PUCRS

ESCOLA POLITÉCNICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
DOUTORADO EM CIÊNCIA DA COMPUTAÇÃO

RAFAEL FOLLMANN FACCENDA

# SECURING APPLICATIONS IN NOC-BASED MANY-CORE SYSTEMS – A COMPREHENSIVE METHODOLOGY

Porto Alegre
2024

PÓS-GRADUAÇÃO - *STRICTO SENSU*


Pontifícia Universidade Católica
do Rio Grande do Sul

**PONTIFICAL CATHOLIC UNIVERSITY OF RIO GRANDE DO SUL**
**SCHOOL OF TECHNOLOGY**
**COMPUTER SCIENCE GRADUATE PROGRAM**

# SECURING APPLICATIONS IN NOC-BASED MANY-CORE SYSTEMS – A COMPREHENSIVE METHODOLOGY

**RAFAEL FOLLMANN FACCENDA**

Doctoral Thesis submitted to the Pontifical Catholic University of Rio Grande do Sul in partial fulfillment of the requirements for the degree of Ph. D. in Computer Science.

Advisor: Prof. Dr. Fernando Gehm Moraes
Co-Advisor: Prof. Dr. Luciano Lores Caimi

**Porto Alegre**
**2024**

# Ficha Catalográfica

F137s Faccenda, Rafael Follmann

Securing Applications in NoC-based Many-Core Systems - A Comprehensive Methodology / Rafael Follmann Faccenda. – 2024.
140.
Tese (Doutorado) – Programa de Pós-Graduação em Ciência da Computação, PUCRS.

Orientador: Prof. Dr. Fernando Gehm Moraes.
Coorientador: Prof. Dr. Luciano Lores Caimi.

1. NoC-based many-cores. 2. Security. 3. OSZ (opaque secure zones). 4. Secure communication. 5. Security framework. I. Moraes, Fernando Gehm. II. Caimi, Luciano Lores. III. , . IV. Título.

Elaborada pelo Sistema de Geração Automática de Ficha Catalográfica da PUCRS
com os dados fornecidos pelo(a) autor(a).
Bibliotecária responsável: Clarissa Jesinska Selbach CRB-10/2051

**RAFAEL FOLLMANN FACCENDA**

# SECURING APPLICATIONS IN NOC-BASED MANY-CORE SYSTEMS – A COMPREHENSIVE METHODOLOGY

This Doctoral Thesis has been submitted in partial fulfillment of the requirements for the degree of Ph. D. in Computer Science, of the Computer Science Graduate Program, School of Technology of the Pontifical Catholic University of Rio Grande do Sul

Sanctioned on 12th March, 2024.

## COMMITTEE MEMBERS:

Prof. Dr. Marcio Eduardo Kreutz (PPgSC/UFRN)

Prof. Dr. Ney Laert Vilar Calazans (PGMICRO/UFRGS)

Prof. Dr. César Augusto Missio Marcon (PPGCC/PUCRS)

Prof. Dr. Luciano Lores Caimi (UFFS- Co-Advisor)

Prof. Dr. Fernando Gehm Moraes (PPGCC/PUCRS - Advisor)

## AGRADECIMENTOS

Gostaria de deixar meus agradecimentos para todos que me apoiaram e acreditaram em mim durante esta etapa.

Primeiramente, aos meus pais Lúcia e Irineu que sempre fizeram seus máximos para que eu tivesse boas oportunidades e conseguisse aproveitá-las da melhor forma. Este agredecimento se estende a toda minha família de Sarandi que mesmo de longe se fizeram presentes na minha vida.

Também à minha família escolhida: Roberto, Iaçanã, Gabriele, Ingrid, Júlia, que sempre me apoiaram e acreditaram em mim, compartilhando angústias mas também alegrias. Foram essenciais para que eu chegasse até aqui.

Aos meus orientadores Prof. Moraes por ter me acolhido na PUCRS, sendo ótimo orientador nesta jornada, constantemente acreditando no meu trabalho, me fazendo enxergar além das minhas inseguranças, e Prof. Caimi que além de emprestar a plataforma base desta tese, sempre se fez presente e deu suporte para minhas ideias. Também ao Gustavo, que chegou no meio deste projeto e trouxe contribuições essenciais para que alcançássemos os objetivos.

Por fim, agradeço à CAPES pela bolsa de doutorado para que eu pudesse me dedicar integralmente a este projeto, e à PUCRS e Escola Politécnica pela estrutura e serviços que viabilizaram a execução das atividades.

# SEGURANÇA DE APLICAÇÕES EM SISTEMAS MANY-CORE BASEADOS EM NOC – UMA METODOLOGIA ABRANGENTE


## RESUMO

Sistemas many-core em Chip (MCSoCs) alcançam alto desempenho através de paralelismo espacial. Estes sistemas contém Elementos de Processamento (PEs) interconectados por infraestruturas de comunicação como Redes-em-Chip (NoCs). À medida que os MCSoCs se tornam amplamente adotados e sua complexidade aumenta, a proteção de dados emerge como um requisito de projeto crítico. A questão central de pesquisa é: "como proteger aplicações executando em MCSoCs contra ameaças de segurança"? A revisão da literatura identifica várias técnicas de defesa, incluindo criptografia, códigos de autenticação, códigos de correção de erros e detectação de fluxos anômalos. Enquanto esses métodos protegem o MCSoC contra ataques específicos, há uma lacuna referente à proteção abrangente contra uma ampla gama de ameaças (DoS e espionagem, por exemplo). A primeira contribuição original desta Tese é uma taxonomia que categoriza as propostas de segurança em cinco critérios ortogonais: (1) fonte e tipo das ameaças; (2) contramedidas; (3) fase da aplicação onde as contramedidas são executadas; (4) sobrecarga relacionada às contramedidas; (5) contramedidas realizadas em tempo de projeto ou de execução. A Zona Segura Opaca (OSZ) é um mecanismo de segurança realizado em tempo de execução que visa o isolamento espacial de aplicações, oferecendo proteção robusta a ataques externos. No entanto, vulnerabilidades persistem quando Hardware Trojans (HTs) infectam roteadores da OSZ ou quando uma aplicação executando um uma OSZ comunica-se com dispositivos de IO. Esta Tese aborda essas vulnerabilidades, propondo três inovações técnicas originais: (1) Mapeamento Seguro com Ponto de Acesso (SeMAP): permite o mapeamento simultâneo de múltiplas OSZs, protegendo aplicações contra acessos não autorizados e garantindo a disponibilidade de caminhos para os dispositivos de IO usando um protocolo de autenticação leve; (2) Gerente de Sessão: um mecanismo para monitoramento, detecção e recuperação de ataques ou falhas que interrompem a entrega de pacotes; (3) Framework de Segurança Integrado: combina os mecanismos de segurança desenvolvidos em um framework, enviando alertas de segurança para um Gerente de Sistema. Uma campanha de ataques severa testou a resiliência da plataforma, que a protegeu de todos ataques executados. Resultados mostraram recuperação bem-sucedida dos dados e execução correta de todas as aplicações, com uma penalidade média de tempo de execução de 4,47%. Este resultado destaca a eficácia das contramedidas propostas, demonstrando que a segurança das aplicações tem baixo custo em termos de tempo de execução. Além disso, a sobrecarga de área de silício é pequena, correspondendo à adição de uma NoC de controle e módulos para controlar o acesso aos links dos roteadores.



