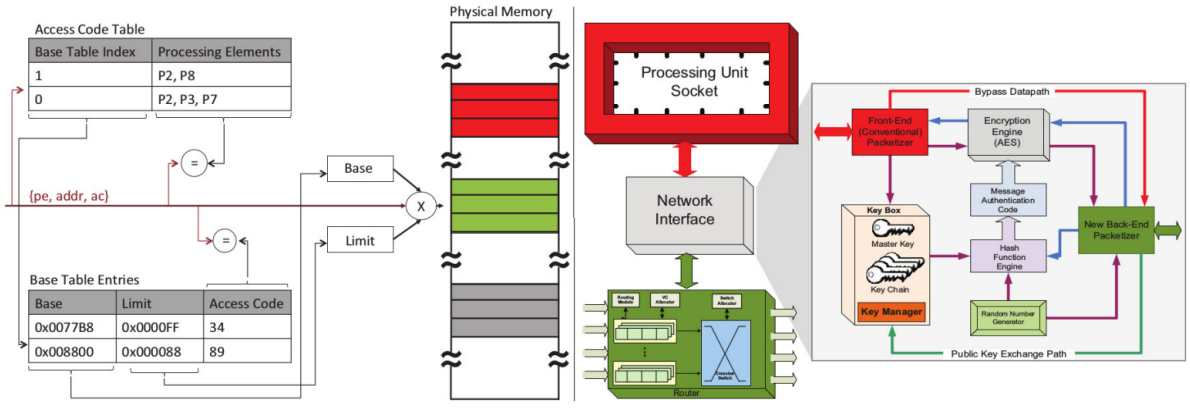
Figura 20: Área do *firewall* variando-se o número de entradas do mesmo [AZA18].

3.9 Hermes: Secure Heterogeneous Multicore Architecture Design

No artigo de Kinsy et al. [KIN17], os autores apresentam uma plataforma MPSoC com suporte à mecanismos de segurança, denominada Hermes. A proposta visa evitar ataques do tipo DoS, ataques em canais virtuais (enlaces da NoC) e ataque na memória física criando uma camada de virtualização que isola os segmentos de computação com base em níveis de confiança e políticas de segurança definidos pelo sistema e pelo usuário.

A Hermes obtém zonas de processamento de hardware e software de modo seguro, agrupando processadores em zonas físicas chamada pelos Autores de *alas* (do inglês, *wards*) e zonas lógicas virtuais chamadas de ilhas. As alas têm diferentes níveis de segurança definidos no momento do projeto e cada uma delas tem um nodo responsável por seu gerenciamento de chaves.

A plataforma protege a memória física usando uma MMU (*Memory Management Unit*) com restrições de acesso baseadas em duas tabelas (tabela de códigos de acesso e a tabela de entrada básica) que fornecem um comportamento de *firewall* para a MMU (Figura 21(a)). A Hermes obtém proteção de comunicação usando vários mecanismos: (*i*) o protocolo DH (*Diffie-Hellman*) distribui chaves públicas para os nodos das zonas físicas, com operações de *join* e *leave* para distribuir as chaves para as zonas lógicas; (*ii*) um módulo AES na NI para criptografar o fluxo de tráfego quando necessário; (*iii*) um gerenciador de chaves na NI para selecionar uma chave apropriada durante a operação; (*iv*) MAC e módulos HASH para gerar chaves de sessão (MMU (Figura 21(b)). Além desses mecanismos, o ambiente fornece um algoritmo de roteamento para proibir ou limitar a travessia de zonas por tráfego gerado por não-membros.



(a) (b)

Figura 21: (a) Mecanismo de acesso a memória. (b) Arquitetura do PE da Hermes com a NI detalhada [KIN17].

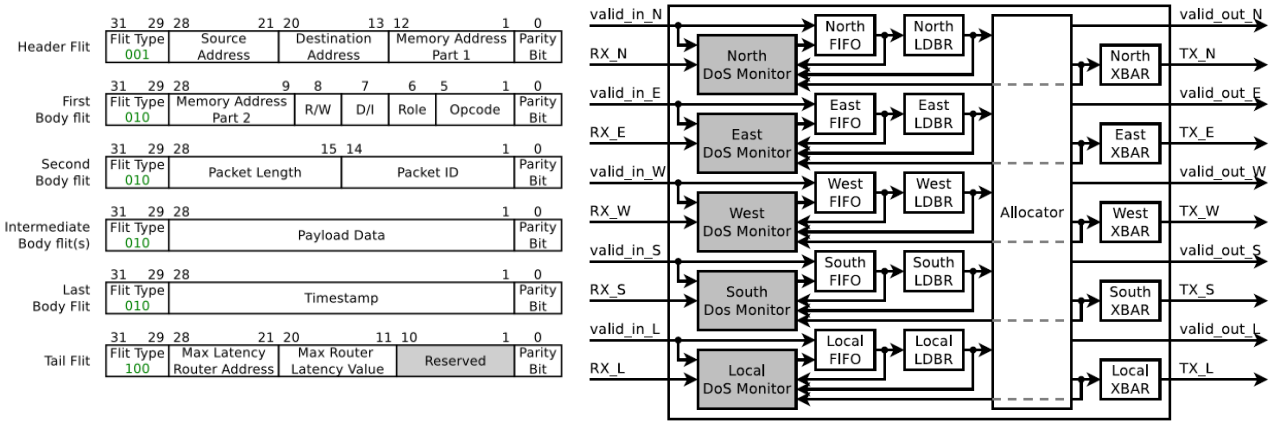
De acordo com os Autores, o aumento de área para implementar os recursos de segurança da arquitetura Hermes é 17%. Os resultados de desempenho nos benchmarks

do SPLASH-2 apresentam um aumento de 1% a 9% quando comparados com a arquitetura de referência não segura.

3.10 A Distributed DoS Detection Scheme for NoC-based MPSoCs

Chaves et al. [CHA18] propõe uma nova técnica de detecção de DoS distribuído que seja capaz de medir a degradação do desempenho de fluxos sensíveis, e também capaz de detectar o roteador que inicia o ataque responsável por interferir no fluxo de comunicação sensível.

A Figura 22(a) mostra a estrutura dos pacotes para monitor os ataques DoS. Os Autores acrescentaram um flit na estrutura do pacote, contendo a marcação de tempo específica de cada pacote (*timestamp*, no último flit do *payload*), cuja função é registrar a latência na transmissão de cada pacote. Com base nesta informação, os Autores argumentam que é possível detectar a posição do ataque DoS. A segunda modificação na estrutura do pacote é relativa ao último flit (*tail flit*), o qual é responsável por armazenar o endereço do roteador e o tempo de espera, em ciclos de relógio. Este tempo é relativo ao tempo que o pacote ficou esperando ser atendido pelo roteador, pois outros pacotes estavam utilizando a porta de saída. Estas informações são analisadas e atualizadas em cada roteador pelo monitor de DoS. Os Autores também acrescentaram no roteador de referência um monitor de DoS em cada porta do roteador, conforme apresenta a Figura 22(b).



(a) (b)

Figura 22: (a) Estrutura do pacote. (b) Arquitetura do roteador proposto [CHA18].

Os Autores verificaram a eficiência dos monitores de DoS através da implementação dos ataques DoS e a detecção do mesmo. De acordo com os Autores, o aumento de área para implementar os monitores de DoS é de 17% do roteador (cada monitor aumenta a área em 3,4% do roteador) e o atraso gerado pelos monitores num caminho crítico, pode ser desprezado. A potência dissipada aumentou 5 % em comparação com a arquitetura de referência.

3.11 Considerações finais

Foram apresentados trabalhos visando proteção a diferentes tipos de ataques, sendo, em sua maioria, seis ([SEP15], [CAI17], [FER16], [REI16], [FER15],[AZA18] [KIN17] e [CHA18]) explorando a vulnerabilidades de ataques baseados em software e suas contramedidas. Dois trabalhos ([PRA17] e [REI16]) exploram a vulnerabilidades de ataques baseados em hardware e a respectiva contramedida.

Os trabalhos [CAI17] e [FER16] apresentam a técnica de zonas seguras para agregar segurança a plataforma em MPSoCs. O trabalho proposto em [SEP15] que evita o ataque DDoS, o trabalho propõe implementar um roteamento adaptativo na NoC ou implementar arbitragem randômica nos roteadores da NoC, ou seja, não agregam nenhum módulo à infraestrutura existente. Os trabalhos [REI16], [FER15], [AZA18] e [CHA18] propõem uma dispositivos de segurança que agregam segurança a plataforma MPSoC. Em [REI16] é proposta uma nova arquitetura resiliente a ataques DTA. O trabalho [KIN17] também propõe uma arquitetura segura que evita ataque DoS, ataques em canais virtuais (enlaces da NoC) e ataque na memória física. Em [FER15] são propostos firewalls não intrusivos ao sistema para garantir a proteção contra invasores da rede. Em [AZA18] é acrescentada a NI uma infraestrutura que gerencia as políticas de segurança implementadas a sua plataforma. Os trabalhos [PRA17] e [REI16] mostram que HTs são possíveis de serem implementadas em MPSoCs e são difíceis de serem detectados.

No estado-da-arte, os Autores, em sua maioria, mostram a implementação do ataque e seu impacto no sistema, e propõe uma contramedida analisando seus custos. Assim apresentam um compromisso entre o aumento de área e potência consumida para obter segurança. A Tabela 4 apresenta um resumo do estado da arte dos artigos descritos no presente Capítulo.

A proposta deste trabalho possui como diferencial ao estado-da-arte o uso de uma rede dedicada para envio de informações sensíveis, como chaves de criptografia e identificador das aplicações. Esta rede dedicada é segura, pois apenas o processador responsável por gerenciar o sistema tem acesso de escrita à rede. Outro diferencial do trabalho proposto nesta Dissertação é o uso de *firewalls* com reduzida área de silício*,* o qual possui uma função de bloquear ou permitir determinados fluxos em função de regras de segurança. Utilizamos um filtro de pacotes*,* ou seja, um módulo simples de hardware, que armazena apenas a chave de criptografia e a identificação da aplicação, diminuindo assim, a complexidade e área de silício ocupada pelo firewall.

Tabela 4: Resumo do estado da arte.

Ref.	Tipo de ataques	Análise do impacto do ataque no sistema	Técnicas de Contramedida	Análise de Custo da Contramedida		
				Potência	Área	Desempenho
[SEP15]	Time-Driven Attacks, DoS	Desempenho	Mudança no roteamento e árbitro do roteador	Sim	Sim	Sim
[CAI17]	DoS, Timing Attack e Spoofing	Área e desempenho	Zonas seguras opacas	Não	Não	Sim
[FER16]	Timing Attacks e DoS	Desempenho	Zona segura com criptografia	Não	Não	Sim
[ANC14]	HT	Área, potência e desempenho	Criptografia, autenticação do pacote e migração de tarefas	Sim	Sim	Sim
[PRA17]	HT e DoS	Área e desempenho	Implementação de uma unidade de segurança no roteador.	Sim	Sim	Não
[REI16]	DTA, timing attack	Desempenho	Mensagens G*ossip*, roteador com monitoramento, mudança de roteamento	Sim	Sim	Não
[FER15]	DoS e Hijacking attacks	Área	Utilização de *firewalls*	Não	Sim	Não
[AZA18]	DoS e Hijacking attacks	Integridade, Disponibilidade e confiabilidade	Utilização de *firewalls*	Não	Sim	Sim
[KIN17]	DoS, ataques em canais virtuais e ataque na memória física	Disponibilidade, Integridade e Confiabilidade	Utilização de *firewalls*	Não	Sim	Sim
[CHA18]	DoS	Disponibilidade	Utilização monitores de tráfego	Sim	Sim	Sim
Proposta	**DoS, Spoofing, ataque na memória física**	**Desempenho**	**Utilização de *filtros*, criptografia, chaves criptográficas dinâmicas, rede privada para segurança.**	**Não**	**Sim**	**Sim**

4 PLATAFORMA DE REFERÊNCIA: HeMPS

Este Capítulo apresenta a arquitetura de referência, HeMPS [CAR09], no nível de software e no nível de hardware. A arquitetura HeMPS é um MPSoC homogêneo simétrico, ou seja, todos elementos de processamento (PEs) possuem a mesma arquitetura e organização. Há uma distinção dos elementos de processamentos: PEs escravos (SP, do inglês *slave PE*), responsáveis por executar as tarefas de aplicações; PE gerentes (MP, do inglês *manager PE*) que são responsáveis pelo mapeamento e gerenciamento das aplicações e tarefas executadas no sistema. Os MPs se distinguem em *LMP – local manager* PE que tem o objetivo de gerenciar o *cluster*, e *GMP – global manager* PE que gerencia todo o sistema. É utilizado um repositório de tarefas externo à arquitetura HeMPS, o qual contém as aplicações que serão executadas no sistema. Este repositório está conectado diretamente a um MP. A Figura 23 apresentada o MPSoC HeMPS.

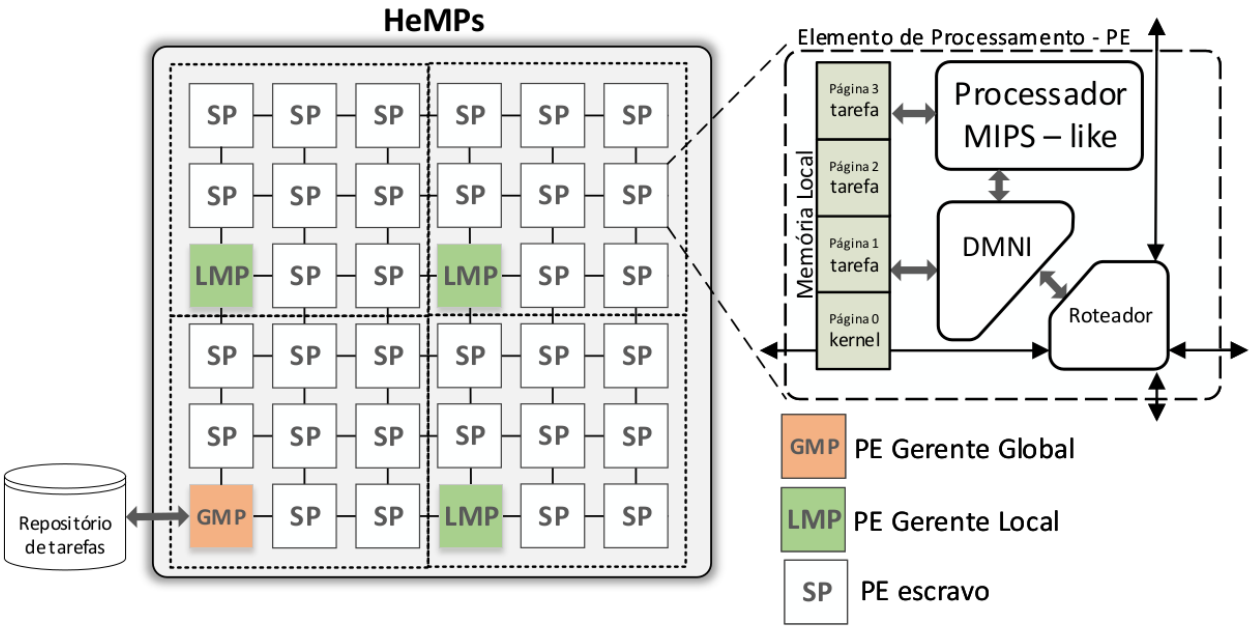


Figura 23: Instância 2x2 do MPSoC HeMPS (fonte: Autor).

As características relevantes do MPSoC HeMPS incluem:

- Processamento homogêneo. Todos os PEs possuem a mesma arquitetura. Justifica-se o emprego de um MPSoC homogêneo para simplificar a geração dos códigos objetos e a migração de tarefas.
- Utilização de rede intrachip (NoC). Justifica-se o emprego desta arquitetura de comunicação dada a escalabilidade da mesma e a possibilidade de múltiplas comunicações entre PEs ocorrerem simultaneamente.
- Memória distribuída. Cada processador possui uma memória privada, responsável por armazenar instruções e dados.

- Comunicação por troca de mensagens. Justifica-se o emprego de troca de mensagens, pois este é o mecanismo natural de comunicação em um sistema com memórias distribuídas. Há duas formas de geração de mensagens. A primeira é via uma API *send-receive*, onde as tarefas apenas informam o destino e o conteúdo da mensagem e o kernel preenche o cabeçalho. A segunda API permite que a tarefas gerem uma mensagem (conteúdo e cabeçalho), oferecendo aos programadores flexibilidade para a geração das mesmas (por exemplo, para a comunicação com periféricos).
- Organização de memória paginada. Cada tarefa tem reservada uma página de memória no PE, ou seja, a página de memória é vista como um recurso de processamento de tarefas, podendo a tarefa ser mapeada em qualquer página livre. Consequentemente, não são necessários mecanismos complexos de gerência de memória.

4.1 Elemento de Processamento (PE)

Todos os PEs do MPSoC possuem o mesmo hardware, porém possuem sistemas operacionais diferentes. O PE do MPSoC HeMPS possui duas instâncias distintas como mencionado acima, *SPs* e *MPs*.

Os MPs são responsáveis pela gerência dos recursos do sistema, executando funções como admissão de aplicações ao sistema (GMP), mapeamento e migração de tarefas alocadas no sistema. Os MPs apenas executam funções de gerenciamento, evitando assim que tarefas maliciosas se instalem nele. Os MPs são projetados para possuir algumas propriedades de segurança, como integridade, disponibilidade e confiabilidade.