**Palavras-Chave:** Sistemas many-core baseados em redes intra-chip, segurança, OSZ (zonas seguras opacas), comunicação segura, framework de seguraça.

# SECURING APPLICATIONS IN NOC-BASED MANY-CORE SYSTEMS – A COMPREHENSIVE METHODOLOGY

## ABSTRACT

Many-core systems on Chip (MCSoCs) achieve high performance through spatial parallelism. They include Processing Elements (PEs) interconnected by complex communication infrastructures as Networks-on-Chip (NoCs). As MCSoCs become widely adopted and their complexity increases, data protection emerges as a critical design requirement. The central research question of this Thesis is: "how to protect applications running on MCSoCs against security threats"? The literature review identifies various defense techniques, including cryptography, authentication codes, error correction codes, and establishing communication flow profiles to detect anomalous behavior. While these methods protect the MCSoC from specific attacks, there is a gap in proposals of comprehensive protection against a wide range of potential threats (e.g. DoS and Eavesdropping). The first original contribution of this Thesis is a taxonomy that categorizes MCSoC security proposals into five orthogonal criteria: (1) source and type of the threats; (2) countermeasures; (3) application phase where countermeasures are executed; (4) overhead related to the countermeasures; (5) design time or runtime countermeasures. The Opaque Secure Zone (OSZ) is a runtime security mechanism designed for spatial isolation of applications, offering robust protection against external attacks. However, vulnerabilities persist when Hardware Trojans (HTs) infect OSZ routers or the need for secure communication with external IO devices. This Thesis approaches these vulnerabilities, proposing three original technical innovations: (1) Secure Mapping with Access Point (SeMAP): enables the simultaneous mapping of multiple OSZs, protecting secure applications against unauthorized accesses and ensuring the availability of paths to the IO devices using a lightweight authentication protocol; (2) Session Manager: a mechanism for monitoring, detecting, and recovering from attacks or faults disrupting packet delivery; (3) Integrated Security Framework: combines the developed security mechanisms into a comprehensive framework, sending security warnings to a System Manager. A severe attack campaign tested the platform's resilience, which protected the platform against all attacks. The results showed successful data recovery and correct execution of all applications, with an average execution time penalty of 4.47%. This outcome underscores the effectiveness of the proposed countermeasures, demonstrating that securing applications has low cost in terms of execution time. The silicon area overhead is small, concerning the addition of a control NoC and modules for controlling access to the routers' links.

**Keywords:** NoC-based many-cores, security, OSZ (opaque secure zones), secure communication, security framework.

# LIST OF FIGURES

# LIST OF TABLES

# LIST OF ACRONYMS

3PIP – Third Part Intellectual Property
AES – Advanced Encryption Standard
AP – Access Point
API – Application Program Interface
CAM – Content Addressable Memory
CPU – Central Processor Unit
CTG – Communication Task Graph
DDoS – Distributed Denial of Service
DH – Diffie-Hellmann
DMA – Direct Memory Access
DMNI – Direct Memory Network Interface
DoS – Denial-of-Service
DSZ – Dynamic Secure Zone
ECC – Error Correction Code
ECDH – Elliptic Curve Diffie-Hellmann
EOP – End-of-Packet
ET – Execution Time
FPGA – Field Programmable Gate Array
HT – Hardware Trojan
IO – Input/Output
IoT – Internet of Things
IP – Intellectual Property
LFSR – Linear-feedback Shift Register
MAC – Message Authentication Code
MPE – Manager Processing Element
MCSoC – Many-core System on Chip
NI – Network Interface
NoC – Network on Chip
OSZ – Opaque Secure Zones
OS – Operating System
PE – Processing Element
PS – Packet Switch

PUF – Physical Unclonable Function
RTL – Register Transfer Level
SCA – Side Channel Attack
SDN – Software Defined Network
SeMAP – Secure Mapping with Access Points
SNIP – Secure Network Interface for Peripherals
SR – Source Routing
SZ – Secure Zone
TEE – Trusted Execution Environment
VHDL – VHSIC Hardware Description Language
VHSIC – Very High Speed Integrated Circuit

# CONTENTS

# 1. INTRODUCTION

Many-core Systems on Chip (MCSoC) are platforms designed to provide high-performance systems based on parallelism, meeting the current demand for embedded devices with power consumption and communication constraints. An MCSoC contains PEs (Processing Elements) interconnected by complex communication infrastructures, such as hierarchical buses or NoCs (Networks-on-Chip) [Popovici et al., 2010]. PEs may be processors, 3PIP (third-party intellectual property) modules, memory blocks, and dedicated hardware accelerators. Examples of modern architectures with a large number of processors interconnected by NoCs include the Mellanox family TILE-Gx72 (72 cores) [Tecnhlogies, 2018], Intel Knights Landing [Sodani et al., 2016], Oracle M8 (32 cores) [Oracle, 2017], Kalray array (256 cores) [Dinechin et al., 2014], KiloCore chip (1,000 cores) [Bohnenstiehl et al., 2016], and Esperanto (1,100 RISC-V cores) [Peckham, 2020].

An NoC consists of routers and links and is responsible for forwarding data and control messages between PEs. Network Interfaces (NI) connect PEs to the routers of the NoC. Whenever a PE sends a message, the NI transforms it into a packet and delivers it to the router. Then, the router sends the packet to a neighbor router through a link according to a path defined by the routing algorithm. The routers constitute the underlying communication infrastructure of the system, where multiple interconnected routers define the network topology [Hemani et al., 2000, Benini and Micheli, 2002].

As the adoption and complexity of MCSoCs increase, the concern for data protection appears as a design requirement [Baron et al., 2013]. An MCSoC may be employed in scenarios where availability is critical, and downtimes must be minimized. These systems may also handle sensitive information; thus, protecting this data from unauthorized access is necessary [Kumar et al., 2021]. According to [Ramachandran, 2002], not only data protection, unauthorized access, and availability are concerns on the MCSoC design. The following seven security principles are generally accepted as the foundation of a good security solution being the three first principles mandatory features [Ramachandran, 2002]:

- Confidentiality: the property of non-disclosure of information to unauthorized processes, entities, or users;
- Availability: the protection of assets from DoS (Denial-of-Service) threats that might impact the utilization of system resources;
- Integrity: the prevention of modification or destruction of an asset by an unauthorized entity or user;
- Authentication: the process of establishment and validation of a claimed identity;
- Authorization: the process of determining whether a validated entity is allowed to access a secured resource based on attributes, predicates, or context;

- Auditing: the property of logging the system activities at levels sufficient for the reconstruction of events;
- Nonrepudiation: the prevention of any participant denying his role in the interaction once completed.

A consequence of the increasing number of features and functionalities inside a single chip is the adoption of 3PIPs to meet time-to-market constraints and reduce design costs. Such IPs come from different vendors, raising the risk of having a Hardware Trojan (HT) insertion [Li et al., 2016]. Assuming HTs infect the NoC, these can perform several attacks that threaten security principles [Ramachandran, 2002]. Such attacks may affect *confidentiality* by redirecting messages to malicious agents, *availability* by dropping messages or blocking a communication path, and *integrity* by corrupting the content of a packet traversing the NoC.

Based on the threats presented above, the *motivation* of this Thesis is to answer the following question: "how to protect applications running on MCSoCs against security threats". The literature presents several techniques, such as cryptography [Charles and Mishra, 2020b], authentication codes [Sharma et al., 2019], error correction codes [Gondal et al., 2020], creation of a communication flow profile to detect anomalous behavior [Charles et al., 2020b]. Adopting these techniques makes it possible to detect violations related to security or faults in the NoC. Moreover, authors propose spatial isolation via Secure Zones (SZ) to protect communication and computation simultaneously. A particular case of SZ is the Opaque Secure Zone (OSZ) [Caimi and Moraes, 2019], which is a defense mechanism executed at runtime that focuses on finding a rectilinear region on the system with free PEs to map an application with security constraints. The OSZ activation occurs by setting wrappers at the boundaries of the rectilinear region, blocking all incoming and outgoing traffic trying to cross the OSZ boarders.

OSZ prevents attacks from outside sources, such as Denial-of-Service (DoS), timing attacks, spoofing, and man-in-the-middle [Caimi and Moraes, 2019, Caimi et al., 2018a]. Even though the method is robust against external attacks, it still presents vulnerabilities when HTs infect routers inside the OSZ or when the application running in the OSZ needs to communicate with peripherals[1].

Therefore, gaps relate to robust defense mechanisms effective against most attacks reported in the literature. As discussed in the state-of-the-art (Chapter 2), defense mechanisms seek to protect the MCSoC from a given specific attack, lacking proposals that protect the system against a plethora of possible threats.

[1] also referred in this Thesis as *IO devices*

## 1.1 Thesis Statement

It is feasible to integrate a comprehensive suite of methods to secure application execution in MCSoCs, with low impact on execution time and area footprint, through a framework that monitors and detects suspicious behavior and applies countermeasures to reinforce security. The focus is to protect simultaneously the computation and communication resources of sensitive applications, including access to IO devices.

## 1.2 Objectives

The OSZ mechanism is the baseline security method. Challenges include protecting the system against HTs and making communication with external devices (e.g., shared memories, 3PIPs, hardware accelerators) safe. The execution of attack campaigns aims to demonstrate the Thesis statement. Nonetheless, it is essential to identify weaknesses not covered by the final set of secure methods.

The strategic objective of this Thesis is to propose a framework that promotes security against several threats by integrating different defense mechanisms in software and hardware that operate throughout the application execution, including IO communication.

To reach the strategic objective, the following specific goals are set:

*SG1* **System organization to define the locations of OSZs**.
One of the major concerns of this Thesis is to promote secure IO communication. This goal aims to solve OSZ mapping issues by defining Gray and Restricted areas. The Gray area runs non-secure applications and provides paths for peripherals, regardless of the OSZs location (mapped in the Restricted areas), also ensuring that new OSZs do not disturb the existing IO flows.

*SG2* **Communication protocol with IO devices**.
Communication with IO devices requires an opening on the OSZ to exchange packets with external elements. This objective focuses on developing software and hardware to enable IO transactions without incurring new vulnerabilities. The software requires a dedicated communication API using a master-slave protocol, with all transactions starting from the PEs inside the OSZ. The hardware support refers to the creation and management of Access Points in the borders of the OSZ.

*SG3* **IO packet authentication**.
The packets exchanged between OSZ and IO devices can be harmed by attackers, the Access Point and the peripheral itself. The objective is to protect the IO communication elements utilizing a lightweight Authentication Protocol, identifying the legitimate packets and protecting against unauthorized access.

*SG4* **Attacks targeting the communication with IO devices**.
Performing attacks on the platform helps measure the impacts of running applications without security mechanisms and the effectiveness of the defense mechanisms. In addition, these attack campaigns guide design choices. For example, traffic monitoring may indicate an anomalous behavior, triggering countermeasures.

*SG5* **Protect the communication inside the OSZ**.
As previously described, HTs are malicious hardware inserted into the system without the designer's knowledge. An undetected HT can affect the secure application even within an OSZ. The goal is to create communication Sessions to monitor the inner OSZ traffic, detecting anomalies that suggest the presence of HTs or faulty links.

*SG6* **Attack campaign with HTs and recovery mechanism**. Objective *SG5* resulted in a protocol to detect the occurrence of attacks inside the OSZ. The objective is to emulate HT on specific routers to validate the proposal. The evaluation of the attack led to the proposal of a Recovery mechanism.

*SG7* **Integration of security mechanisms**.
Objectives *SG1*-*SG5* target different threats and were analyzed separately. The current objective aims to integrate the defense mechanisms inside a Framework, based on a monitor-detection-countermeasure structure.

*SG8* **Warning reports**.
As the attacks are becoming more complex, the countermeasures need to evolve. Therefore, this objective focuses on implementing a detection system that reports security violations at runtime to a Security Manager responsible for system-level countermeasures.

*SG9* **Final attack campaing**.
This objective aims to submit the Framework to a distributed attack campaign, including simultaneous DoS, Spoofing, and Eavesdropping attacks. Then, analyze the effectiveness of security mechanisms and measure the total overhead induced by the countermeasures.