Os SPs são responsáveis por executar tarefas de aplicações. Os SPs possuem um SO multitarefa. É importante destacar que aplicações maliciosas podem ser mapeadas nos SPs, pois no momento do mapeamento não é conhecido o comportamento das tarefas das aplicações.

Os PEs do MPSoC HeMPS contêm os seguintes componentes:

- processador: cada PE contém um processador RISC de 32 bits com subconjunto de instruções da arquitetura MIPS [RHO01];
- memória privada (RAM): contém o sistema operacional, denominado *microkernel*, executado pelo processador Plasma. No caso dos SPs, a memória é dividida em páginas de tamanho fixo, onde é feita a alocação de tarefas;
- módulo de acesso direto à memória e interface de rede (DMNI) [RUA16]: possibilita o processador continuar a execução de tarefas sem controlar diretamente a troca de mensagens com a rede e também realiza a interface entre o Plasma e a NoC Hermes. É responsável pelo envio e recebimento de pacotes na rede.

4.2 NoC Hermes

A interconexão dos PEs é realizada através da NoC Hermes [MOR04]. Esta NoC é parametrizável e possui topologia malha 2D. O mecanismo de comunicação é realizado por chaveamento de pacotes, utilizando o modo *wormhole*, no qual um pacote é transmitido entre os roteadores através de *flits*.

Os roteadores da NoC possuem buffers de entrada, uma lógica de controle compartilhada por todas as portas do roteador, um *crossbar* interno e até cinco portas bidirecionais. A porta Local estabelece a comunicação entre o roteador e o módulo DMNI, sendo as demais portas utilizadas para conectar o roteador aos roteadores vizinhos. A arbitragem *round-robin* é utilizada pelo roteador da NoC Hermes. Essa política utiliza um esquema de prioridades dinâmicas, proporcionando um serviço mais justo que a prioridade estática. Cada roteador possui um endereço exclusivo, o qual varia de acordo com o tamanho do MPSoC instanciado. O algoritmo de roteamento utilizado é o XY, que envia os pacotes na rede primeiramente horizontalmente até chegar à coordenada X do roteador destino, e depois percorre verticalmente, até encontrar o roteador destino.

4.3 Repositório de Tarefas

O MPSoC HeMPS assume que todas as aplicações são modeladas através de um grafo de tarefas, onde os vértices representam tarefas e as arestas representam a comunicações entre as tarefas da aplicação. As tarefas sem dependências de recepção de dados são denominadas iniciais (tarefa A na Figura 24). No momento de inicialização de uma dada aplicação, a heurística de mapeamento carrega no MPSoC somente as tarefas iniciais. As demais tarefas são inseridas dinamicamente no sistema em função das requisições de comunicação e dos recursos disponíveis. A Figura 24). mostra um exemplo de uma aplicação modelada de acordo com esta abordagem.

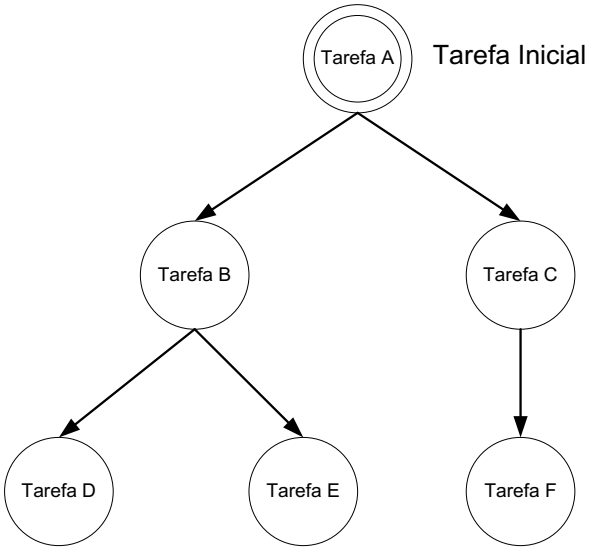


Figura 24: Exemplo de aplicação modelada por um grafo de tarefas [CAS13].

O repositório de tarefas é uma memória externa ao MPSoC que contém o código-objeto de todas as tarefas que executarão no sistema. As primeiras posições do repositório de tarefas contêm os descritores de cada tarefa, com informações como identificador único, posição inicial no repositório de tarefas, tamanho da tarefa, dentre outras informações

4.4 Comunicação entre Tarefas

O sistema operacional suporta duas APIs de comunicação: orientada a serviços e *raw* (direta). A comunicação orientada a serviços é realizada através de geração do pacote pelo *microkernel*, armazenando as mensagens em um vetor denominado *pipe*. O *pipe* é uma área de memória reservada para a troca de mensagens. Nesta área são armazenadas todas as mensagens das tarefas que executam em um determinado PE. As mensagens são armazenadas de forma ordenada, e consumidas sequencialmente. Por outro lado, a comunicação *raw* deixa a função de criação do pacote para a tarefa do usuário. Este mecanismo é útil para, por exemplo, realizar a comunicação com dispositivos externos, como memórias compartilhadas

Para a comunicação orientada a serviço, duas chamadas de sistema são utilizadas para a troca de mensagens: *WritePipe*() e *Readpipe*(). No nível de aplicação estas chamadas de sistema são utilizadas através das primitivas *Send*() e *Receive*(), que respectivamente chamam as rotinas de *WritePipe*() e *Readpipe*() contidas no *microkernel* do PE. Na implementação do *microkernel* do MPSoC HeMPS, o *Send*() é assíncrono e o *Receive*() síncrono. A principal vantagem desse método é que uma dada mensagem só é injetada na NoC se esta foi requisitada pelo receptor, reduzindo o congestionamento da rede, pois não há a possibilidade de pacotes ficarem bloqueando a rede para serem posteriormente consumidos. Para implementar um *Send*() assíncrono utiliza-se o *pipe*.

A Figura 25 ilustra a comunicação entre duas tarefas, A e B, mapeadas em diferentes PEs. Quando a tarefa A executa um *Send*() (1 na Figura), a mensagem é armazenada no *pipe* e a execução da tarefa A continua (2). Isso caracteriza uma escrita assíncrona não-bloqueante. Quando a tarefa B executa um *Receive*() (3), duas situações podem ocorrer. Se a tarefa destino está mapeada no mesmo processador, a tarefa executa uma leitura no *pipe* local. Se a tarefa está mapeada em outro PE, como nesse exemplo, o *microkernel* envia uma requisição de mensagem através da NoC (4) e a tarefa B entra em estado de espera (estado *waiting*) (5), caracterizando uma leitura síncrona bloqueante. A tarefa A recebe a requisição de mensagem (6) e envia a mensagem através da NoC, liberando espaço no *pipe* (6). Quando a mensagem chega no PE da tarefa B (7), o *microkernel* armazena a mensagem no espaço de memória da tarefa B, habilitando a execução da tarefa B e mudando o seu estado para *ready* (8).

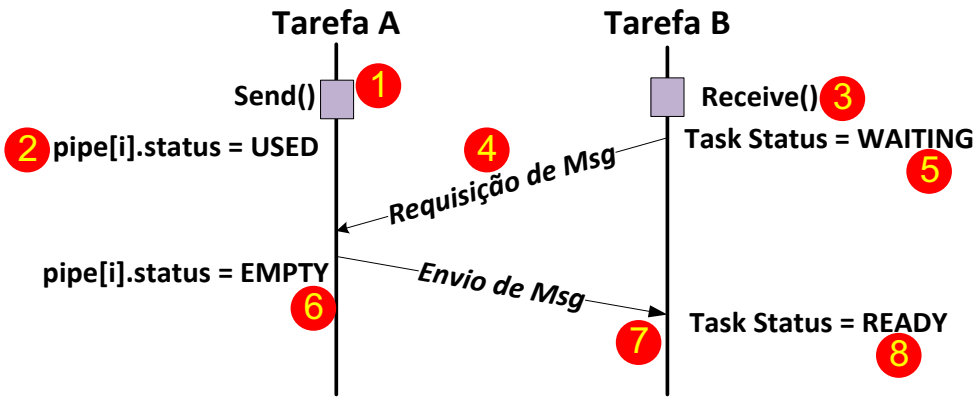


Figura 25: Protocolo de comunicação orientado a serviço [CAS13].

A comunicação *raw* utiliza no nível de aplicação, duas primitivas: *RawSend*() e *RawReceive*(). O *RawSend*() permite a criação do pacote pela tarefa. Todos os dados contidos são enviados diretamente na rede de comunicação pela DMNI, de forma assíncrona. Desta forma é possível injetar pacotes sem nenhuma regra de formatação de pacotes. O *RawReceive*() é bloqueante em nível de execução de tarefas. Quando é chamado ele aguarda até o recebimento das mensagens, sem implementar nenhum protocolo de comunicação como é feito na comunicação orientada a serviço.

A Figura 26 ilustra a comunicação entre duas tarefas, A e B, mapeadas em diferentes PEs. Quando a tarefa B executa um *RawReceive()* (1), a tarefa interrompe sua execução e fica aguardando a chegada de um pacote. Isso caracteriza uma leitura síncrona bloqueante. Quando a tarefa A executa um *RawSend()* (2), a tarefa A envia a mensagem através da NoC (3). Quando a mensagem chega no PE da tarefa B (4), o *microkernel* armazena a mensagem no espaço de memória da tarefa B, habilitando a execução da tarefa B.

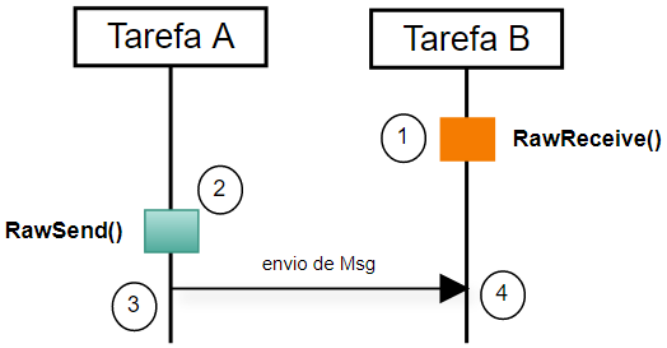


Figura 26: Protocolo de comunicação *raw* (fonte: Autor).

A Tabela 5 apresenta o código de envio e recepção de dados em modo *raw*, no nível de tarefa e *kernel*. No nível de tarefa, o a função para envio de pacote em modo *raw* inicialmente monta o pacote (linhas 3-7), que neste caso corresponde a um pacote para o endereço 0x100, de tamanho 11 no *payload* (0xB), emulando-se um serviço de atualização

de posição de tarefa (0x230). Na sequência, linha 9, faz-se a chamada de sistema RAW_SEND, passando como argumentos o ponteiro do pacote (arg0) e seu tamanho (arg1). Esta linha de código só é liberada após execução da chamada de sistema. No nível de kernel, ajusta-se o ponteiro do pacote de acordo com a página onde a tarefa está executando (linha 2). Na sequência, linha 3, programa-se o módulo DMNI com o ponteiro da área de memória onde está o pacote, e o volume de dados a transmitir. Na linha 4, aguarda-se o final da transmissão do pacote através do registrador mapeado em memória DMNI_SEND_ACTIVE.

No nível de tarefa, a função para recepção de pacote em modo raw apenas aguarda em modo bloqueante que a chamada de sistema RAW_RECEIVE retorne dados. A chamada de sistema RAW_RECEIVE apenas coloca a tarefa em modo de espera (linha 3), e informa que a tarefa em questão deve receber dados em modo raw (linha 4). Quando um pacote é recebido através da DMNI, o processador é interrompido, e executa-se no kernel a função read_packet(). Nesta função verifica-se se há tarefa aguardando dados em modo raw. Caso esta condição for verdadeira, os dados provenientes da DMNI são escritos na área de memória especificada pela tarefa (msg). Após a recepção dos dados, a função do kernel escreve '1' no registrador de retorno ($v0), liberando a tarefa do *while* (este processo de retorno de função não é convencional, dado que há manipulação de registradores do processador diretamente pelo kernel).

Tabela 5: Código de envio e recepção de dados em modo *raw*.

	RawSend()
Tarefa	1. volatile unsigned int data[13]; 2. //Montando os dados do pacote 3. data[0] = 0x100; 4. data[1] = 0xB; 5. data[2] = 0x230; 6. data[3] = 2; 7. data[8] = 0x101; 8. //envia_pacote; 9. while(!SystemCall(RAW_SEND, unsigned int*)data, 13));
Kernel	1. Case RAW_SEND: 2. data_app_ptr = (unsigned int *) (current->offset | arg0); 3. DMNI_send_data((unsigned int) data_app_ptr, arg1); 4. while (MemoryRead(DMNI_SEND_ACTIVE)); 5. return 1;
	RawReceive()
Tarefa	1. while(!SystemCall(RAW_RECEIVE, (unsigned int*)msg, 0));
Kernel	1. case RAW_RECEIVE: 2. schedule_after_syscall = 1; 3. current->scheduling_ptr->waiting_msg = 1; 4. current->raw_recv = 1; 5. return 0;

5 VULNERABILIDADES DA HEMPS

Este Capítulo apresenta uma análise da plataforma HeMPS submetida a diferentes tipos de ataques de software e hardware, de modo a explorar suas vulnerabilidades. Nestes experimentos não há nenhuma contramedida de segurança implementada. Esta análise aborda o nível de hardware e software da plataforma.

5.1 Vulnerabilidades na API de comunicação (kernel)

O *kernel* do MPSoC pode ser comprometido por processos maliciosos como trojans, vírus e ataques em tempo de execução. Os protocolos de comunicação estão sujeitos a ataques, incluindo ataques de *man-in-the-middle* e de DoS [KOU12]. Nesta seção abordaremos um ataque relacionado à comunicação entre as tarefas executadas no MPSoC. O ataque escolhido foi o *man-in-the-middle* (do inglês, homem no meio). Este ataque consiste na interceptação de pacotes através de um processo ou tarefa maliciosa intermediária que pode modificar os mesmos quando a origem se comunica com o destino. Essa tarefa é capaz de receber os dados, transmiti-los e manipulá-los.

Como foi visto no Capítulo 4, as aplicações a serem executadas pelo MPSoC são constituídas de tarefas. As aplicações podem se caracterizar por aplicações comunicantes (que possuem muito tráfego de comunicação) e por aplicações que possuem muita computação de dados. No nível de aplicação, pode-se explorar vulnerabilidades no nível da API de comunicação orientada a serviço. A Figura 27 mostra o grafo de tarefas de uma aplicação, contendo uma tarefa produtora da dados (PROD) e uma tarefa consumidora (CONS).

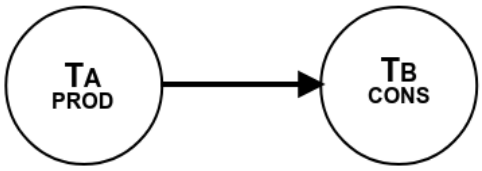


Figura 27: Grafo de tarefa da produtora e consumidora (fonte: Autor).

Na implementação deste ataque, foram utilizadas as APIs orientada a serviço e *raw* (direta). O intruso utiliza exclusivamente a comunicação *raw* e as tarefas produtora e consumidora utilizam a comunicação orientada a serviços.

A Figura 28 exemplifica o ataque *man-in-the-middle* aplicado ao MPSoC, onde T_A é a representada como a tarefa produtora que está alocada no endereço 0x1, T_B é representada a tarefa consumidora e está alocada no endereço 1x0 e T_M é a tarefa maliciosa que está alocada no endereço 0x2. O cenário apresentado na Figura 28(a), consiste em T_A enviando mensagens para T_B, e uma tarefa maliciosa T_M é alocada no sistema. Na Figura 28(b) o ataque inicia com T_M enviando mensagem para o PE que está

executando T_A, o qual altera o endereço T_B. Depois de modificar a tabela de endereço, T_A envia mensagens para T_M, conforme mostra a Figura 28(c). Consequentemente, T_M pode extrair e modificar os dados sensíveis no fluxo de mensagens que está interceptando, pois T_M obtém acesso ao conteúdo da mensagem, podendo assim enviar uma mensagem diferente para T_B.

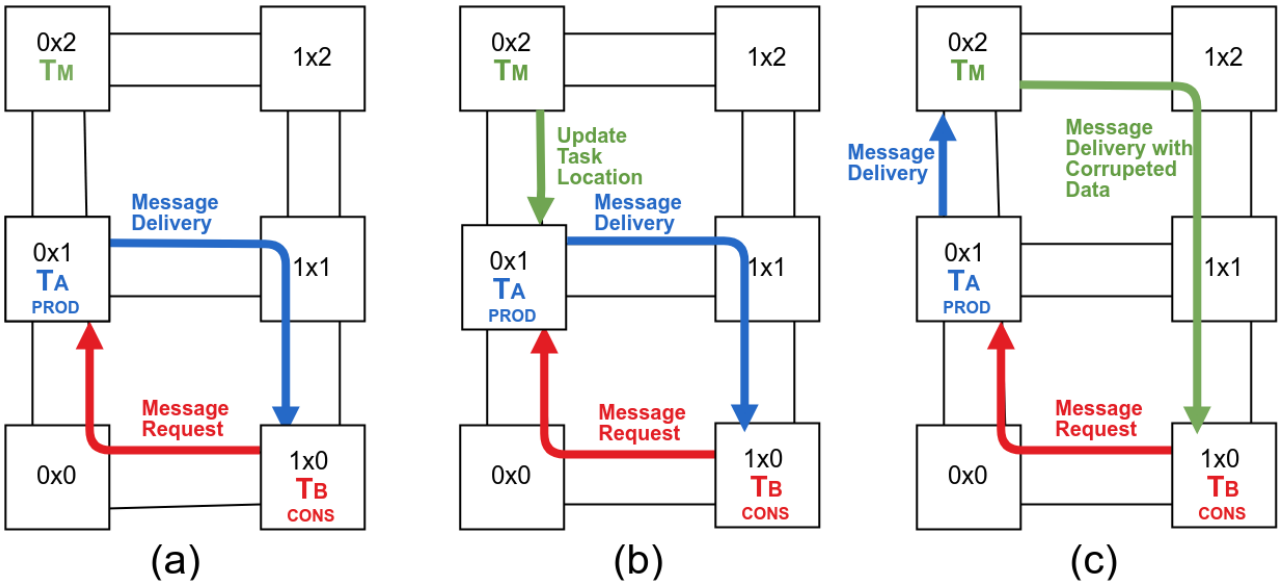


Figura 28: Exemplo do ataque *man-in-middle* implementado num MPSoC. (a) T_A se comunica com T_B. (b) T_M inicia o ataque. (c) T_M tem acesso o fluxo de comunicação (fonte: Autor).