## 1.3 Original Contributions

This Thesis has four original contributions:

1. A taxonomy that categorizes related work into five orthogonal criteria: (1) source and type of the threats, (2) countermeasures; (3) application phase where countermeasures are executed; (4) overhead related to the countermeasures; (5) design time or runtime countermeasures (Chapter 2);
2. Secure Mapping with Access Point (SeMAP), a proposal enabling the mapping of multiple OSZs simultaneously, authenticating IO communication, and protecting secure applications against unauthorized accesses (Chapter 4);
3. Session Manager, a mechanism designed to monitor, detect, and recover the system against attacks or faults that disrupt packet delivery (Chapter 5);
4. Framework that integrates all developed security mechanisms, sending security warnings for a System Manager to enable systemic countermeasures (Chapter 6).

## 1.4 Document Organization

This document is organized as follows.

- Chapter 2 presents contemporary literature research and the proposed taxonomy for security in NoC-based MCSoC;
- Chapter 3 details the baseline platform selected to develop the proposed methods;
- Chapter 4 presents SeMAP, fulfilling objectives *SG1* to *SG4*;
- Chapter 5 presents the Session Manager, fulfilling objectives *SG5* and *SG6*;
- Chapter 6 describes Security Framework, fulfilling objectives *SG7* to *SG9*.
- Chapter 7 presents the conclusion and future work.

# 2. CONTEMPORARY RESEARCH IN MCSOC SECURITY

This Chapter presents contemporary research on threats and countermeasures for MCSoCs. We performed a literature review in two moments: first, in March 2020, that included works from 2018 up to 2020, and the second, in December 2023, that covered works published between 2021 and 2023. The review used the search string (detailed in Appendix C) on Scopus.

In 2020, instead of discussing proposals individually, we proposed a taxonomy related to security for these systems, corresponding to the *first original contribution* of this Thesis. This taxonomy embraces five orthogonal criteria explained in detail referencing related work.

In 2023, we used the same search string to select works between 2021 and 2023, analyzing them through our proposed taxonomy and comparing them with the 2020 research. Besides the overview of the MCSoC security research field, this Chapter also presents a detailed discussion of works closely related to this Thesis.

This Chapter is structured beginning with the proposed taxonomy (Section 2.1), followed by detailed discussions on each criterion (Sections 2.2 to 2.6). The Chapter concludes with the Thesis placement w.r.t. the related work and a table comparing them (Section 2.8).

## 2.1 Proposed Taxonomy

The foundation of the proposed taxonomy is based on the work of Avizienis et al. [Avizienis et al., 2004], who proposed an initial taxonomy related to dependability and security. This taxonomy categorizes proposals into three distinct groups: attributes, threats, and means. These categories are visually represented in Figure 2.1.

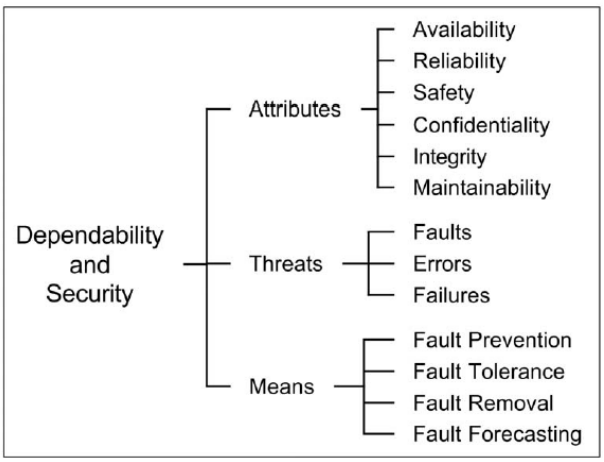



Figure 2.1 – Taxonomy for dependability and security proposed by [Avizienis et al., 2004].

Avizienis' taxonomy includes security and dependability in a unified way. However, since this Thesis focuses on the security axis and not fault tolerance, we propose a specific taxonomy for security in MCSoCs. This taxonomy covers the following criteria:

1. **attack**: describes the source and type of the threats;

2. **countermeasure**: methods adopted to detect, avoid or mitigate the attacks;

3. **application phase**: an application typically has 3 phases: admission, execution, and communication with peripherals [Caimi, 2019]. This criterion refers to when the attack occurs, and which is the adopted countermeasure;

4. **cost**: the overhead related to the countermeasure in the system;

5. **integration** : when the countermeasure or the attack is implemented or activated.

Figure 2.2 presents the proposed taxonomy. The next five subsections discuss each taxonomy column in detail.

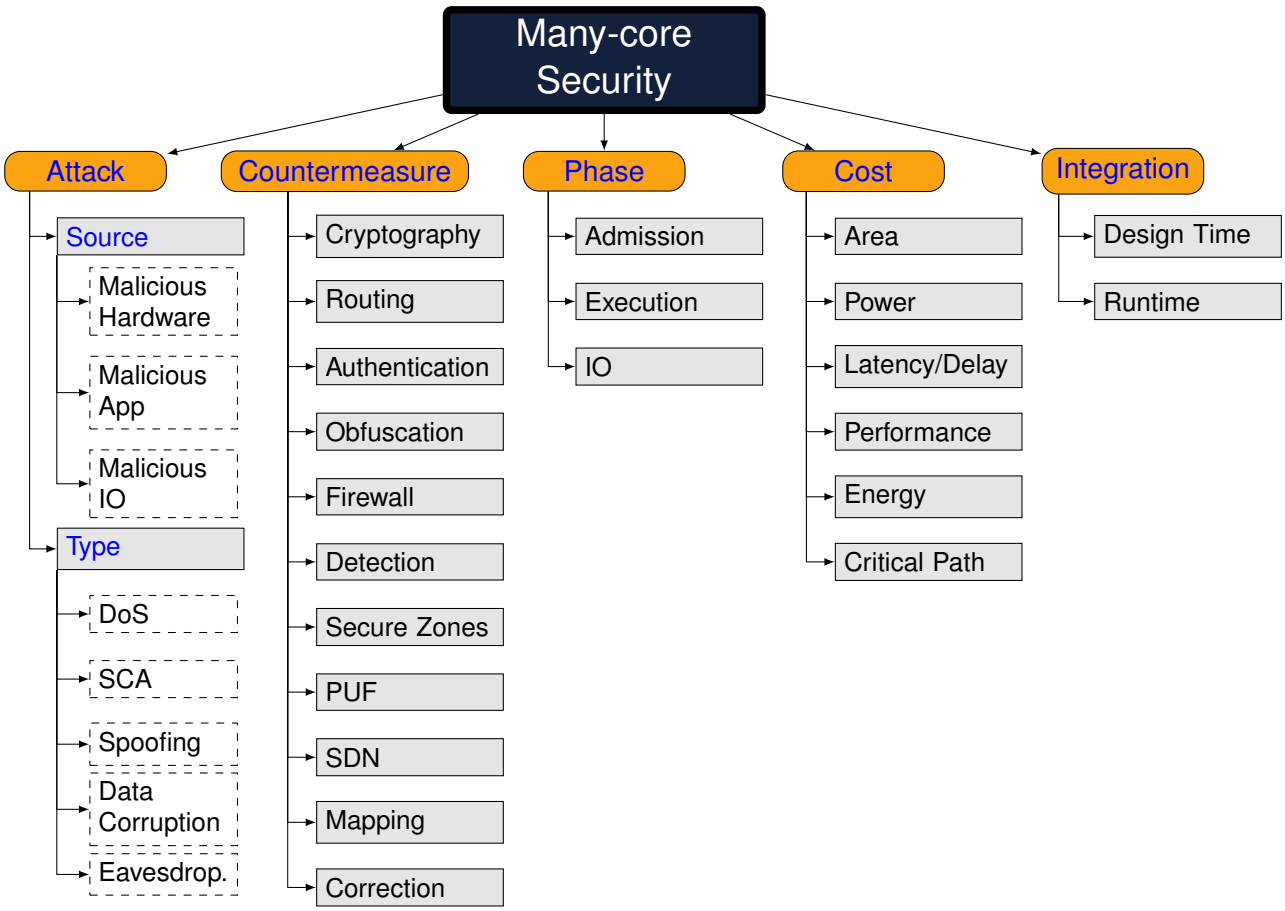



Figure 2.2 – Proposed taxonomy related to threats and countermeasures for MCSoCs.

## 2.2 Criterion 1 - Attack

Commonly, the first topic discussed on an MCSoC security work is the threat model, where the authors describe the activity of a possible intruder or harmful behavior. It is important to identify two main attacks components: the **source** and the the **type**.

The **source** refers to the agent of the threat, which can be Malicious Hardware (commonly known as Hardware Trojan - HT), Malicious Application (*MalApp*), or Malicious IO device. The other component is **type**, which specifies the threat activity and its effects on the system, such as Denial of Service (DoS), Side-channel Attack (SCA), Spoofing, Data Corruption, or Eavesdropping, for example. Observing those components in the threat model is essential to clearly understand the specific issues that each author is working on. For example, a DoS attack performed by an HT or a *MalApp* may cause similar effects on the system, but the action taken against an HT DoS may not be effective against an *MalApp*. Some examples of the combination of different types with sources: DoS caused by an HT [Daoud and Rafla, 2019b], SCA done by a *MalApp* [Reinbrecht et al., 2020], and a spoofing attack performed by a malicious IO device [Elkanishy et al., 2019].

### 2.2.1 Source

In Figure 2.2, under the *Attack* branch, there are three main **sources** of attacks: Malicious Hardware, Malicious Application, Malicious IO device. This subsection discusses those attacks sources.

#### Malicious Hardware

The literature contains several reports of malicious hardware, a.k.a. Hardware Trojan (HT) [Boraten and Kodi, 2016, Kulkarni et al., 2016, Zhang et al., 2018, Raparti and Pasricha, 2019, Zhao et al., 2020, Gondal et al., 2020]. The main reason enabling the insertion of HTs in a design is the distributed production chain adopted in the microelectronics industry, which allows designers to build a system with IPs from different design companies (3PIPs). Such a situation raises the question: “Is this foreign IP trustworthy?”. The answer to this question is not simple since most IPs do not reveal their content or design process to protect intellectual property.

Related works investigate the feasibility of inserting HTs, detecting them, or countermeasures to the attacks carried out by them. Philomina [Philomina, 2021] overviews HT effects on NoC-based systems. The author also categorizes HTs according to a taxonomy,

which includes the IC supply chain phase (when the HT can be embedded), activation, effects, location, type, and size.

Zhang et al. [Zhang et al., 2018] present an example of HT in a NoC, inserting malicious hardware at the input ports of the routers. This HT is designed to discard packets that pass through it, characterizing a DoS attack. In this case, the HT needs to be configured and activated by a malicious application. In summary, Zhang et al. present a threat model of an attack executed through an HT in cooperation with a malicious application with considerable knowledge of the system and access to system functions.

Raparti and Pasricha [Raparti and Pasricha, 2019] describe an HT inside the network interface (NI), which is the component that makes the interface between the PE and the NoC. The HT is placed near the packetization modules modifying the packet header, changing the target, and causing packets to be delivered to a different PE. The authors also mention that such HT is intrusive, with high area overhead and needs to be configured by an application with access to specific system functions.

Therefore, based on the assumptions made by the threat models with HTs, the attacker should furtively insert the HT in a given system module. Some authors also suggest an application that can configure and activate the HT to perform an attack successfully.

### Malicious Application

Intruders broadly use malicious applications to execute attacks. This attack is characterized by suspicious software loaded to the system that runs a harmful code able to damage or spy the many-core resources, such as memories, communication traffic, power dissipation (by monitoring DVFS messages).

Reinbrecht et al. [Reinbrecht et al., 2020] use malicious software to discover sensitive information inside memories through cache access attempts (timing attacks, discussed on Section 2.3). The malicious application needs permission to access the moment of memory readings besides a privileged view of the shared memory addresses to execute this attack.

Forlin et al. [Forlin et al., 2019] use malicious applications to inject low-priority traffic to retrieve time information about the high-priority traffic behavior. In this case, the malicious application must assign priorities to its messages (a function normally restricted to the system manager). Moreover, the malicious software needs to be placed in specific locations and with knowledge of the routing algorithm and system topology, which requires advanced access to system managing functions.

Mountford et al. [Mountford et al., 2023] present a threat model that combines Malicious Hardware (HT) with Malicious Applicaation. A rogue application allocated in the system activates an HT that counts the packets passing through the infected NoC router.

The HT then sends the gathered information back to the rogue application, which forwards the information to an external device that can perform attacks, such as traffic analysis.

#### Malicious IO devices

Malicious IO devices are peripherals that attempt to perform an attack when connected to an input/output port of the system. Elkanishy et al. [Elkanishy et al., 2019] describe the BlueBorne attack through a Bluetooth (BT) interface. The BlueBorne attack allows the external device to pair with the BT chip without consent or when the BT chip is not in discovery mode. Once this connection is established, the external device can access and control crucial resources to the system operation.

It is also possible to execute attacks using hardware outside the system, which can, by proximity, monitor and extract information related, e.g., to the system electromagnetic irradiation. As discussed in [Kenarangi and Partin-Vaisband, 2019], it is possible to use these electromagnetic values in statistical analysis to extract sensitive information about an AES (Advanced Encryption Standard) encryption process.

Ahmed et al. [Ahmed et al., 2021] consider a threat model composed by a malicious hardware, called Remote Access Hardware Trojan (RAHT), placed on NoC routers that leak information about traffic pattern to a malicious IO device. The RAHT is able to count packets and send this information via an IO port to an external data-analysis attacker (Malicious IO device) that could utilize machine learning techniques to infer the applications running on the system or even reverse engineer IPs from the system.