A Figura 29 exemplifica o protocolo de comunicação executado pela tarefa maliciosa (T_M) realizar o ataque *man-in-the-middle*. Nesse ataque, assume-se que a tarefa maliciosa conhece a localização das tarefas produtora e consumidora, e também conhece a estrutura dos pacotes que trafegam na rede, podendo assim, interceptar pacotes, ler o cabeçalho destes, falsificar o endereço de um pacote e manipular as informações das mensagens.

As trocas de mensagens entre a tarefa consumidora (T_B), a tarefa produtora (T_A) e o invasor (T_M) são as seguintes:

1. MSG_REQ: T_B envia um pacote de requisição de mensagem para T_A.
2. MSG_DELY: T_A envia um pacote que contém a mensagem requisitada por T_B.
3. UPDATE_TASK_LOCATION: A tarefa maliciosa informa a T_A que T_B trocou de endereço, sendo esse novo endereço o de T_M (utilizando a função *UpdateID*).
4. MSG_REQ: T_B envia um pacote de requisição de mensagem para T_A.
5. MSG_DELY: T_A envia um pacote que contém mensagem requisitada por T_B, porém destinada para T_M.
6. MSG_DELY: T_M envia um pacote que contém a mensagem requisitada por T_B (utilizando a função *Send_Atk*).

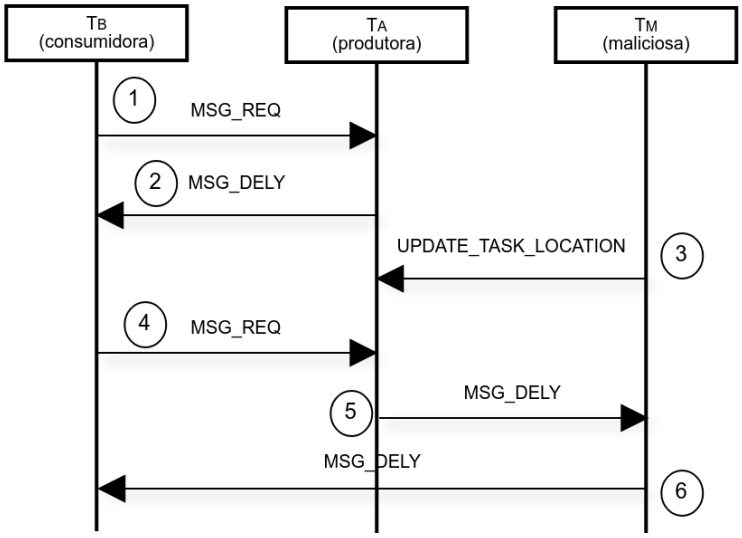


Figura 29: Fluxo de comunicação do ataque man-in-the-middle (fonte: Autor).

No processo descrito acima, o invasor (T_M) tem autonomia de modificar as mensagens, interceptá-las e ficar inativo sem que as tarefas suspeitam, assim pode-se estudar inúmeras formas de ataque e suas respectivas contramedidas.

Os resultados do ataque *man-in-the-middle* implementado são apresentados na Figura 30. A Figura 30(a) mostra o conteúdo original da mensagem enviada de T_A à T_B, como também é apresentado o conteúdo da mensagem recebida por T_M através da Figura 30(b). A Figura 30(c) mostra o conteúdo recebido por T_B. Nota-se que foi enviado 6 números quaisquer, porém o intruso ao interceptar a mensagem do produtor, modificou seu conteúdo e enviou para a tarefa consumidora.

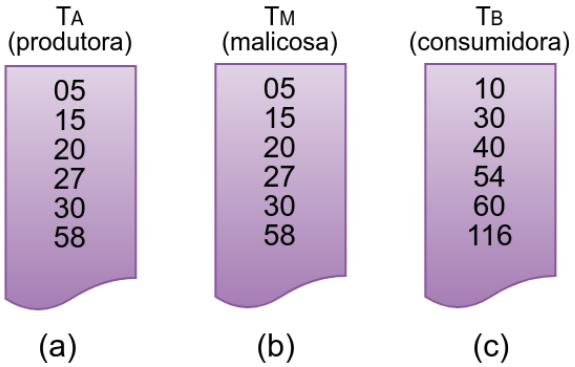


Figura 30: Conteúdo das mensagens das tarefas submetidas ao ataque man-in-the-middle (fonte: Autor).

Este ataque realizado na plataforma HeMPS não é particular a esta plataforma. Para a execução do ataque, é suficiente o protocolo de comunicação permitir a criação de pacotes no nível de aplicação. Quando se consideram periféricos, como memórias compartilhadas ou aceleradores de hardware, há a necessidade de uma API com esta característica. Da mesma forma, havendo aceleradores de hardware, estes podem forjar os pacotes e realizar os ataques.

5.2 Vulnerabilidades na infraestrutura de comunicação

Com intuito de criar um ataque do tipo DoS na rede de comunicação, foi criada uma tarefa maliciosa para congestionar a NoC, e assim degradar o desempenho do fluxo de dados. Uma fragilidade da Hermes é a ausência de mecanismos de proteção para inibir a transmissão de pacotes malformados. Um pacote malformado pode gerar problemas na NoC, como por exemplo, o bloqueio do tráfego de pacotes.

A Figura 31(a) apresenta um cenário proposto, onde as tarefas T_A (produtora) e T_B (consumidora) comunicam-se entre si através de troca de mensagens. Na Figura 31(b) a tarefa T_M (maliciosa) é alocada entre o fluxo de comunicação com o intuito de degradar a comunicação entre as tarefas. T_M envia um pacote no mesmo sentido do fluxo de comunicação de T_A e T_B com o intuito de degradar a comunicação entre as tarefas, para um PE inexistente no MPSoC, como por exemplo, o PE de endereço 4x1.

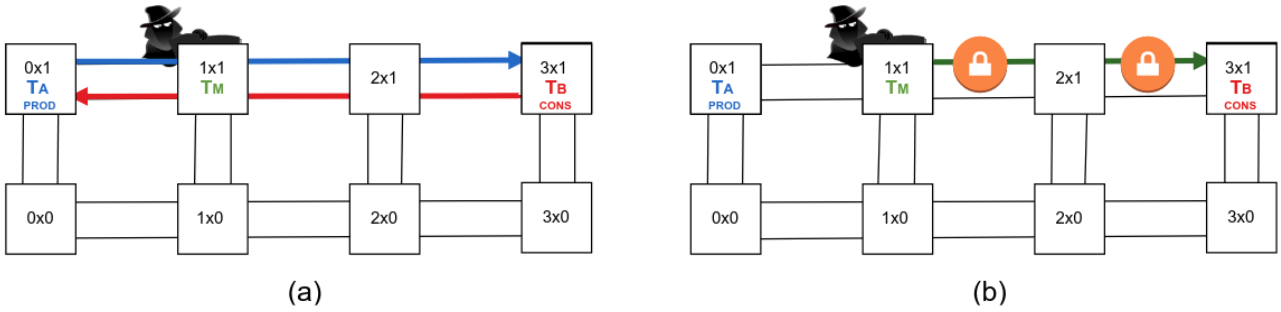


Figura 31: Cenário do ataque DoS na HeMPs. (a) Fluxo de comunicação entre as tarefas. (b) Intruso executando o ataque (fonte: Autor).

Após o envio do pacote malformado, houve um bloqueio na NoC no caminho onde o pacote malicioso trafegou. Consequentemente, este fato acarreta no bloqueio de novos pacotes que trafeguem pelo caminho que o pacote gerado por T_M.

Novamente este não é um ataque particular à rede Hermes. Pacotes malformados podem ocorrer em qualquer NoC. Exemplos de informações erradas em pacotes incluem: pacote definido com determinado tamanho e envio de um número diferente de *flits*; não transmissão de EOP (*end-of-packet*), caso a rede utilize esta sinalização; endereço inválido. Estes problemas normalmente causam bloqueio na rede, ocasionado um ataque DoS. Outro problema comum em NoCs é a geração de dados com uma alta taxa de injeção, causando atraso nas aplicações com restrições de tempo real.

5.3 Vulnerabilidades do SO ao receber pacotes inesperados

Este ataque visa desabilitar os recursos disponíveis no MPSoC, gerando um ataque DoS com o objetivo de degradar o desempenho do sistema. Nesse contexto, foi analisado o tratamento que o *kernel* ao realizar o recebimento de um pacote contendo um serviço não previsto pelo mesmo.

A Figura 32 apresenta o cenário do ataque DoS com o objetivo de desabilitar os recursos de um PE do sistema. A Figura 32(a) mostra uma tarefa maliciosa (T_M) enviando um pacote pela rede ao PE (1x1). A Figura 32(b) apresenta o PE desabilitado, ou seja, todos os recursos deste PE estão bloqueados devido ao pacote inesperado pela tarefa executada.

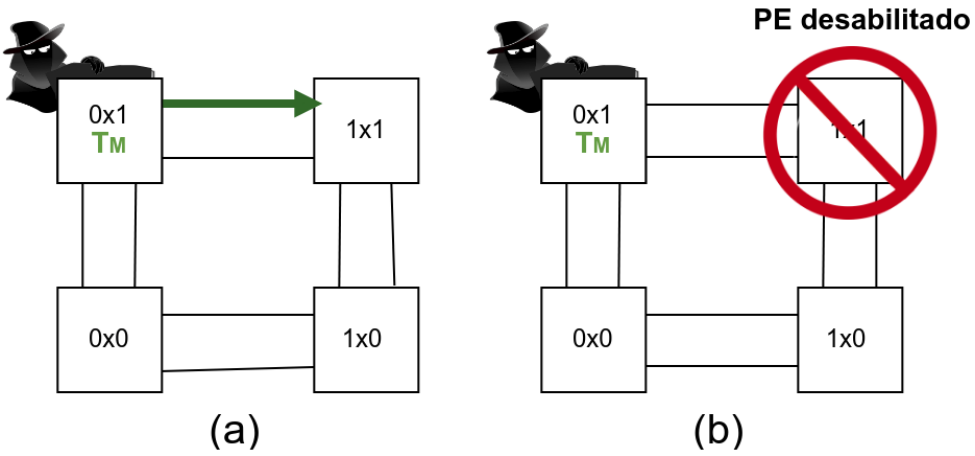


Figura 32: Ataque para desabilitar PEs do MPSoC (fonte: Autor).

Este ataque explora uma fragilidade do SO (*kernel*), A fragilidade é o não tratamento de pacotes com serviços não especificados no *kernel*. O *kernel* é interrompido para executar o processamento do pacote, porém não há um *driver* específico para tratá-lo. Assim, este pacote com serviço desconhecido fica preso na interface de rede, e na prática o processador é desabilitado. Este ataque pode ter consequências mais graves, pois o processador que executa o mapeamento pode tentar alocar uma tarefa neste processador, e ao tentar executar a mapeamento, o código objeto fica preso na NoC, travando todos os fluxos no caminho.

Para inibir este tipo de ataque, o SO deve prever o consumo de pacotes desconhecidos, e uma forma de descartá-los.

5.4 Vulnerabilidades no nível de acesso à memória (aplicação)

O objetivo deste ataque é demostrar que ataques em memória podem ocorrer em MPSoCs. Sabe-se que mecanismos de proteção de memória são técnicas comuns em sistemas multitarefas tanto no nível de hardware como em software [YAM14]. Entretanto alguns sistemas embarcados utilizam sistemas *bare-metal,* os quais raramente empregam qualquer sistema de segurança ou mecanismos de proteção [CLE17], necessitando assim, de mecanismos de proteção de baixo nível.

Neste ataque, foi estudado uma forma de uma tarefa maliciosa acessar a região da memória reservada para uma determinada tarefa sensível alocada no mesmo PE. Esse é um problema em sistemas que não possuem proteção de memória.

A arquitetura de referência adota uma organização da memória baseada em paginação, onde todas as páginas de memórias possuem o mesmo espaço de armazenamento, conforme apresentado na Figura 33. A primeira página é reservada para o *kernel*. As demais páginas são destinadas às tarefas de aplicação, onde cada nova tarefa alocada ocupa uma página de memória.

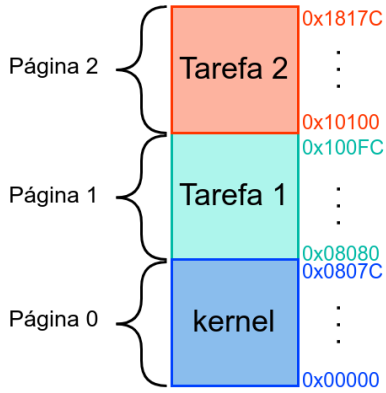


Figura 33: Estrutura da memória de 96KB utilizando paginação (fonte: Autor).

A Figura 34 apresenta o cenário de ataque na memória física. Para implementação do ataque, foi criada uma tarefa maliciosa (T_M) com a permissão de escrever, ler e modificar os dados específicos de uma tarefa alocada em seu PE.

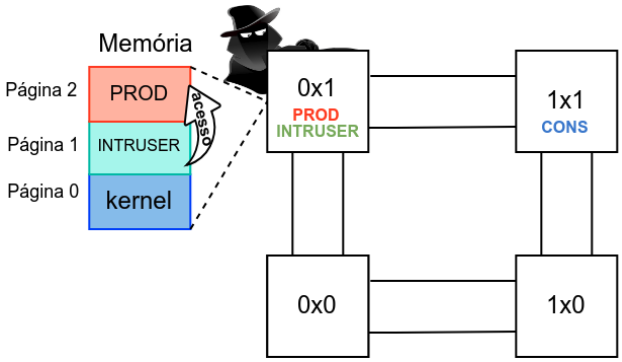


Figura 34: Cenário de ataque à memória (fonte: Autor).

A tarefa alocada foi instalada no repositório de tarefas, porém é importante ressaltar que na literatura, é encontrada referências de injeção da tarefa maliciosa no sistema de diferentes formas. Outra forma possível de acessar dados sensíveis através da memória é quando a tarefa sensível termina sua execução, caso ela não apague o conteúdo dos dados alocados em sua região de memória, uma outra tarefa poderá ser alocada nesta região, podendo assim, acessar os dados sensíveis disponíveis.

5.5 Vulnerabilidades no nível de Hardware

Esta subseção apresenta o trabalho realizado em parceria com os alunos Iaçanã Weber e Geaninne Lopes do Programa de Pós-Graduação em Ciência da Computação (PPGCC) da disciplina de Sistemas Integrados em Chip [WEB19] ministrada pelo Dr.

Fernando Gehm Moraes, com intuito de instalar um Hardware Trojan (HT) capaz de duplicar pacotes oriundos da NoC para seus respectivos alvos. O HT induz uma baixa sobrecarga de área (<10%), e não altera o desempenho do roteador.

Inspirados no trabalho de Ancajas et al. [ANC14], o HT é inserido no roteador, e mantém-se inativo até que seja despertado por uma aplicação maliciosa, com o objetivo de realizar a duplicação dos pacotes enviados pela porta local. As suposições realizadas para da execução do ataque compreendem: (*i*) a aplicação maliciosa é capaz de montar o pacote; (*ii*) a estrutura do pacote é conhecida pelo intruso; (*iii*) a localização do IP sensível; (*iv*) a localização das tarefas; (*v*) algoritmo de roteamento. As suposições (*i*) e (*ii*) são factíveis, havendo a API apresentada na Seção 4.4. Quanto à localização das tarefas, a tarefa maliciosa pode operar em modo “tentativa-e-erro” até localizar o seu alvo. E finalmente, o invasor pode assumir XY como modo de roteamento. Logo, as suposições, apesar de inicialmente parecerem muito fortes, o ataque é factível de ser realizado.

A Figura 35 apresenta o roteador infectado pelo HT. O roteador é composto pelos seguintes módulos: (*i*) Hermes Buffer; (*ii*) *Infected Hermes Buffer*; (*iii*) *Hardware Trojan*; (*iv*) *Infected Switch Control* e (*v*) *Infected Crossbar*. Na Figura 35 estão destacados, em vermelho, os sinais adicionados para dar suporte ao funcionamento do HT.