### 2.2.2 Type

The attack **type** refers to the threat goal, including the targeted resources and the harmful behavior to affect them.

#### Denial-of-Service (DoS)

Denial-of-Service (DoS) is a broad category that includes attacks that deny or hinder the operation of a given system function. Flooding, packet loss, and misrouting are common attacks executed at the communication level. The goal of these attacks is to disturb or even block the communication in the many-core.

Zhang et al. [Zhang et al., 2018] explore DoS attacks known as Blackhole and Sinkhole, executed by HTs inside routers. The Blackhole attack consists of an infected router that discards packets passing through it. Similar to the Blackhole attack, the Sinkhole

attack redirects the flow that passes through it to another target (i.e., misrouting). In both cases, the communication is compromised, denying the packet exchange between PEs.

An extended version of DoS, the Distributed Denial-of-Service (DDoS), consists of several compromised IPs performing DoS attacks simultaneously. Charles et al. [Charles et al., 2020b] present a DDoS executed from four malicious IPs targeting one single IP, congesting the NoC (i.e., flooding), causing the latency to rise significantly, severely compromising the system performance.

Yao et al. [Yao et al., 2023] describe a DoS attack targeting the arbiters of input ports within a router's switch allocator to disrupt packet transmission. This attack is executed by an HT, named Spotlight, which maliciously suppresses the priority of packets, leading to a significant increase in packet latency.

### Side-Channel Attacks (SCA)

The Side-Channel Attack gathers information through direct interference and eavesdropping. The attacker may discover important information about the many-core based on the retrieved data. The SCA is the primary type of attack associated with leaking or stealing sensitive information, which can be from different natures, as a physical attribute, such as power dissipation, or timing information, such as latency.

Ho et al. [Ho et al., 2019] present an SCA over a physical attribute, named *Differential Power Analisys* (DPA). This attack monitors and analyzes the power dissipation of a PE during an AES encryption process. As the power dissipation of each encryption round is different, the attacker may be able to identify which round is being processed and infer the key through statistical methods.

Besides power dissipation, there are electromagnetic SCAs that monitor electromagnetic irradiation [Xiao et al., 2020]. In this case, the electromagnetic irradiation over a capacitor at the core power supply line enables the extraction of power traces from an AES encryption. These traces are fed to a Convolutional Neural Network (CNN) to retrieve the key.

Reinbrecht et al. [Reinbrecht et al., 2020] presents a Logical SCA (LSCA). This SCA attempts to retrieve data by observing logical effects, such as Timing and Cache Attacks. The authors execute the LSCA by intentionally causing traffic collisions and then monitoring their latency. Based on the latency values, it is possible to estimate the traffic pattern of a sensitive application. Reinbrecht et al. also discuss the cache attack, which attempts to retrieve information stored on shared memory, forcing flushes and reloads of the cache. Based on the timing and behavior of the memory, one can guess the specific area of a memory used by a sensitive application. Implementations and further investigation of cache attacks are presented in detail in [Reinbrecht et al., 2019] and [Ge et al., 2019].

Ali et al. [Ali and Khan, 2021] present in their threat model a *covert communication attack*, which, according to the authors, is one category of software side-channel attack (sSCA). Two independent applications exploit the platform setup to communicate covertly. To transmit a secret bit, application X floods the network with memory access, while application Y measures memory access latency. Each determined time window, application Y infers the value of the secret bit: if the latency is high, it means that the secret bit is 1, if the latency is low, the secret bit is 0.

### Spoofing

Spoofing is an attack characterized by a malicious source (hardware, IO, or software) that successfully falsifies its identity to obtain unauthorized privileges. Once received these privileges, the attacker can access sensitive information. Elkanishy et al. [Elkanishy et al., 2019] present a Bluetooth attack promoted by a malicious peripheral that accessed the system bypassing the pairing protocol.

Through spoofing attacks, malicious sources can access system controlling units and violate the resource distribution. Zhao et al. [Zhao et al., 2020] explore a vulnerability in the system power budgeting by modifying power configuration packets. This attack may alter the performance of a sensitive application, enhance the resources of a malicious task, or even damage the system by violating safe temperature limits.

Bish et al. [Bisht and Das, 2022] developed an HT that misroutes the packets and is able to counteract a defense mechanism by blaming their neighbor routers for the attack. This is a spoofing attack because the HT utilizes techniques to falsify its identity as a non-infected router.

### Data Corruption

Most authors consider that faults occur due to external agents, such as radiation, electromagnetic interference, crosstalk, or aging factors. On the other hand, attackers can induce intentional faults, by corrupting data or carrying out laser attacks.

Gondal et al. [Gondal et al., 2020] present an HT that corrupts messages to disturb communication. Based on the effects of this HT, they propose an error correction mechanism (discussed in the Countermeasure section).

Zhang et al. [Zhang et al., 2018] discuss a laser attack, which forces glitches on a neuromorphic processing unit (NPU). Using a laser source, the attacker can change the neurons' weights. From the retrieved information, it is possible to discover the learning strategy.

Eavesdropping

Eavesdropping is the direct steal of information from applications. Raparti and Pasricha [Raparti and Pasricha, 2019] present a data-snooping promoted through message duplication and redirection. In this case, the attack is performed by a router infected with an HT. The authors propose a solution to this attack by designing a module to invalidate the eavesdropping and detect the attack location.

Kulkarni et al. [Kulkarni et al., 2021] implement an HT as a small circuit installed in the router's input buffers. The HT attack, which is randomly triggered, manipulates the header flit, changing the packet target and redirecting it to another PE.

### 2.2.3 Discussion

Figures 2.3 and 2.4 summarize the 2020 and 2023 research results, respectively, regarding the type and source of attacks. The squares in the intersection of the type-source grid contain the number of works found in the research. Note that the values are not accumulated, meaning that the graphs do not share works.