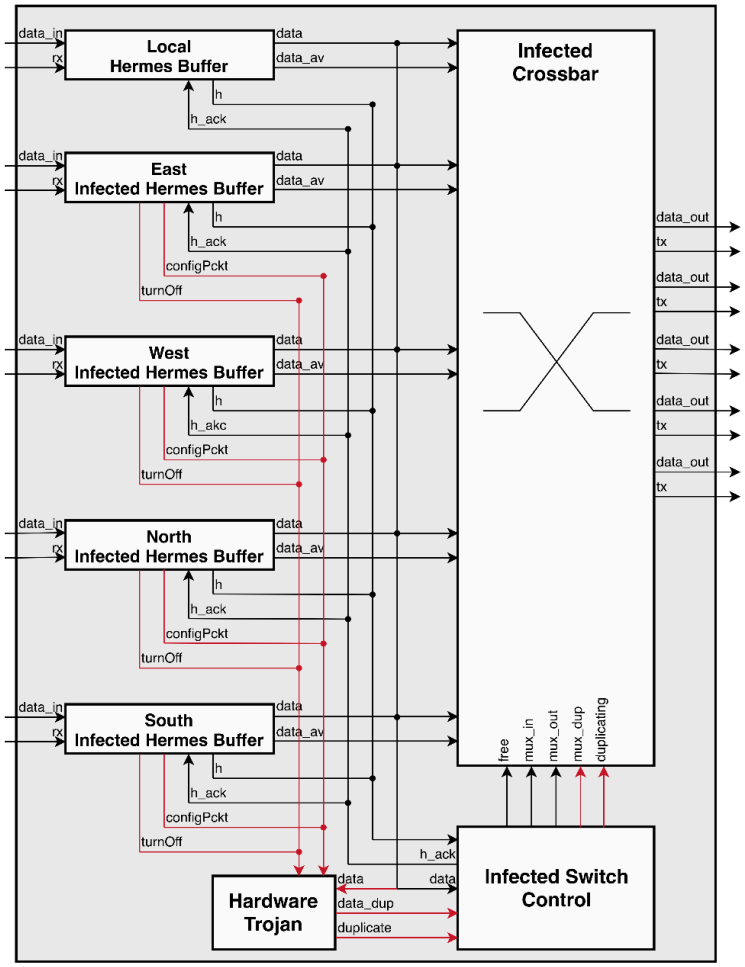


Figura 35: Arquitetura do roteador proposta para a instação dos HTs [WEB19].

A instalação dos HT é realizada na porta local dos roteadores. O HTs foi projetado para adquirir os dados da porta local, sendo imperceptível sua detecção no sistema. Os HTs possuem dois estados de funcionamento:

- Inativo: O HT se mantém em um estado latente, ou seja, não está realizando nenhum tipo de ação, não interferindo com a operação do roteador;
- Ativo: Uma vez desperto (através de um pacote específico de ativação), o HT passa a duplicar os dados provenientes da porta local, enviando-os para o destino correto e também para o endereço da tarefa maliciosa (este endereço é obtido pelo pacote de ativação).

Ao receber um pacote de ativação através da NoC, o HT passa para o estado ativo. A Figura 36 ilustra a estrutura do pacote de ativação do HT, composto por 3 flits. É importante destacar que este pacote difere do padrão adotado pela rede Hermes, cujo tamanho mínimo é de 13 flits. A transmissão deste pacote deve ser realizada pela tarefa maliciosa, sem a interferência do sistema operacional no processo.

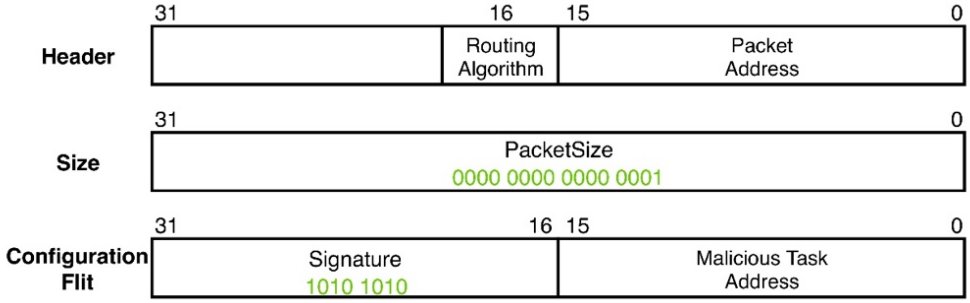


Figura 36: Estrutura do pacote de ativação dos HTs [WEB19].

O cabeçalho (*header*) do pacote, contém duas informações: o endereço do roteador alvo e o tipo do algoritmo de roteamento. O endereço consiste em meio flit contendo as coordenadas X e Y do roteador. O algoritmo de roteamento pode ser selecionado entre duas opções: o algoritmo XY, padrão da Hermes (indicado por '0' no 16º bit) e YX, adicionado durante a implementação do trojan (indicado por '1' no 16º bit).

O segundo *flit* carrega o tamanho do payload do pacote, o qual deve ser igual a 1. O terceiro flit é o *configuration flit*. Este flit carrega duas informações, a assinatura do HT e o endereço da tarefa maliciosa. A assinatura é um valor previamente definido, que o HT utiliza para filtrar pacotes de um flit que por ventura possam estar sendo transmitidos pela NoC, desta forma, reduzindo a chance de ativação indesejada do HT. Por fim, o último campo carrega o endereço para o qual o HT deve enviar os pacotes quanto ativado.

O HT no seu estado inativo aguarda o buffer identificar um pacote de configuração. Uma vez que algum dos buffers envia um sinal indicando que há um pacote de configuração, o HT captura o endereço do atacante, que está contido na parte baixa do terceiro flit do pacote de configuração, como apresentado na Figura 36.

A identificação do pacote de ativação, e o de desativação, ocorre como segue. O *Infected Hermes Buffer* detecta o tamanho do pacote igual a 1, e o *flit* de assinatura em 3 ciclos de relógio, enquanto o *Switch Control* necessita de 5 ciclos no mínimo para arbitrar e rotear. Assim, há tempo suficiente para que o aviso de infecção bloquear o aviso de chegada de dados na porta local.

A Figura 37 apresenta a arquitetura o controle de fluxo interno ao roteador, quando os pacotes estão sendo duplicados. O ponto inicial para compreender o processo são dos dois sinais de *ack* (*data_ack_dup* e *data_ack(4)*) que sinalizam que os dois roteadores destinos estão aptos a receberem dados. Dado que o *duplicating* está ativo, o *data_ack_local* para o buffer local é o resultado de um AND entre este dois sinais. Caso uma das portas não esteja livre, o crédito baixa para o buffer local (sinal *data_ack_local*), e ocorre a sinalização para crossbar que não há dado neste momento (sinal *data_av).*

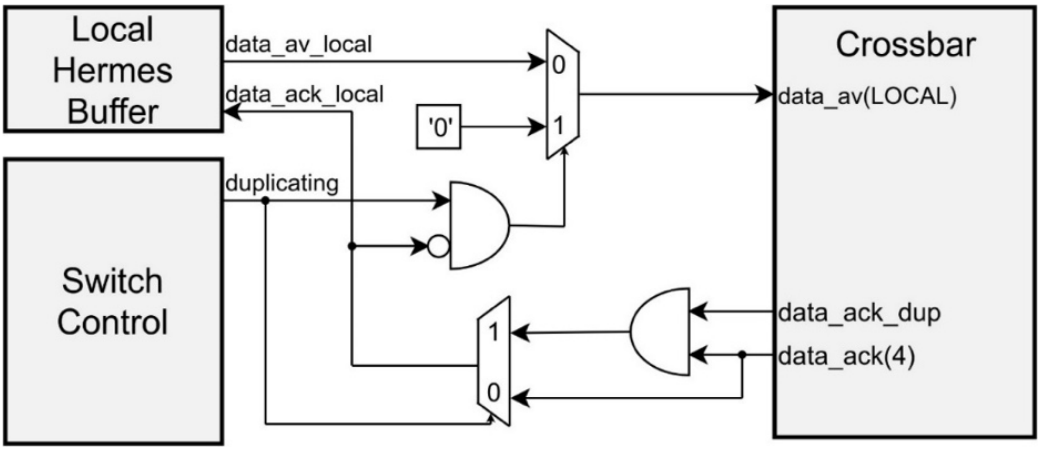
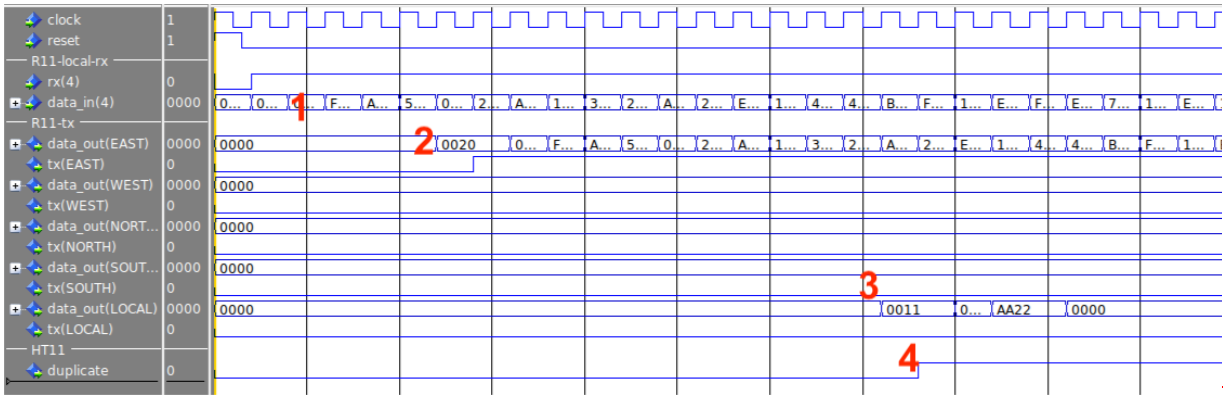


Figura 37: Arquitetura proposta do HT [WEB19].

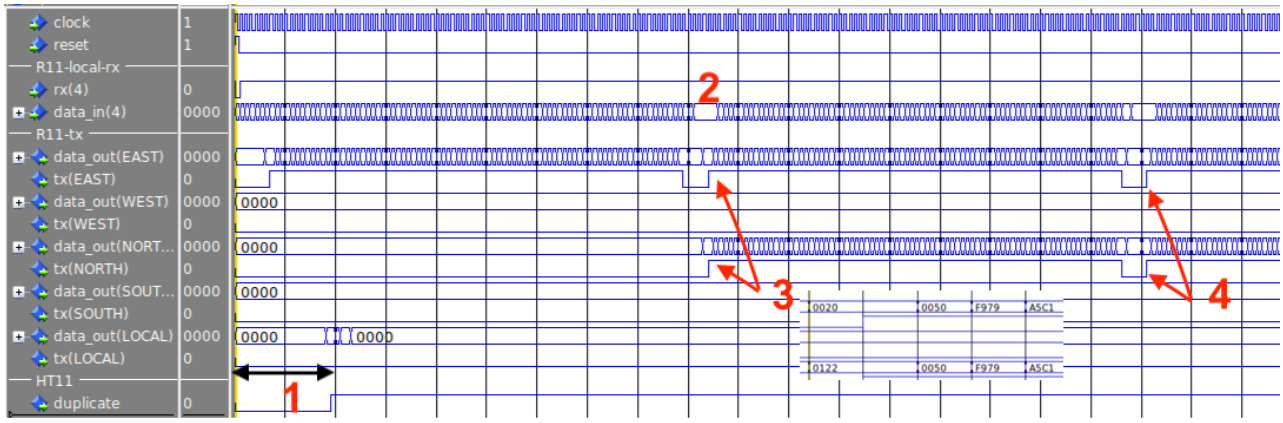
As Figuras 38, 39 e 40 apresentam o resultado da implementação do HT. A Figura 38 ilustra a ativação do HT, a Figura 39 o ataque ocorrendo, e a Figura 40 apresenta a execução do ataque sendo realizado com sucesso.

Este ataque é de difícil detecção, pois os processadores não têm ciência que seus dados estão sendo duplicados. Neste caso, a melhor contramedida é a criptografia, dado que a tarefa maliciosa não possui a chave criptográfica para interpretar os dados. No Capítulo 6 apresentamos uma forma segura de transmissão da chave criptográfica, de forma a garantir que a mesma seja transmitida por um canal seguro. Ataques mais sofisticados, como SCA (*side channel attack*), podem ainda comprometer a segurança do sistema, mas estes estão forma do escopo do presente trabalho.



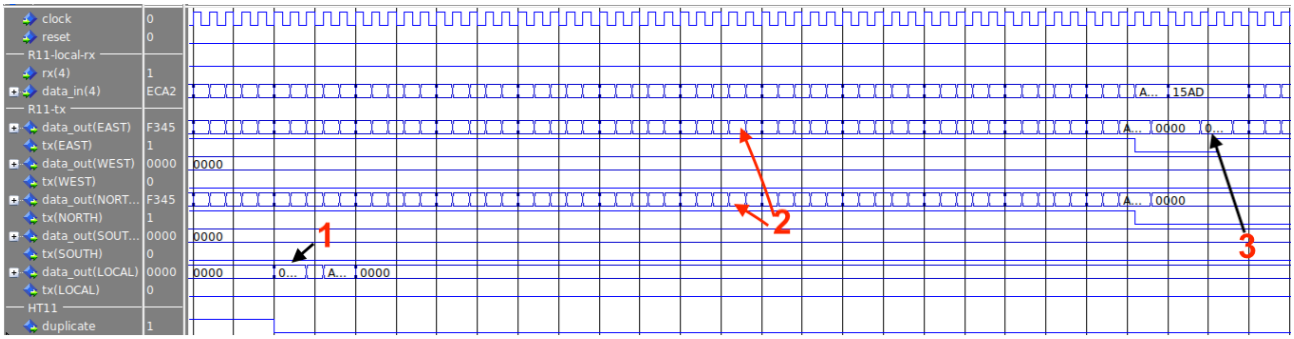
1. Pacote sendo enviado pelo IP para a NOC através da porta de entrada LOCAL;
2. Pacote sendo transmitido pela porta Leste;
3. Aqui observamos o pacote de ativação do trojan sendo recebido. Perceba que por ser endereçado a este roteador, a NoC tenta transmitir o pacote para o IP, através da porta de saída LOCAL. Porém, observe que o tx(LOCAL) mantem- se em ZERO enquanto os *flits* do pacote de configuração são recebidos. Isso é a "*camuflagem*" que o HT faz para evitar a propagação do pacote para o nível do IP. Outro fato interessante é observar que o endereço da aplicação maliciosa aparece na parte baixa do 3º flit, ou seja, "2x2"
4. Sinal *duplicate* sobe, indicando início do ataque. Logo depois da recepção do pacote de configuração o HT passa ao estado de atividade, indicando ao *Switch Control* que os pacotes devem ser duplicados.

Figura 38: Simulação da ativação do HT [WEB19].



1. Simulação da Figura 38 com ativação do HT, onde o sinal *duplicate* sobe;
2. Final da transmissão do primeiro pacote;
3. Pacote sendo direcionado pela porta correta (LESTE) e duplicado para a porte NORTE. No destaque vemos que o pacote é enviado para o endereço correto (0x0020), e para o endereço da tarefa maliciosa (0x0022), com roteamento YX;
4. Pacote sendo direcionado pela porta correta (LESTE) e duplicado para a porte NORTE.

Figura 39: Simulação com o ataque via HT ocorrendo [WEB19].



1. Pacote de desativação do HT recebido, notar que o sinal *duplicate* desce;
2. Pacote é duplicado é enviado até o final da transmissão do mesmo;
3. O pacote volta a ser transmitido apenas na porta LESTE, dado que o HT foi desativado.

Figura 40: Término do ataque com HT [WEB19].

5.6 Considerações Finais

Este Capítulo apresentou um conjunto de vulnerabilidades da plataforma de referência. Para verificar a exequibilidade de possíveis ataques ao sistema, foram realizados ataques ao mesmo. Dentre as vulnerabilidades estudadas destaca-se:

a) API de Comunicação;

b) Sistema operacional;

c) Infraestrutura de rede;

d) Acesso à memória;

e) Instalação de HTs.

É importante destacar que este conjunto de vulnerabilidades encontradas na plataforma de referência é o suficiente para que possa desenvolver contramedidas aplicadas em MPSoCs semelhantes.

6 SECURED ENVIROMENT ARCHITECTURE (SEA)

A arquitetura SEA tem como base de sua construção a arquitetura HeMPS (*Hermes Multiprocessor System*) descrita no Capítulo 4. Destaca-se que esta arquitetura iniciou como objeto da Dissertação de Mestrado de Bruno Scherer Oliveira, e que o Autor desta Dissertação contribuiu no desenvolvimento da mesma, adicionando mecanismos de segurança. É importante destacar que o *firewall* originalmente proposto não possuía conexão com a arquitetura HeMPS.

A Figura 41 apresenta a SEA com os módulos de segurança, onde:

- *F*: representa os firewalls;
- PE': representa os elementos de processamento, tais como: DMNI, processador e memória;
- *HNoC*: é a rede dedicada para transmitir informações relativas à segurança das aplicações;
- *R*: roteadores da NoC Hermes [MOR04];
- *MP* (*Manager Processor*): elemento de processamento responsável por gerenciar o MPSoC.

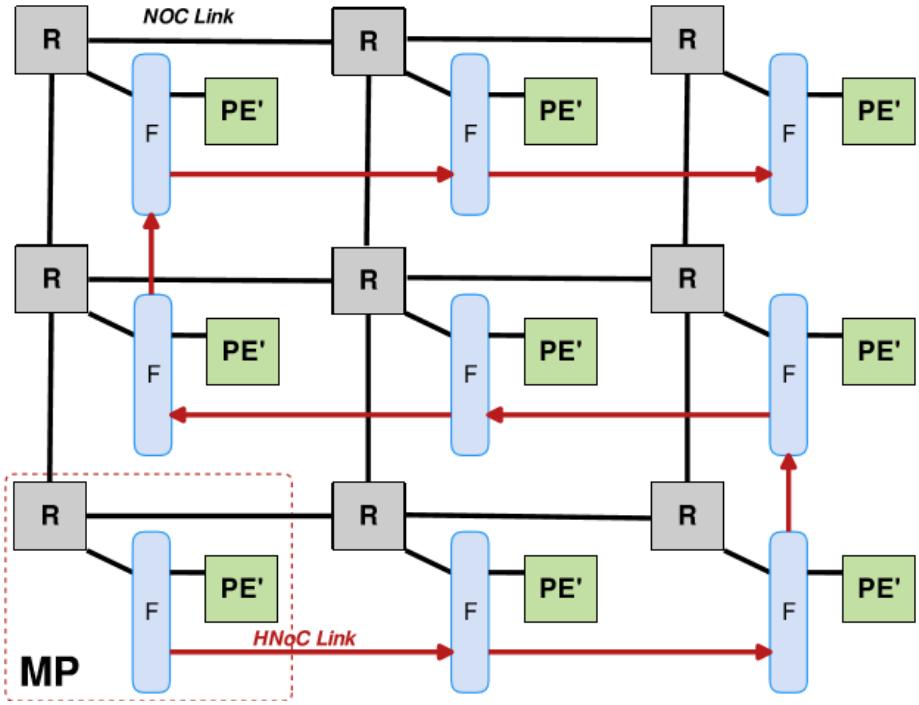


Figura 41: Arquitetura SEA, com detalhamento dos módulos de segurança [OLI18b].

O objetivo principal no desenvolvimento da arquitetura SEA é inserir os módulos de segurança preservando as características originais da plataforma HeMPS, ou seja, agregando segurança ao sistema de forma não invasiva. Desta forma, alterações futuras na plataforma HeMPS podem ser beneficiadas com esta arquitetura. Os *firewalls* foram

posicionados entre a porta local do roteador e o PE', e a chave de criptografia é transmitida aos *firewalls* através da *HNoC*. O módulo de criptografia AES (*Advanced Encryption Standard*) é conectado ao *firewall*. Apenas o MP possui acesso de escrita nesta rede, impedindo que aplicações maliciosas interfiram no processo de configuração dos *firewalls*. Apesar de não alterarmos o hardware da plataforma HeMPS, é necessário o desenvolvimento de diversos *drivers* no nível de *kernel* para o gerenciamento das aplicações com requisitos de segurança.

A arquitetura SEA foi construída baseada na utilização de *firewalls* e criptografia, com o intuito de proteger os dados de comunicação que trafegam na NoC contra ataques realizados por processos maliciosos. A rede que interconecta os *firewalls*, HNoC, é constituída por um caminho Hamiltoniano (linha vermelha na Figura 41), sendo responsável pelo envio da chave de criptografia para cada bloco AES e a identificação de cada aplicação alocada no sistema.

A plataforma SEA foi inspirada pelo trabalho relatado em [FER15]. Este trabalho teve por principal objetivo proteger o sistema contra invasores utilizando apenas *firewalls*. Os *firewalls* garantem proteção contra ataques, filtrando dados de saída e entrada dos PEs, conforme regras previamente carregadas nos mesmos. Posteriormente, houve um acréscimo de mecanismos de segurança, como por exemplo, a instalação do módulo de criptografia AES.

O MP é responsável por gerar as chaves de criptografia randomicamente[2], como também enviar a chave através da *HNoC*, para cada *firewall*. As chaves são únicas por aplicação, ou seja, todas as tarefas de uma dada aplicação compartilham a mesma chave. Quando o MP mapear as aplicações na HeMPS, ele também enviará a chave de criptografia para os *firewalls* de acordo com o mapeamento das tarefas. Esta "conexão" entre o mapeamento de tarefas, a geração das chaves e a distribuição das chaves aos PEs é uma contribuição do Autor à arquitetura SEA.

Observar que o MP não executa tarefas de aplicações, apenas gerenciamento do sistema. Este PE é o único que possui acesso de escrita no caminho Hamiltoniano. Assim, não é possível às aplicações terem acesso à *HNoC*, impedindo que processos maliciosos realizem ataques visando a obtenção de dados sensíveis que trafegam nesta rede, os quais correspondem às chaves de criptografia e regras de acesso.

As próximas seções apresentam as características de cada módulo e contramedidas instaladas na plataforma SEA. Também são abordadas técnicas de hardware e software presentes na SEA para inibir e mitigar as vulnerabilidades apresentadas no Capítulo 5.

[2] Não está no escopo deste trabalho a geração de uma chave forte de criptografia. Um método simples de geração foi adotado, baseado em software.

6.1 Firewall

Os *firewalls* são posicionados entre o roteador e a interface de rede (DMNI [RUA16]). A Figura 42 apresenta as conexões do *firewall*. Os *firewalls* propostos em [FER15] receberam as seguintes modificações no presente trabalho:

- *Interconexão à rede HNoC* – automatizou-se o processo de instanciação dos *firewalls* na geração da plataforma SEA através de arquivo em formato YAML (formato de codificação de dados[3]) que define o tamanho da plataforma, se a mesma terá ou não os *firewalls*, e diversos outros parâmetros de hardware. A rede *HNoC* requer apenas dois registradores internos por *firewall*, *reg_IN* e *reg_Out* (Figura 42). Quando um pacote é destinado ao endereço do *firewall*, o processo de recepção de dados inicia, com armazenamento do *App ID* e *App Key* (processo descrito na Seção 86.2.2).
- Bloco AES (*crypto core* na Figura) – o bloco de criptografia é conectado ao *firewall*, não fazendo parte do mesmo. Com isto, em trabalhos futuros, pode-se utilizar outros módulos de criptografia que consumam uma menor área de silício (*lightweight cryptography*), como por exemplo, o módulo de criptografia SIMON [BEA15]. A chave do módulo AES é armazenada no *firewall*, durante a configuração do mesmo.

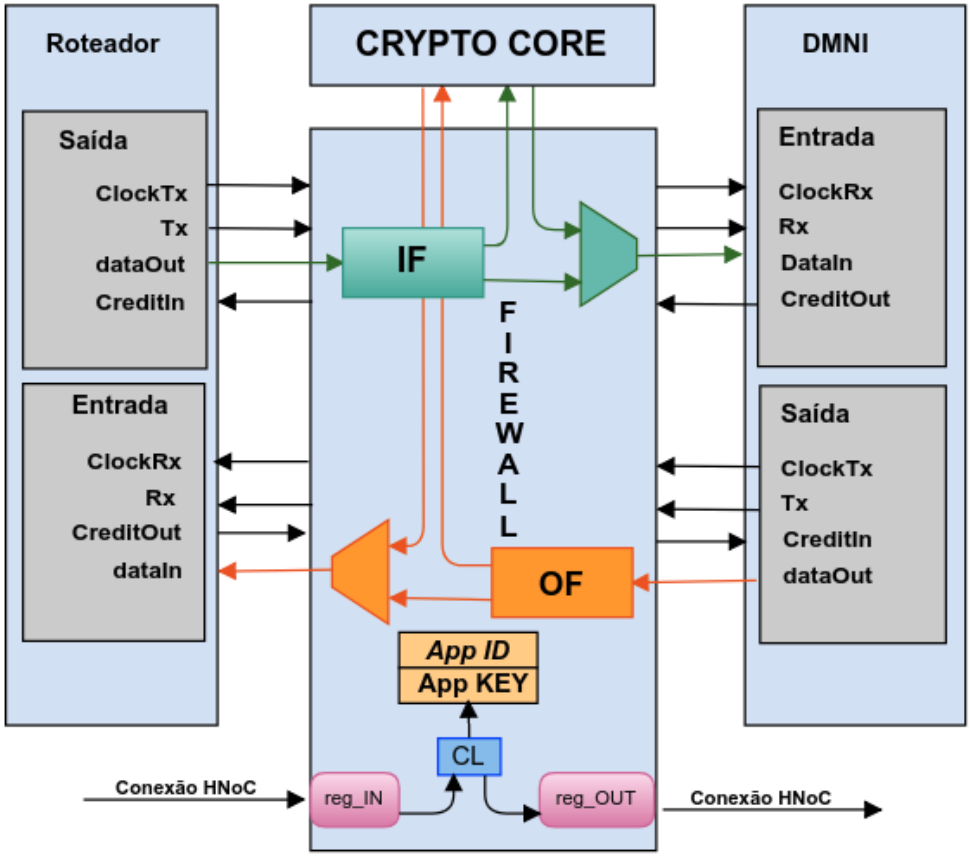


Figura 42: Representação das conexões do *firewall* e principais módulos internos (fonte: Autor).

[3] https://pt.wikipedia.org/wiki/YAML

- Filtros de entrada (IF) e saída (OF) – descritos na Seção 6.5.
- Árbitro (não representado na Figura) – o *firewall* possui internamente um árbitro, possibilitando a comunicação *half-duplex*, ou seja, pode-se receber e enviar de forma intercalada fluxos criptografados. Caso os dados não sejam criptografados, pode-se receber e enviar dados de forma simultânea, obtendo-se uma comunicação *full duplex*. A comunicação também é *full duplex* quando existe um fluxo criptografado e outro não criptografado que trafega através do *firewall*. O árbitro é necessário para evitar *deadlocks* no nível de aplicação. Este problema ocorre quando é necessário enviar e a receber dados por tarefas distintas mapeadas no mesmo PE, e não havendo o árbitro uma das tarefas poderia ficar bloqueada indefinidamente.

O *firewall* é projetado para ter reduzida área de silício. Para atender a este requisito de projeto os fluxos de entrada e saída não são internamente armazenados. Utiliza-se apenas registradores para armazenamento de chaves e para a comunicação entre os *firewalls*.

6.2 Rede Dedicada à Segurança (HNoC)

Se a transmissão dos dados sensíveis utilizasse a NoC de dados, a segurança do sistema estaria vulnerável, pois processos maliciosos poderiam acessar estes dados. Assim, para fornecer segurança a dados sensíveis, este trabalho propõe a adoção de uma NoC simples e dedicada para enviar dados sensíveis aos PEs. A adoção de uma NoC dedicada isola dados confidenciais de dados oriundos de aplicações e dados de controle do MP. A rede dedicada, denominada *HNoC*, é um caminho Hamiltoniano que percorre todos os PEs. A rede dedicada adota um canal de transmissão de dados de 32 bits. É importante ressaltar que o sinal FW_CONFIG é responsável por distinguir os pacotes de configuração e de dados que trafegam na *HNOC*, então considera-se que a rede utiliza 33 bits para transmitir um pacote.

As razões para adotar essa topologia incluem: (*i*) pequena área de silício; 2 portas em vez de 5 portas de uma rede de topologia *mesh*; (*ii*) a quantidade de dados a ser transmitido é pequena, justificando o caminho sequencial.

A *HNoC* conecta todos *firewalls* entre si, não repetindo nenhuma conexão entre eles. A Figura 43 ilustra a interface externa no *firewall* relativa à *HNoC* (parte inferior da Figura 42). Os *firewalls* utilizam um protocolo de comunicação *handshake* de quatro fases, utilizando os sinais *send* e *ack.* O barramento FW_DATA corresponde à informação que deve ser tratada pelo *firewall.*

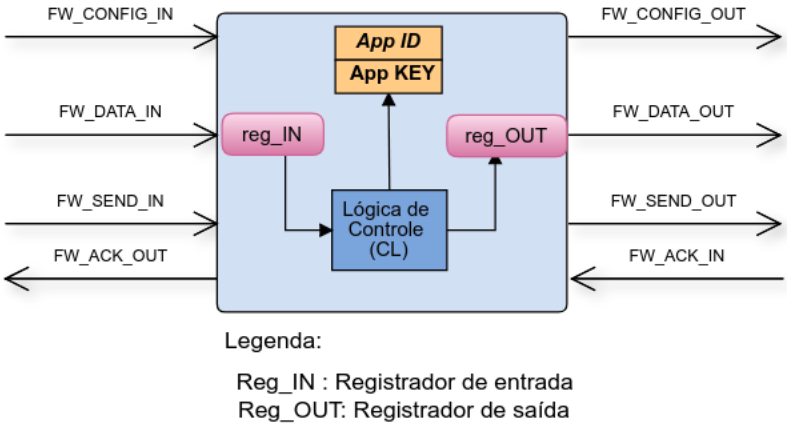


Figura 43: Sinais utilizados nos *firewalls* para a realização do caminho Hamiltoniano (fonte: Autor).

6.2.1 Estrutura dos pacotes utilizados nas Redes Hermes e HNoC

Na plataforma SEA há duas redes onde trafegam pacotes, uma da dados e controle – Hermes, e outra para envio de dados sensíveis – HNoC.

Os pacotes que trafegam na rede Hermes possuem o formato apresentado na Figura 44. É importante destacar que este formato corresponde ao nível de aplicação. Os *flits* de cabeçalho identificam cada pacote, de acordo com sua função. No nível de rede há dois *flits* de cabeçalho (destino e tamanho do pacote) e *payload*. Cada *flit* possui tamanho de 32 bits.

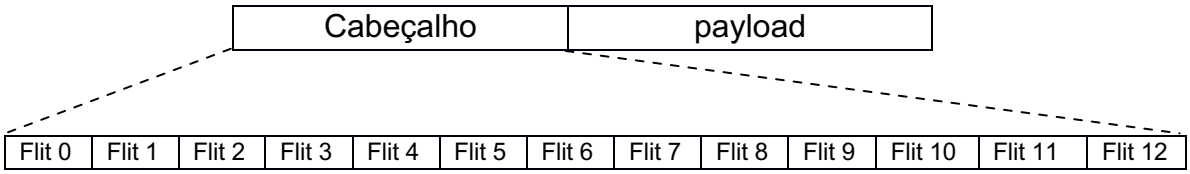


Figura 44: Composição de um pacote de dados na NoC, no nível de aplicação (fonte: Autor).

O cabeçalho contém informações para que o pacote consiga trafegar na rede e ser consumido pelo sistema operacional (*kernel*), sendo constituído por 13 *flits*. O primeiro e o segundo *flits* correspondem ao destino e o tamanho da carga de informação, sendo utilizando no nível de rede. Um extenso conjunto de funções são suportadas na plataforma HeMPS. O detalhamento dos campos e cada função pode ser consultado em: http://www.inf.pucrs.br/hemps/docs/HeMPS_Messages.pdf. O *payload* é opcional, e corresponde por exemplo a códigos-objeto de tarefas, dados de comunicação entre tarefas (*send – receive*), ou informações de controle, como monitoramento dos SPs (e.g., número de instruções executadas para fins de cálculo de energia consumida) [MAR18].

A HNoC distingue pacotes em dois tipos: pacotes de configuração e pacotes de dados. Os pacotes de dados são caracterizados por conter partes da chave de criptografia e a identificação da aplicação, e pacotes de configuração contêm o endereço do PE. Na HNoC os pacotes são limitados a 32 bits, não havendo distinção entre pacote e *flit*. Dois

tipos de pacotes são utilizados:

- Pacote de configuração, contendo apenas o endereço do *firewall* a ser configurado e a identificação da aplicação que a tarefa a ser alocada pertence, conforme apresenta a Figura 45. A função deste pacote é preparar os *firewalls* para receberem na sequência *n* pacotes de *payload*. Este procedimento é necessário para habilitar o envio de dados em *multicast* e *payloads* maiores que o suportado pelo tamanho do pacote - 32 bits. Por exemplo, para o envio de uma chave de 128 bits são enviados 4 pacotes de dados. É importante ressaltar que foram reservados 4 bits (identificado como header na Figura 45) para futuramente ser implementado um protocolo de comunicação mais complexo.

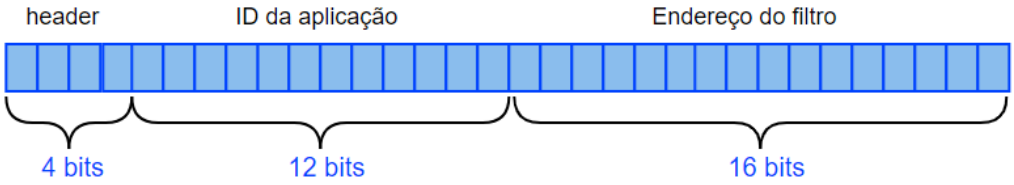


Figura 45: Estrutura do pacote de configuração de 32 bits.

- Pacotes de dados: todos os *firewalls* aptos a receberem dados consomem o pacote de dados.

Assim, o processo de transmissão de um volume de dados com *n* palavras de 32 bits para *k firewalls* ocorre como segue. Inicialmente, *k* pacotes de configuração são enviados aos *firewalls*. Na sequência, envia-se *n* pacotes de dados aos *firewalls* previamente configurados, através da HNoC. Considere que se deseje enviar 8 pacotes de dados a 5 firewalls. No método proposto, com *multicast*, são necessários 13 pacotes. Se a transmissão fosse *unicast*, seriam necessários 40 pacotes (5 destinos devendo receber cada um os 8 pacotes de dados). Assim, este método de transmissão reduz o tráfego na HNoC.

6.2.2 MODO DE OPERAÇÃO DO HNOC

A Figura 46 ilustra o modo de operação da HNoC. A Figura 46(a) ilustra um MPSoC de dimensão 2x2, com o MP no endereço 0x0, e duas tarefas, *PROD e CONS*, mapeadas nos endereços 0x1 e 1x0, respectivamente.