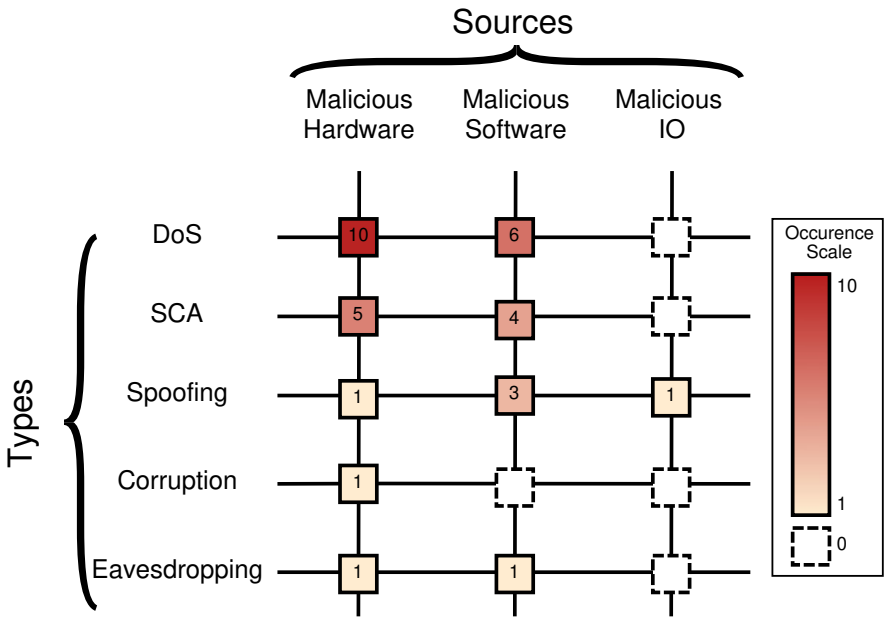



Figure 2.3 – Research overview according to the attack type and source - 2020.

In the 2020 research, the attack source with the most occurrences is the malicious hardware (18 mentions). This higher occurrence is a consequence of the system modularization, with 3PIPs that may not be completely trustworthy. However, an issue related to the malicious hardware approach is that the insertion of the HTs is invasive. The attacker usually needs access to the system design or execute a reverse engineering process. For this

reason, the authors make assumptions that severely limit the application of the proposals related to HTs.

Regarding the attack type, the most mentioned is Denial-of-Service (DoS). This happens because DoS is a generic term that covers different attacks. In addition, some DoS attacks are simpler to implement when compared with other attack types, such as SCA and Spoofing, which require more resources to perform the attack successfully.

On the other hand, the least explored attack source is the malicious IO. The absence of publicly available MCSoC platforms allowing the connection with peripherals to explore such attacks explains why few works explore them. The connection with external devices opens a considerable attack surface. The many-core may now interact with devices that may not be known in advance (e.g., the previously mentioned attack using a Bluetooth interface). Therefore, the field of on-chip security focusing on IO connections was an open research field.

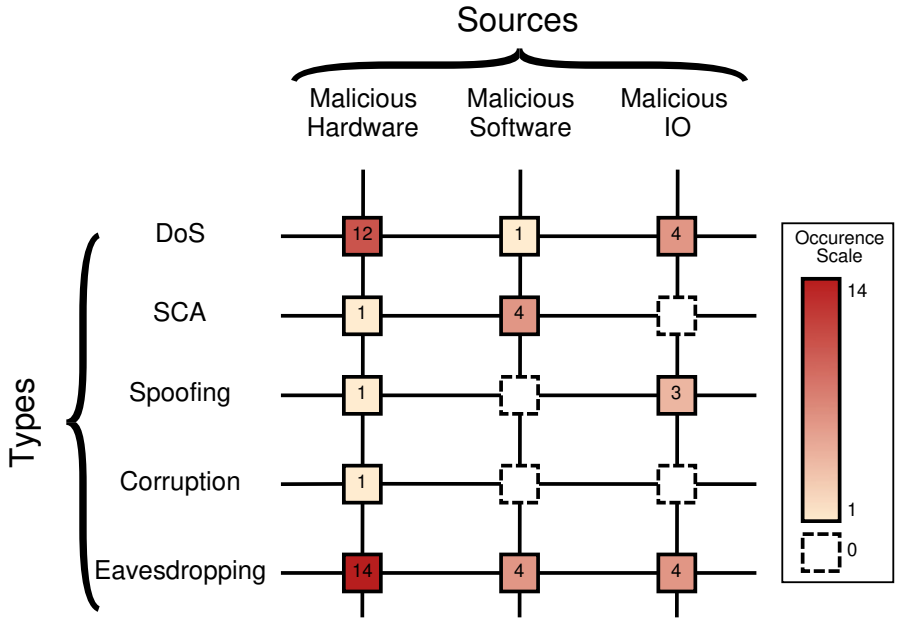



Figure 2.4 – Research overview according to the attack type and source - 2023.

In the 2023 research (Figure 2.4), there was a continued focus on malicious hardware as the primary source of attacks, but with a notable shift in the type of attacks to eavesdropping. This shift indicates that researchers have broadened the spectrum of attacks executable by HTs, highlighting the need for countermeasures against not only impairing threats like Denial of Service (DoS) but also those that involve leaking sensitive information, such as Eavesdropping. While DoS remains a prevalent threat, several studies have considered HTs capable of executing both DoS and Eavesdropping attacks.

The incidence of malicious software executing attacks on MCSoCs has decreased. While numerous articles reference malicious applications or tasks to contextualize threats

to MCSoCs, a limited number of papers specifically focus on and developing attacks from malicious software. Among those that do, the majority execute SCA.

Malicious IO were almost non-existent in the 2020 research and now present a slight increase in occurrences. However, most of these occurrences are attributed to our publications, mainly derived from the protection of IO communication (Chapter 4). Beyond our contributions, only two other papers present malicious peripherals.

## 2.3 Criterion 2 - Countermeasure

This section categorizes the state-of-the-art proposals for defense mechanisms. We present a brief description for each countermeasure, followed by works using them. Proposals found in the literature research are, in most cases, robust defense mechanisms that employ more than one technique simultaneously to protect the system. For this reason, we may use the same work in different categories.

### Cryptography

Cryptography is used to create a secure channel for sensitive information exchange, restricting the messages' reading only by units with cryptography keys.

Charles and Mishra [Charles and Mishra, 2020b] present an optimized encryption method named incremental cryptography. The authors focus on reusing similar blocks that have already been encrypted in previous rounds, preventing the system from repeating the encryption and saving time and energy.

Oliveira et al. [Oliveira et al., 2018] propose a secure architecture and provide the costs of increasing the security for NoCs. The architecture contains a firewall capable of filtering incoming and outgoing network traffic and encrypting sensitive information, performing end-to-end security using an AES cipher block. The firewall plus the AES increases the router area by 193.7%, and latency increases in the worst-case scenario by 395.92%. Despite this performance penalty, the authors argue that the injection rate of applications is low (typically 5-10%), resulting in a small performance overhead at the application level.

Furthermore, cryptography is often discussed in works that explore key-exchange protocols to establish secure communication channels. Harttung et al. [Harttung et al., 2019] propose three lightweight authentication methods, named Individual Authentication, Interwoven Authentication, and Full Generation Authentication. The authors combine the authentication methods with lightweight ciphers (PRESENT, mCRypton, and PRINCE). Results show that the information rate (amount of flits created by PEs and the total number of flits injected into the network, indicating the protocol overhead) increases by 40% and

increases the average end-to-end latencies by 10%. The authors also argue that this is an acceptable penalty considering the added security in the communication.