O processo de envio das chaves é ilustrado na Figura 46(b), constituído pelas seguintes etapas:

1 O MP envia o primeiro pacote de configuração para a tarefa alocada mais próxima ao MP - *CONS*.
2 Ao receber o pacote de configuração, o *firewall* armazena o pacote.
3 O GMP envia o segundo pacote de configuração para a tarefa *PROD*.
4 Finalização do envio dos pacotes de configuração.

5 O PE gerente gera a chave de criptografia e envia quatro pacotes de dados contendo o conteúdo da chave pelo caminho Hamiltoniano. É importante ressaltar que o pacote de dados irá trafegar todos os firewalls. A Figura ilustra somente o envio do primeiro pacote de dados.

6 O *firewall* da tarefa *CONS* armazena o conteúdo da chave e repassa para o *firewall* seguinte.

7 Como o *firewall* 1x1 não recebeu um pacote de configuração com o seu endereço, o *firewall* somente repassa o pacote de dados para o *firewall* seguinte.

8 O *firewall* 0x1 recebe os pacotes de dados, dado que recebeu um pacote de configuração.

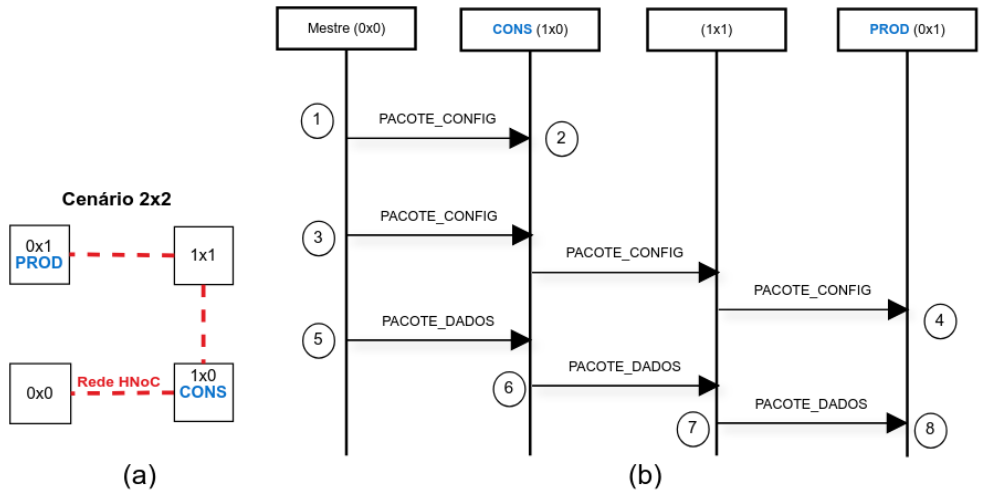


Figura 46: Protocolo de envio de chaves de criptografia para os *firewalls*. (a) Cenário de um MPSoC 2x2. (b) Protocolo de comunicação (fonte: Autor).

De acordo com a máquina de estados implementada para a transmissão de dados da HNoC, a latência de propagação de cada pacote (configuração ou dados) é obtido pela aplicação da Equação 1.

$$latência\ do\ pacote_{HNoC} = (r_{bypass} * h_{bypass}) + (r_{conf} * h_{conf}) \quad (1)$$

Onde: r_{bypass} é número de ciclos de relógio para transmitir dados pelo firewall sem armazenamento (3 ciclos); r_{conf} é o número de ciclos de relógio para armazenar o pacote e transmiti-lo ao *firewall* seguinte (5 ciclos); e h é o número total de hops que o pacote deve percorrer, correspondendo a $h_{bypass} + h_{conf}$.

6.3 Geração de chaves de criptografia

O MP é responsável por gerar uma chave de criptografia exclusiva para cada aplicação a ser mapeada no MPSoC. A geração desta chave é executada em software, no *kernel*. Na geração, a chave é dividida em 16 partes, onde cada parte é gerada através de uma operação XOR com a identificação da aplicação (atribui-se um número específico para cada aplicação) e o tempo atual (dado em *ticks*) em que ocorre a operação, garantindo

assim a exclusividade da chave criptográfica para a aplicação. Processos mais complexos podem ser utilizados, como a utilização de um hardware gerador de números randômicos ou PUFs (*Physical Unclonable Function*s). Dado que o objetivo é o desenvolvimento da infraestrutura completa de segurança, optou-se por este processo em software.

A chave de criptografia é constituída por 128 bits. Esta chave é transmitida pela HNoC em quatro blocos de 32 bits, devido a limitação do tamanho do *payload* (32 bits).

Para a chave de descriptografia, foi utilizado o bloco AES para gerar tal chave. Logo, a geração da chave é local e não é transmitida pelo caminho Hamiltoniano[4]. Após receber a última parte da chave de criptografia, o *firewall* solicita ao bloco AES a chave de descriptografia. Esta chave é gerada na décima rodada (do inglês, *round*) do processo de criptografia do AES. Ao chegar nessa rodado, uma flag interna do AES retorna a chave de descriptografia para o *firewall* que a armazena em seus registrados internos.

6.4 AES Core

O *core* AES utilizado é baseado no projeto disponibilizado no *site opencores* [HEM14], que é uma versão do FIPS-197 implementada no modo ECB (*Electronic Codebook*). O Autor do *core* menciona que o modo ECB apresenta risco em relação aos ataques com repetição de mensagens. A arquitetura opera com dois blocos de 64 bits, carregados em ciclos de relógio consecutivos. A Figura 47 ilustra a temporização para que o *core* AES criptografe um bloco de 128 bits. Inicialmente, o sinal *load* inicializa a chave e os dados. Uma vez que os dados são carregados, o sinal *start* é ativado, e depois de 13 ciclos de relógio, o sinal *done* é ativado e os dados (criptografados/descriptografados) estão disponíveis no barramento de saída.

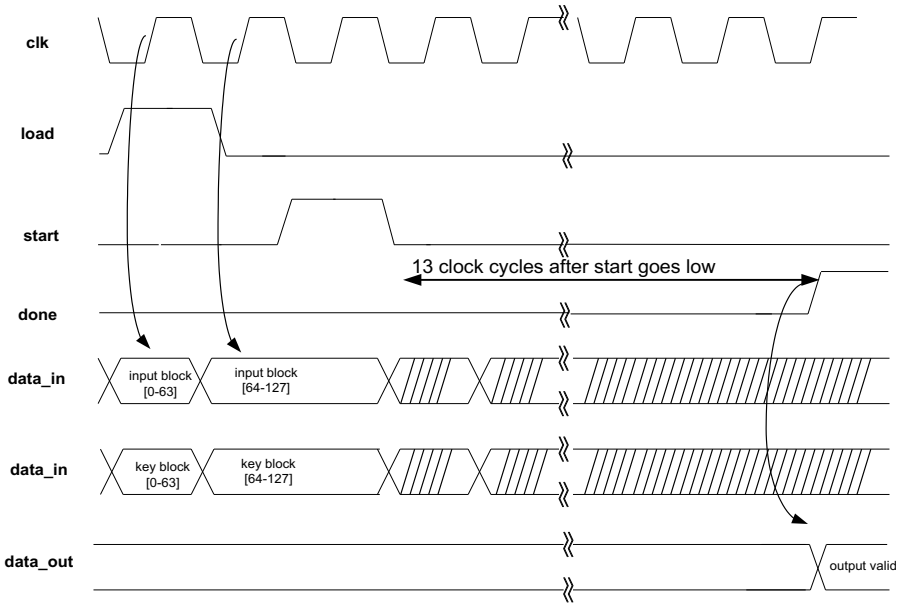


Figura 47: Diagrama de tempos do módulo AES [HEM14].

[4] O AES é um algoritmo simétrico de criptografia. A chave para descriptografar um dado é obtido a partir da chave utilizada para criptografar o dado.

6.5 Filtro do *firewall*

Este módulo faz parte do *firewall* e tem por objetivo filtrar pacotes oriundo da rede de comunicação. Esta técnica é semelhante ao trabalho apresentado em [AZA18]. Os pacotes que não fazem parte do conjunto de tarefas de uma determinada aplicação são descartados, evitando assim ataques proveniente de tarefas maliciosas.

A Figura 42 apresenta o *firewall* com os filtros. Os dados recebidos através da HNoC são armazenados em um registrador denominado *reg_IN*, e a lógica de controle (CL) avalia o conteúdo dos dados. Para uma mensagem de configuração, CL avalia o endereço do *firewall* com o conteúdo da mensagem. Se corresponderem, significa que as mensagens seguintes contêm dados a serem armazenados nos registradores internos (*App_ID* e *App_key*). A implementação atual do filtro recebe cinco mensagens de *payload*: 4 para o conteúdo da chave de aplicação de 128 bits (*App_key*) e uma para o identificador de aplicação (App_ID). Ambos os valores, *App_ID* e *App_key*, são exclusivos para cada aplicação. Quando uma nova aplicação é admitida no sistema, o MP gera novos valores, e os envia pela rede segura.

Os retângulos IF e OF na Figura 42 correspondem, respectivamente, aos filtros de entrada e de saída. Qualquer aplicação, usando as APIs de comunicação orientada a serviço ou *raw*, deve inserir o *App_ID* no 4º flit do cabeçalho do pacote (Figura 44).

O OF evita que uma tarefa tente forjar o *App_ID* de outra aplicação para tentar executar um ataque (como o anteriormente descrito *man-in-the-middle*). Nessa situação, o OF descarta o pacote, uma vez que o pacote *App_ID* não corresponde ao conteúdo do registador *App_ID* e o fluxo malicioso não entra no NoC. É importante ressaltar que uma tarefa que injete um pacote malicioso na rede não percebe se o pacote foi descartado. Esta tarefa continua a agir conforme o ataque está programado.

O IF descarta um pacote se este for de outra aplicação ou oriundo de um dispositivo de E/S (entrada/saída) através da incoerência dos *App_ID* nos pacotes. Um pacote oriundo de uma aplicação diferente do *App_ID* armazenado pode corresponder a um ataque DoS, onde uma determinada aplicação teria o objetivo de sobrecarregar a CPU. Um pacote oriundo de um dispositivo de E/S pode vir a forjar o *App_ID*, passando através do filtro IF. Entretanto, o mecanismo de E/S previsto na plataforma HeMPS é resiliente a este tipo de ataque por utilizar um mecanismo de comunicação mestre-escravo. Toda comunicação de E/S parte do processador. Logo, não é possível a um fluxo de E/S extrair dados da memória [CAI19]. Este pacote de ataque é detectado e descartado no nível de kernel.

6.6 Restrição do mapeamento de tarefas

A restrição do mapeamento de tarefas visa evitar que tarefas maliciosas se instalem num mesmo processador, evitando assim que uma tarefa maliciosa acesse uma região de

memória destinada a outra tarefa ou realizar um ataque DoS na tarefa sensível. Portanto esta técnica de isolamento espacial das aplicações evita o compartilhamento da computação das tarefas no nível do processador.

O objetivo desta técnica é restringir o mapeamento de tarefas de modo que as tarefas só possam compartilhar o mesmo PE se pertencerem à mesma aplicação. A Figura 48(a-b) mostra diferentes cenários para mapear duas aplicações e a Figura 48(c) apresenta o grafo de tarefas para cada aplicação. Na Figura 48(a) observamos duas aplicações distintas (app1 e app2) compartilhando o mesmo SP(0x1). Nesse cenário, a tarefa maliciosa (T_M) pode acessar a memória ou acessar o roteador, para que ele possa roubar informações e realizar um ataque. Na Figura 48(b), o app1 é mapeado sem compartilhar os processadores com outros aplicativos, garantindo o isolamento espacial da aplicação.

A heurística do mapeamento ocorre no MP. A medida implementada bloqueia a alocação de aplicações distintas no mesmo PE, evitando assim o acesso direto à memória por tarefas maliciosas. Para tanto foi alterado o *kernel* mestre, modificando-se o processo de mapeamento de tarefas. Criou-se um vetor, atribuindo-se o índice da aplicação quando o processador receber uma dada tarefa da aplicação. O algoritmo de mapeamento verifica se o processador está livre, caso contrário, ele verifica se a aplicação da tarefa a ser mapeada é diferente da aplicação atual. Se for diferente, o mapeamento escolhe outro PE, e se for igual ele pode mapear neste mesmo processador. Esta contramedida protege a integridade e a confidencialidade no nível do PE.

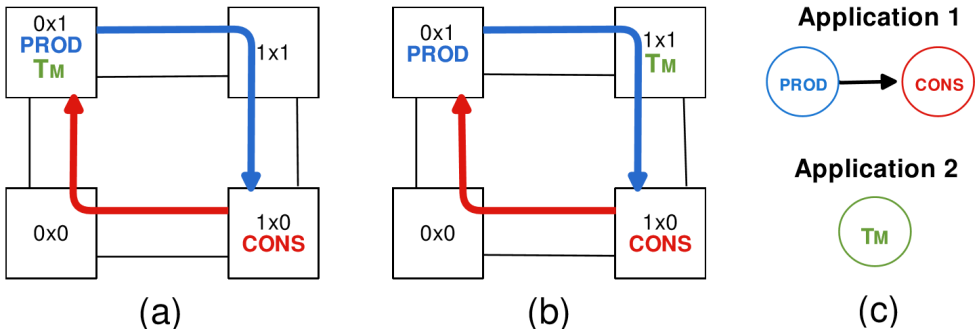


Figura 48: Cenários de mapeamento. (a) Duas aplicações estão compartilhando o mesmo PE (0x1). (b) App1 foi mapeada sem compartilhar o processador com tarefas de outras aplicações (c) Comportamento das tarefas de acordo com a aplicação (fonte: Autor).

É importante ressaltar que o objetivo é apenas evitar o compartilhamento de um PE por tarefas de aplicações distintas. Esta técnica não garante a segurança total para as aplicações, pois a tarefa maliciosa pode realizar um ataque se instalando em outro PE, podendo assim, executar ataque DoS. Não há custo de hardware para implementar essa contramedida, pois ela é implementada via software.

Acoplando dois mecanismos simples, isolamento espacial de aplicativos com filtros de bloqueio de tráfego, evita-se DoS, spoofing e ataques do tipo *hijacking*.

6.7 Mascaramento de acesso à memória

A plataforma SEA também utiliza como contramedida uma máscara para limitar o acesso da região da memória de uma tarefa, visto que a restrição do mapeamento de tarefas afeta do desempenho do MPSoCs quando há muitas tarefas instaladas no sistema. Para isso ela utiliza a identificação da página que a tarefa está alocada, conforme apresentado na Figura 49. Esta máscara limita os endereços acessados por cada tarefa alocada na plataforma.

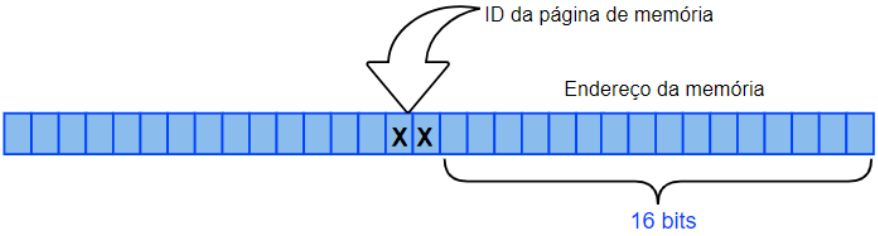


Figura 49: Contramedida instalada na plataforma SEA para evitar acessos indevidos em regiões de memória (fonte: Autor).

6.8 Considerações Finais

Este Capítulo apresentou um conjunto de novas características adicionadas à plataforma SEA, além da existente criptografia AES. Dentre estas características destaca-se:

a) Efetiva implementação da HNoC para transporte de dados sensíveis;

b) Método de filtragem baseada na identificação única das aplicações;

c) Isolamento espacial das aplicações através de restrições aplicadas ao mapeamento de tarefas;

d) Efetiva proteção à gerência de memória através da geração do endereço físico de memória utilizando o número da página.

A Tabela 6 apresenta a relação das contramedidas apresentadas nesta Dissertação com as vulnerabilidades estudadas pela mesma. Este conjunto de contramedidas é eficaz em proteger as aplicações contra os ataques mais comuns, como *DoS*, *spoofing*, *hijacking*, e acesso indevido à memória. Salienta-se que ataque DoS não é totalmente evitado, pois o mesmo pode ocorrer com a saturação dos enlaces utilizados pela aplicação. Ataques mais sofisticados, como *SCA*, também podem ocorrer, sendo mitigados em grande parte pelo uso de criptografia.

Tabela 6: Relação das contramedidas de segurança (primeira coluna da Tabela), e a respectiva proteção a diferentes ataques.

Contramedidas	Vulnerabilidades				
	Ataque à API de Comunicação	Ataque à NoC	Ataque ao SO	Ataque à Memória	Ataque via HT.
Criptografia	x				x
HNoC	x				
Filtro do Firewall	x	x	x		x
Isolação Espacial				x	
Mascaramento de memória				x	

7 RESULTADOS

Este Capítulo apresenta a análise de segurança, latência e o custo de área dos mecanismos de segurança abordados nessa Dissertação. A Seção 7.4 apresenta a comparação dos resultados com o estado-da-arte. É importante destacar que o MPSoC é modelado em VHDL sintetizável, e a simulação possui precisão de ciclos de relógio, o qual permite a avaliação do desempenho usando a ferramenta ModelSim.

7.1 Atuação dos mecanismos de segurança

Esta Seção apresenta um cenário que utiliza três dos quatro mecanismos de segurança: (*i*) isolamento espacial das aplicações; (*ii*) utilização da rede dedicada; (*iii*) filtro de bloqueio do tráfego.

A Figura 50(a) apresenta um pacote com um *App_ID* correto entrando no PE, enquanto a Figura 50(b) apresenta a ação do filtro bloqueando um pacote malicioso. Os quatro sinais superiores em ambas as figuras correspondem aos sinais da porta de saída do roteador e os três sinais inferiores às portas do filtro de saída (Figura 42).

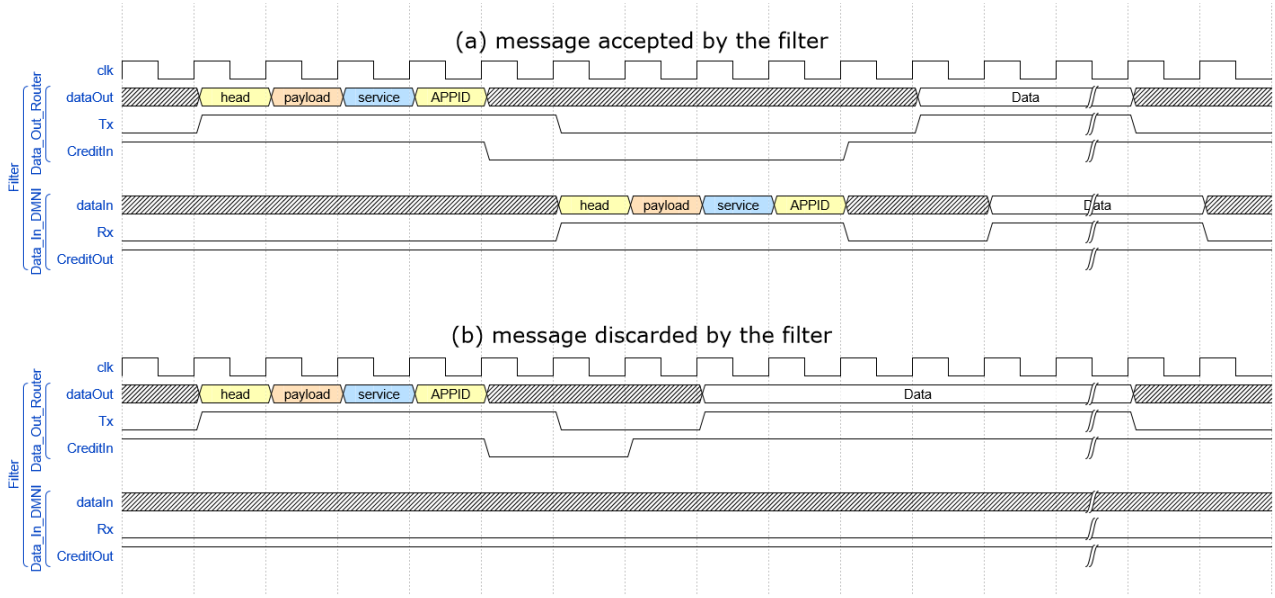


Figura 50: Forma de onda que ilustra a operação do filtro bloqueando mensagens indesejadas (fonte: Autor).

O comportamento da simulação em ambos os cenários é o mesmo. O filtro armazena os primeiros quatro *flits* (o quarto *flit* contém o *App_ID*), e o sinal de crédito (*CreditIn*) vai para zero, interrompendo o fluxo de dados do roteador. Neste momento, o *App_ID* é comparado com o *App_ID* armazenado no filtro (este *App_ID* é enviado através do HNoC durante o mapeamento e armazenado pelo filtro). Se um *App_ID* correto for recebido, as seguintes ações ocorrem: (*i*) o filtro notifica a DMNI (sinal RX = 1) quando há dados válidos, e os quatro primeiros *flits* são consumidos pelo DMNI (*dataIn bus*); (*ii*) o CreditIn vai para

nível lógico alto, e os demais *flits* são consumidos. Se o *App_ID* não coincidir, a ação executada pelo filtro é a de manter o sinal CreditIn em nível lógico alto, sem notificar o DMNI. Essa simples ação corresponde a um consumo virtual, resultando no descarte de qualquer fluxo não direcionado ao *App_ID*. Os filtros bloqueiam qualquer tentativa de injetar pacotes com um *App_ID* errado. No entanto, como mencionado anteriormente, os periféricos ligados a NoC (como memórias compartilhadas ou aceleradores de hardware) podem gerar um pacote com identificação forjada. Nosso mecanismo de segurança proposto descarta tais pacotes, utilizando uma API dedicada de comunicação, o qual é responsável pela troca de dados com periféricos.

7.2 Resultados de latência

Esta subseção avalia o impacto dos mecanismos de segurança inseridos no MPSoC em termos de latência na execução das aplicações no sistema. Compara-se o mesmo cenário de execução com a plataforma de referência denominada HeMPS. O objetivo dos cenários simulados é definir o pior caso de latência e encontrar a latência típica, utilizando aplicações com um perfil intensivo de comunicação (*prod_cons*, duas tarefas) e uma aplicação intensiva de computação (MPEG, cinco tarefas).

O cenário simulado utiliza um MPSoC 3x3, com um *hop* de distância entre as tarefas comunicantes. Devido à pilha de software para criar, transmitir e receber os pacotes, o número de *hops* entre as tarefas comunicantes tem um impacto mínimo na latência dos pacotes. Ambas aplicações executam 50 iterações (50 trocas de mensagens), e a latência média de iteração é obtida a partir dos valores médios entre a décima e a quadragésima interação, ou seja, descartando-se o período de *warm up* e de esvaziamento da rede.

A Tabela 7 apresenta a latência média de iteração para: (*i*) plataforma MPSoC de referência (HeMPS); (*ii*) MPSoC com filtro e criptografia desabilitados; (*iii*) MPSoC com criptografia AES ativa; (*iv*) O MPSoC com filtro (estrutura do firewall somado com o filtro) e o módulo de criptografia SIMON habilitado. Os resultados apresentam comportamentos distintos, de acordo com o perfil de cada aplicação. Para a aplicação de perfil comunicante (*prod_cons*) a média latência aumentou 13,63% numa plataforma com o filtro instalado (sem AES); 46,51% com a AES criptografia, e 256,14% com SIMON, em comparação com a HeMPS de referência. A Figura 51 apresenta um gráfico comparando a latência em ciclos de relógio com as interações da aplicação prod_cons, nota-se o aumento da latência do SIMON em relação as demais plataformas em uma aplicação que explora uma aplicação com perfil comunicante.

O aumento na latência média de iteração da aplicação MPEG é 0,1% com filtro habilitado; 2,55% utilizando criptografia AES e 21,38% utilizando módulo SIMON de criptografia. A razão para explicar o pequeno aumento da latência vêm do fato de que a maior parte do tempo, a aplicação está processando as tarefas de decodificação de

quadros, e não a transferir pacotes pela NoC. A Figura 52 apresenta um gráfico comparando a latência em ciclos de relógio com as interações da aplicação MPEG. Nota-se que o aumento da latência do SIMON em relação as outras plataformas diminuiu, pois a aplicação MPEG possui pouca comunicação entre suas tarefas.

Tabela 7: Latência média de interação, resultados em ciclos de relógio.

	HeMPS	Com filtro	Com AES	Com SIMON
Prod_Cons	**1.973**	**2.128**	**2.752**	**7.081**
MPEG	**220.660**	**220.883**	**226.288**	**241.968**

Estes resultados revelam o custo dos métodos propostos no desempenho das aplicações. Utilizando o mecanismo de filtragem sem habilitar a criptografia, houve um aumento na latência de 0,1% para MPEG e 13,63% para a *prod_cons*, um pequeno custo considerando o aumento da segurança adicionada ao sistema. A criptografia protege o sistema contra SCAs, e como mostrado para um benchmark real, o aumento de latência foi de 2,55%.

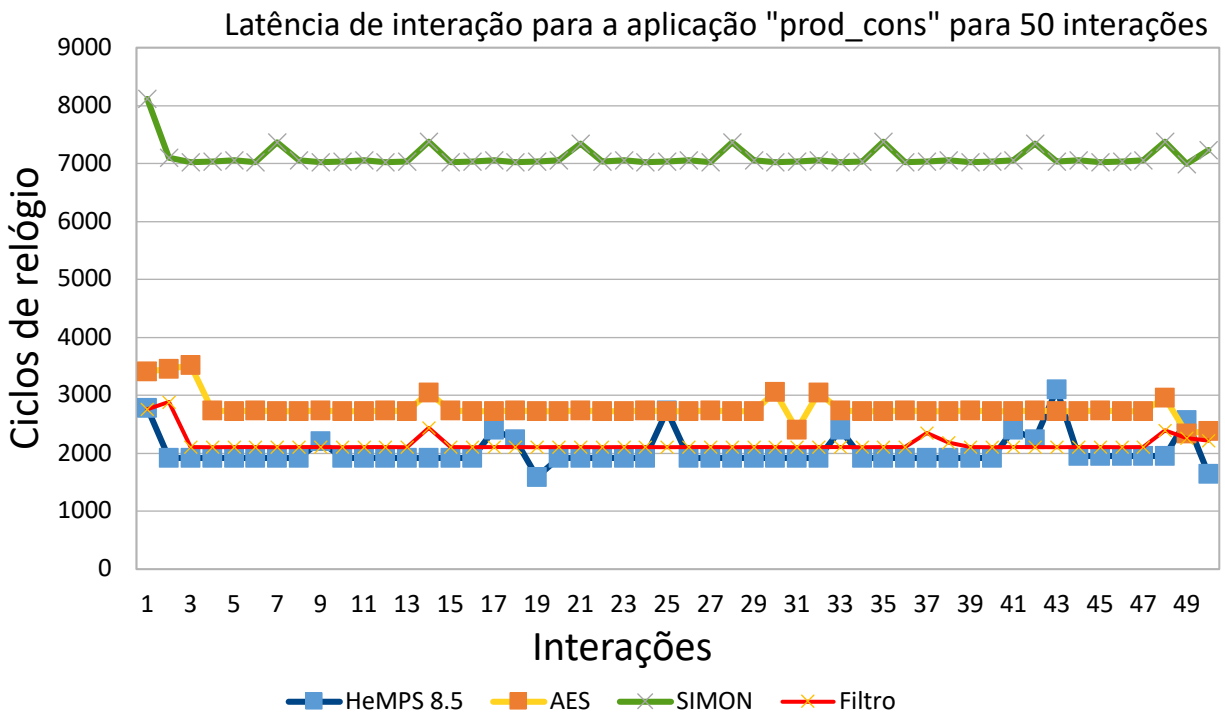


Figura 51: Latência média (ciclos de relógio) para 50 interações realizadas pela aplicação Prod Cons, considerando a plataforma HeMPS, o filtro e o módulo de criptografia AES.

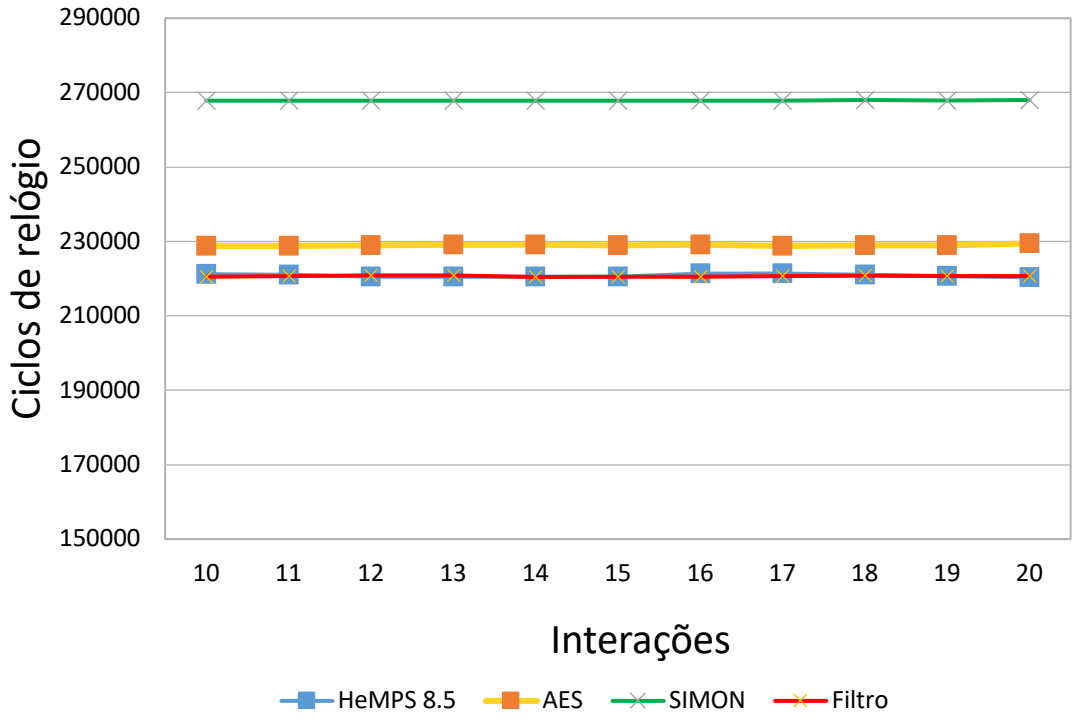


Figura 52: Latência (ciclos de relógio) para as interações 10-20 (exclui-se o período de *warm-up)* realizadas pela aplicação MPEG, considerando a plataforma HeMPS, o filtro e o módulo de criptografia AES.

7.3 Resultados de área

A Tabela 8 ilustra a área ocupada pelo roteador, o filtro e a instanciação de ambos os módulos. O filtro corresponde a 16,33% da área do roteador. Este pequeno custo deve-se ao fato de se utilizar poucos registradores para armazenamento das informações de segurança (chave AES e identificação da aplicação). É importante ressaltar que a coluna de área total apresenta a área das células que compõe o módulo, somado com o roteamento estimado pela ferramenta Genus.

Tabela 8: Área consumida, biblioteca CORE65GPSVT.

Módulos	Células	Área das células (μm²)	Área total (μm²)
Roteador	**5.906**	**43.735**	**59.255**
Filtro	**1.229**	**6.638**	**9.677**
Roteador + Filtro	**7.593**	**53.324**	**73.154**

O módulo de criptografia é desacoplado na implementação do filtro, permitindo ao usuário adotar diferentes mecanismos de criptografia. O presente trabalho adota o método de criptografia AES (núcleo IP de [HEM04], com implementação paralela das rodadas).

Algoritmos de criptografia, como o AES, buscam alcançar altos níveis de segurança, com pouca ou nenhuma preocupação quanto ao consumo de energia ou área de silício. Os módulos de criptografia leve visam a segurança em dispositivos onde a área e a potência são limitadas. Lançada em 2013 pela Agência Nacional de Segurança (NSA), a família de algoritmos Simon e Speck visa trazer alternativas para a criptografia leve. As implementações Speck foram projetadas para melhor desempenho em software, enquanto Simon [BEA15] é mais eficiente para implementações de hardware [BEA13].

A Tabela 9 compara o desempenho de ambos os métodos de criptografia, por simulação e síntese. Ambos os IPs foram implementados para uma chave de 128 bits e um bloco de 128 bits para cifra. A Tabela apresenta a latência para cifrar um bloco de 128 bits, a área de silício e a potência dissipada. Como esperado, há um compromisso entre desempenho e área. O bloco AES é mais rápido, mas apresenta uma área maior. Por outro lado, o SIMON, apresenta uma área pequena, sendo adequado para ser utilizado em todos os filtros sem penalizar a área do PE. Da Tabela 9, os módulos SIMON e AES correspondem, respectivamente a 37,75% e 177,73% da área do roteador.

Tabela 9: Comparação entre os módulos de criptografia Simon e AES – tecnologia 65nm.

	Simon	AES
Latência (Ciclos de relógio)	**140**	**19**
Área ocupada (μm²)	**22.371**	**105.316**
Potência dissipada (μW)	**16.033**	**399.233**

Assim, considerando o módulo de criptografia SIMON, o aumento de área na infraestrutura de comunicação é de aproximadamente 60%, um pequeno custo considerando os benefícios da criptografia para evitar ataques SCA.

7.4 Comparação ao estado-da-arte

Esta seção tem o objetivo de apresentar uma análise quantitativa do presente trabalho com o estado-da-arte mencionado no Capítulo 3.

A Tabela 10 apresenta uma comparação quantitativa da proposta desta Dissertação com o estado-da-arte. É importante ressaltar que os resultados apresentados pelo estado-da-arte utilizam cenários particulares e ambientes próprios, com plataformas distintas da utilizada nesta Dissertação.

Nota-se que o acréscimo do custo é em comparação com a plataforma de referência utilizada pelos trabalhos abordados. Todas as técnicas possuem um aumento de custo de área, potência dissipada e desempenho das plataformas MPSoCs.

É importante destacar que o roteador, ou seja, a infraestrutura de comunicação, não representa mais do que 15% da área PE (processador + memória + NI + DMA + roteador). Consequentemente, os aumentos de área relatados na Tabela 10 representam valores menos significativos em comparação com o MPSoC como um todo.

Tabela 10: Comparação de resultados.

Ref.	Tipo de ataques	Técnicas de Contramedida	Avaliação do Custo para a implementação da Contramedida		
			Potência	Área	Latência
[SEP15]	Time-Driven Attacks, DoS	Mudança no roteamento	+8% (NoC)	+5% (NoC)	Desprezível (caminho crítico)
		Árbitro do roteador	+9% (NoC)	+11% (NoC)	Desprezível (caminho crítico)
[CAI17]	DoS, Timing Attack e Spoofing	Zonas seguras opacas	-	-	+2,56% (DTW)
[FER16]	Timing Attacks e DoS	Zona segura com criptografia	-	-	+16,56% (NAS)
[ANC14]	HT	Criptografia	+5,08% (Roteador)	+9,57% (Roteador)	+3,8% (média benchmark)
		Autenticação do pacote e	Desprezível (Roteador)	+0,34% (Roteador)	+2% (média benchmark)
		Migração de tarefas	-	-	0,01% (média)
[PRA17]	HT e DoS	Implementação de uma Unidade de segurança no roteador.	+3,54% (roteador)	+3,64% (roteador)	-
[REI16]	DTA, timing attack	Mensagens *gossip*, roteador com monitoramento, mudança de roteamento	+16,2% (roteador)	+21,16% (roteador)	-
[FER15]	DoS e Hijacking attacks	Utilização de *firewalls*	-	+13,5% (roteador)	-
[AZA18]	DoS e Hijacking attacks	Utilização de *firewalls*	-	+139% da NI (TSMC 40nm)	+6,46% (caminho crítico)
[KIN17]	DoS, ataques em canais virtuais e ataque na memória física	Utilização de *firewalls*	-	+17% (arquitetura)	+1% até +9% (benchmark) arquitetura
[CHA18]	DoS	Utilização monitores de tráfego	5% - Roteador 5 monitor	+3,4 % (roteador por monitor)	desprezível
Proposta	SCA, DoS, Spoofing, ataque na memória física	Criptografia AES	399,233 µW	+177,7% (roteador)	+46,51% (prod_cons) +2,55% (MPEG)
		Utilização de *firewalls*	-	+16% (roteador)	+0,1% (prod_cons) e +13,63% (MPEG)
		Rede privada para segurança (HNoC).	Apenas custo de interconexão entre os *firewalls*		
		Chaves criptográficas dinâmicas	Apenas custo de software		

Uma observação geral é que a potência é avaliada de forma parcial, apenas no nível do roteador ou o próprio módulo de criptografia. Falta, no estado-da-arte, uma avaliação global do custo de área e potência dos métodos propostos em um sistema completo.

No que se refere ao desempenho das aplicações, utilizando-se benchmarks reais, o impacto das técnicas de segurança empregadas na NoC relatado é sempre mínimo. Isto deve-se ao fato que em benchmarks reais a relação computação/comunicação é alta, ou seja, o tempo em que as aplicações gastam em computação é muito superior ao tempo

gasto em comunicação. Por exemplo, na plataforma de referência esta relação é tipicamente de 95%, o que significa que em apenas 5% do tempo de execução está-se realizando a troca de pacotes entre as tarefas.

8 CONCLUSÕES E TRABALHOS FUTUROS

A presente Dissertação contribuiu com o estudo de mecanismos de segurança instalados em MPSoCs. Dentre estes mecanismos destacam-se a instalação do firewall e suas respectivas políticas de segurança, rede dedicada (HNoC), heurística de geração e distribuição de chaves de criptografia, restrição do mapeamento de tarefas, mascaramento de memória. Além destes, foram implementados os ataques mencionados na literatura de segurança em MPSoCs, mostrando que são factíveis e presentes em nossa realidade.

Pode-se resumir que a segurança abordada nesse trabalho é dividida em duas etapas. Durante a admissão da aplicação, a heurística de mapeamento aplica o isolamento espacial, e o MP gera um identificador de aplicação único e uma chave de criptografia, transmitindo-os através de uma rede de segurança (HNoC) para os PEs que irão executar a aplicação. Em seguida, em tempo de execução, um filtro instalado no firewall verifica se o identificador da aplicação de todos os pacotes que chegam ou saem dos estabelecimentos permanentes. Compreende também, a utilização de criptografia para os pacotes que trafegam na rede.

Além disso, se necessário, os pacotes são criptografados usando AES ou criptografia leve (SIMON), protegendo o tráfego contra SCAs. O custo da abordagem é uma sobrecarga de área (aumento de 60% do roteador para SIMON e 177,7% para AES) e uma sobrecarga de latência inferior a 3% para um *benchmark* real para um sistema com criptografia. Dado que o roteador ocupa uma área de aproximadamente 15% do PE, os aumentos de área comparados ao MPSoC se tornam menos significativos.

É importante ressaltar que o estudo sobre os mecanismos de segurança foi implementado sobre uma plataforma de referência, denominada HeMPS. Através das modificações para o acréscimo de segurança da plataforma, surgiu uma nova plataforma denominada SEA. Entretanto, o Autor acredita que as características arquiteturais adotadas e justificadas nos Capítulos 4 e 6, e as vulnerabilidades de segurança implementadas no Capítulo 5 são genéricas o suficiente para que os resultados obtidos possam ser aplicados em para MPSoCs semelhantes, como os que estão sendo hoje adotados pela indústria e academia.

8.1 Publicações Relacionadas ao Tema da Dissertação

O autor trabalhou na plataforma SEA apresentada no Capítulo 6, cujas principais contribuições incluem: a implementação do caminho Hamiltoniano, protocolo de comunicação da rede dos firewalls, distribuição das chaves pelo caminho Hamiltoniano.

Destaca-se a participação do Autor desta Dissertação nas publicações abaixo, a qual descreve a plataforma SEA e a avaliação de seus mecanismos de segurança:

Secure Environment Architecture for MPSoCs
*Bruno Scherer Oliveira, Henrique Medina, **Anderson Sant'Ana**, Fernando G. Moraes*
*In: **SBCCI**, 2018 - http://dx.doi.org/10.1109/SBCCI.2018.8533238*

Lightweight Security Mechanism for MPSoCs
***Anderson Sant'Ana**, Henrique Medina, Kevin Fiorentin, Fernando G. Moraes*
*In: **SBCCI**, 2019 - https://doi.org/10.1145/3338852.3339876*

8.2 Trabalhos Futuros

Como trabalhos futuros enumeram-se as seguintes atividades:

- Avaliar proteção à ataques de entrada ou saída e definir políticas de contramedida;
- Estudar a plataforma desenvolvida por Luciano Caimi [CAI17], que possui implementada a técnica de zonas seguras contínuas com bloqueio de tráfego, e o comportamento deste frente aos atraques desenvolvidos nesta Dissertação;
- Otimizar o módulo de criptografia SIMON para reduzir sua latência;
- Adicionar mecanismos de detecção de DoS em nível do SO, através do monitoramento da latência dos pacotes.

REFERÊNCIAS

[AGO16] Agosta G.; Barenghi A; Pelosi G. *Automated instantiation of side-channel attacks countermeasures for software cipher implementations*. In: ACM International Conference on Computing Frontiers (CF), 2016, pp. 455-460.

[ALK08] Alkabani, Y; Koushanfar, F. *Extended abstract: Designer's hardware Trojan Horse*. In: IEEE International Workshop on Hardware-Oriented Security and Trust (HOST), 2008, pp. 82-83.

[ANC14] Ancajas, D. M.; Chakraborty, K.; Roy, S. *Fort-NoCs: Mitigating the threat of a compromised NoC*. In: ACM/IEEE Design Automation Conference (DAC), 2014, pp. 1–6.

[AVI04] Avizienis, A; Laprie, J. C; Randell, B.; Landwehr, C. *Basic Concepts and Taxonomy of Dependable and Secure Computing*. IEEE Transactions on Dependable and Secure Computing, v.1(1), Novembro 2004, pp. 11-33.

[AZA18] Azad, S.; Niazmand, B; Jervan G.; Sepulveda, M. *Enabling Secure MPSoC Dynamic Operation through Protected Communication*. In: IEEE International Conference on Electronics, Circuits and Systems (ICECS), 2018, pp. 481-484.

[BEA13] Beaulieu, R.; Shors, D.; Smith, J.; Treatman-Clark, S.; Weeks, B.; Wingers, L. *The SIMON and SPECK Families of Lightweight Block Ciphers. Cryptology.* ePrint Archive, Report 2013/404, 2013. https://eprint.iacr.org/2013/404.

[BEA15] Beaulieu, R.; Treatman-Clark, S.; Shors, D.; Weeks, Smith, J.; Wingers, L. *The SIMON and SPECK lightweight block ciphers*. In: ACM/IEEE Design Automation Conference (DAC), 2015, pp. 1-6.

[BIS15] Biswas, A; Nandy, S; Narayan,R. *Network-on-chip router attacks and their prevention in MP-SoCs with multiple Trusted Execution Environments*. In: IEEE Conference on Electronics, Computing and Communication Technologies (CONECCT), 2015, pp. 6p.

[CAI17] Caimi, L.; Fochi, V.; Wachter, E.; Munhoz, D.; Moraes, F. *Activation of secure zones in many-core systems with dynamic rerouting*. In: IEEE International Symposium on Circuits and Systems (ISCAS), 2017, pp. 4p.

[CAI19] Caimi, L.; Moraes, F. G. *Security in Many-Core SoCs Leveraged by Opaque Secure Zones*. In: IEEE Computer Society Annual Symposium on VLSI (ISVLSI), 2019, 6p.

[CAR09] Carara, E.; Oliveira, R.; Calazans, N.; Moraes, F. *HeMPS: A Framework for NoC Based MPSoC Generation*. In: IEEE International Symposium on Circuits and Systems (ISCAS), 2009, pp. 1345-1348.

[CAS13] Castilhos, G. M. *Gerência Distribuída de Recursos em MPSoCs – Mapeamento e Migração de Tarefas*. 2018. Dissertação de Mestrado, Programa de Pós-Graduação em Ciência da Computação, Pontifícia Universidade Católica do Rio Grande do Sul, 2013.

[CHA18] Chaves, C. G; Azad S.; Sepulveda, M. *A Distributed DoS Detection Scheme for NoC-based MPSoCs*. In: IEEE Nordic Circuits and Systems (NORCAS), 2018, pp. 1-6.

[CLE17] Clements, A. A.; Almakhdhub, N. S.; Saab, K. S.; Srivastava, P.; Koo, J.; Bagchi, S.;

Payer, M. *Protecting Bare-Metal Embedded Systems with Privilege Overlays*. In: IEEE Symposium on Security and Privacy (SP), 2017, pp. 289-303.

[CON16] Conti, M.; Dragoni, N.; Lesyk, V. *A Survey of Man in The Middle Attacks*. IEEE Communications Surveys & Tutorials, v.18(3), Novembro 2016, pp. 2027-2051.

[EVA05] Evain, S; Diguet, J. *From NoC security analysis to design solutions*. In: IEEE Workshop on Signal Processing Systems Design and Implementation, 2005, pp. 166-171.

[FER15] Fernandes, R; Oliveira, B; Sepulveda, J; Marcon, C; Moraes; F. G. *A Non-Intrusive and Reconfigurable Access Control to Secure NoCs*. In: IEEE International Conference on Electronics, Circuits and Systems (ICECS), 2015, pp. 316-319.

[FER16] Fernandes, R.; Marcon, C.; Cataldo, R.; Silveira, J.; Sigl, G.; Sepúlveda J. *A security aware routing approach for NoC-based MPSoCs*. In: Symposium on Integrated Circuits and Systems Design (SBCCI), 2016, 6p.

[GRE11] Greenhalgh, P. big.LITTLE *Processing with ARM Cortex-A15 and Cortex-A7*. Disponível em: https://www.cl.cam.ac.uk/~rdm34/big.LITTLE.pdf. Acessado junho de 2019.

[HAN05] Hansman S.; Hunt R. *A taxonomy of network and computer attacks*. Computers and Security, v.24(1), Maio 2005, pp. 31-43.

[HEM14] Hemanth. *AES128::Overview*. Disponível em: https://opencores.org/project,aes_crypto_core. Acessado em junho de 2019.

[ISA13] Isakovic, H.; Wasicek, A. *Secure channels in an integrated MPSoC architecture*. In: IEEE Industrial Electronics Society (IECON), 2013, pp. 4488–4493.

[KAR01] Karri R.; Wu, K; Mishra, P; Kim, Y. *Concurrent error detection of fault-based side-channel cryptanalysis of 128-bit symmetric block ciphers*. In: ACM/IEEE Design Automation Conference (DAC), 2001, pp. 579–584.

[KIN17] Kinsy M. A; Khadka, S.; Isakov, M.; Farrukh, A. *Hermes: Secure heterogeneous multicore architecture design.* In: Hardware Oriented Security and Trust (HOST), 2017, pp. 14–20.

[KOC11] Kocher, P.; Jaffe, J.; Jun, B.; Rohatgi, P. *Introduction to differential power analysis*. Journal of Cryptographic Engineering, v.1(1), Maio 2011, pp. 5-27.

[KOU12] Koushanfar, F.; Sadeghi, A.; Seudie, H. *Eda for secure and dependable cybercars: Challenges and opportunities*. In: ACM/IEEE Design Automation Conference (DAC), 2012, pp. 220-228.

[LEE11] Lee, S.; Oh, J.; Park, J.; Kwon, M.; Kim, M.; Yoo H. *A 345 mW Heterogeneous Many-Core Processor with an Intelligent Inference Engine for Robust Object Recognition.* IEEE Journal of Solid-State Circuits, vol. 46(1), Dezembro 2011, pp. 42-51.

[MAR18] Martins, A. M. *Multi-Objective Resource Management for Manycore Systems*. 2018. Tese de Doutorado, Programa de Pós-Graduação em Ciência da Computação, Pontifícia Universidade Católica do Rio Grande do Sul, 2018.

[MOR04] Moraes, F.; Calazans, N.; Mello, A.; Möller, L.; Ost, L. *HERMES: an Infrastructure for Low Area Overhead Packet-switching Networks on Chip*. Integration VLSI Journal, vol. 38(1), Maio 2004, pp. 69-93.

[OLI18a] Oliveira, B.; Reusch, R.; Medina, H.; Moraes, F. *Evaluating the Cost to Cipher the*

NoC Communication. In: Latin American Symposium on Circuits & Systems (LASCAS), 2018, 4p.

[OLI18b] Oliveira, B. S.; Medina, H.; Sant'Ana, A.; Moraes, F. *Secure Environment Architecture for MPSoCs*. In: Symposium on Integrated Circuits and Systems Design (SBCCI), 2018, pp. 1-6.

[PRA17] Prasad, N.; Karmakar, R.; Chattopadhyay S.; Chakrabarti, I. *Runtime mitigation of illegal packet request attacks in Networks-on-Chip*. In: IEEE International Symposium on Circuits and Systems (ISCAS), 2017, pp. 1-4.

[RAM02] Ramachandran, J. *Designing security architecture solutions. Canada,* John Wiley & Sons, Inc. 2002, 452p.

[REI16] Reinbrecht, C.; Susin, A.; Bossuet, L.; Sepúlveda, J. *Gossip NoC - Avoiding Timing Side-Channel Attacks through Traffic Management*. In: IEEE Computer Society Annual Symposium on VLSI (ISVLSI), 2016, pp. 601-606.

[RHO01] Rhoads, S. *Plasma - most MIPS I(TM) opcodes::Overview*. Disponível em: https://opencores.org/projects/plasma, acessado em junho 2019.

[RUA16] Ruaro, M.; Lazzarotto, F. B.; Marcon, C. A.; Moraes, F. G. *DMNI: A specialized network interface for NoC-based MPSoCs.* In: IEEE International Symposium on Circuits and Systems (ISCAS), 2016, pp. 1202-1205.

[SEP14] Sepúlveda, J; Gogniat, G; Flórez, D; Diguet, J; Zeferino, C; Strum, M. *Elastic Security Zones for NoC-Based 3D-Many-core*. In: IEEE International Conference on Electronics, Circuits and Systems (ICECS), 2014, pp. 506-509.

[SEP15] Sepulveda, M.; Diguet, J.; Strum M.; Gogniat G. *NoC-Based Protection for SoC Time-Driven Attacks*. IEEE Embedded Systems Letters, v.7(1), Maio 2015, pp. 7–10.

[SEP17] Sepulveda, J.; Flórez, D.; Immler, V.; Gogniat, G.; Sigl, G. *Efficient security zones implementation through hierarchical group key management at NoC-based MPSoCs*. Microprocessors and Microsystems, v.50, Abril 2017, pp. 164-174.

[TEH10] Tehranipoor M.; Koushanfar, F. *A Survey of Hardware Trojan Taxonomy and Detection*. IEEE Design & Test of Computers, v.27(1), Novembro 2010, pp. 10-25.

[WAC17] Wachter, E.; Caimi, L.; Fochi, L.; Munhoz, D.; Moraes, F. *BrNoC: a Broadcast NoC for Control Messages in Many-core Systems*. Microelectronics Journal, vol. 68, Outubro 2017, pp. 69–77.

[WAS13] Wassel, H. M. G.; Gao, Y.; Oberg, J. K.; Huffmire, T.; Chong, F. T.; Sherwood, T. *SurfNoC: A low latency and provably non-interfering approach to secure networks-on-chip*. In: International Symposium on Computer Architecture (ISCA), 2013, pp. 583-594.

[WEB19] Wever, I.; Lopes, G. “Hermes Odisseia: Desenvolvimento de um Hardware Trojan”. Relatório da Disciplina de Sistemas Integrados – PPGCC, 2019.

[YAM14] Yamada, S.; Nakamoto, Y. *Protection Mechanism in Privileged Memory Space for Embedded Systems, Real-Time OS*. In: IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW), 2014, pp. 161-166.

Pontifícia Universidade Católica do Rio Grande do Sul
Pró-Reitoria de Graduação
Av. Ipiranga, 6681 - Prédio 1 - 3º. andar
Porto Alegre - RS - Brasil
Fone: (51) 3320-3500 - Fax: (51) 3339-1564
E-mail: prograd@pucrs.br
Site: www.pucrs.br