Tran et al. [Tran et al., 2021] proposes a hardware-based cryptosystem to protect IoT (Internet-of-Things) devices. The authors implement an AES 128-bit symmetric cryptography algorithm using VHDL focusing on an SoC FPGA. They embed the system onto the DE10-Standard board, which can act as node or gateway. As a node, it receives environment data through sensors, encrypts it and sends to the gateway devices via wireless. As a gateway, the system must receive this data, decrypt it, and put it on the server or cloud.

### Routing

Several attacks propose routing algorithms to avoid, e.g., HTs, in a given path between PEs. Charles et al. [Charles et al., 2020a] propose a method named Anonymous Routing. This method inserts extra layers of protection that hide the source and target fields of the packets. The method has two phases: (*i*) route discovery, which sends packets to discover the route between a source and target PEs, sharing security parameters with the routers in the defined path; (*ii*) data transfer, using virtual circuit numbers (VCN). When transferring packets, intermediate routers only see the VCNs corresponding to the preceding router and the following router, which reveals no information about the source or the destination. As a result, the message exchange becomes untraceable for devices that do not have access to the key to decrypt the packets. In this case, anonymous routing combines the routing algorithm with cryptography and obfuscation.

Indrusiak et al. [Indrusiak et al., 2019] describe the implementation of route randomization in a many-core as a protection mechanism against SCA. In this case, varying the routers taken by sensitive traffic prevents the collision with malicious traffic provoked by an SCA, making the SCA information extraction considerably harder since the timing measures are no longer precise.

Patooghy et al. [Patooghy et al., 2023] proposes another type of anonymous routing, encrypting the entire packet and the NoC routes with the encrypted addresses. Each router must decipher the turns on the header address to decide the channel to forward the packet. The secure packet is delivered to the target when the encrypted destination field matches the encrypted address of the router.

### Authentication

Authentication is the process of establishment and validation of a claimed identity. Kornaros et al. [Kornaros et al., 2018] use authentication to secure the system boot. Caimi et al. [Caimi et al., 2018a] use authentication to protect the admission of secure applications.

Kornaros et al. [Kornaros et al., 2018] present the security structure named IOSCU (I/O Secure Communication Unit), which is a firewall-based authentication mechanism. The IOSCU goal is to promote a secure boot, especially in systems with connected peripherals, with the authentication process isolated from the software. During regular operation, the firewall can be configured by units with credentials to create a secure communication session.

Caimi et al. [Caimi et al., 2018a] protect the application deploy into the many-core. The first step is the mutual authentication, where an external device and the many-core exchange keys, using the ECDH (Elliptic Curve Diffie-Hellman) protocol. After authentication, tasks are loaded with a MAC (Message Authentication Code) attached to the object code to guarantee the code integrity.

Sharma et al. [Sharma et al., 2019] propose a twofold key agreement to create secure communication sessions among processing clusters inside the same chip, guaranteeing message exchange only between authenticated units. Harttung et al. [Harttung et al., 2019] propose a mechanism at the message level. Authentication is performed for every message, combined with a MAC to check the delivered message's integrity.

Charles et al. [Charles et al., 2022] propose a digital watermarking of packets to guarantee packet authenticity and detect eavesdropping attacks. Watermark encoders and decoders are installed at the NI of each node, which inserts unique watermarks on each packet stream based on a shared secret between source and destination.

### Obfuscation

Obfuscation represents a group of countermeasures that protect information or resources, hiding or disguising them against attackers that try to eavesdrop to discover sensitive data. Ho et al. [Ho et al., 2019] obfuscate the power dissipation values of an AES encryption process by taking turns between the AES execution and a dummy process. As a result, the power values SCAs could target are not accurate enough to retrieve the cryptographic key. Besides this technique, the authors also use configurable clock frequencies to obfuscate the data leaks.

Forlin et al. [Forlin et al., 2019] make an in-depth analysis of SCA attacks in a Priority-Preemptive NoC (PP-NoC). The authors propose three defense mechanisms that use obfuscation methods against SCAs:

- RT-Blinding: periodically send dummy high-priority packets to avoid attackers to gather information about the timing of sensitive packets;
- RT-Masking: intentionally saturate the channels when a high-priority flow is passing, so the actual size and pattern of the flow is not easily discovered;

- RT-Shielding: tuning of the NoC parameters (as buffer size) to prevent the interference of low-priority traffic into high-priority traffic. This countermeasure require design time traffic analysis to define the NoC parameters.

Mountford et al. [Mountford et al., 2023] introduce a method for obfuscating the addresses of packets transmitted through the NoC. They propose the implementation of a look-up table (LUT) within the router, which determines the packet's direction based on a specific routing code. To enhance security, the indexes of the LUT are randomized for each router at the design time, making it more challenging for an attacker to discern their values. Additionally, the authors add another layer of obfuscation by padding the path length, ensuring that every packet has an identical number of routing flits.

### Firewall

A firewall in the scope of on-chip communication is a hardware barrier placed at the ports of the communication structure to control the input and output of an element. This mechanism generally comprises a table to store the recognized trusted sources and a controller allowing credentialed traffic and blocking unauthorized traffic. Azad et al. [Azad et al., 2018] propose a firewall to guarantee the integrity and confidentiality of the platform. The firewall is placed at the NI and has two tables: (*i*) *initiator table*, which checks if the source has permission to send messages; (*ii*) *target table*, which verifies if the message is allowed to enter the target unit. Another important aspect of this proposal is that a Security Manager configures the firewalls that send configuration messages protected with a MAC.

The same authors, in [Azad et al., 2019], propose the CAESAR-MCSoC, using a group of firewalls to isolate a cluster of nodes in the platform, resulting in a secure zone, a concept discussed later. This work places the firewalls into the NI (CAESAR NI) alongside the Packetizer, Depacketizer, and a CAESAR Core (crypto core for encryption and authentication).

Oliveira et al. [Oliveira et al., 2018] propose an architecture that includes a firewall capable of filtering incoming and outgoing NoC traffic, an AES cipher block to encrypt the NoC flow, and an auxiliary NoC that uses a Hamiltonian path to configure the firewall rules and distribute the keys. Instead of changing the interfaces, state machines in the firewall manage the flow control signals. The firewall may encrypt the packets according to an identifier in the packet or not.

### Detection

Countermeasures categorized as detecting and monitoring are the proposals that focus on tracking suspicious behavior and, in some cases, taking actions to neutralize the activity of a threat (with the help of other countermeasures). This class is characterized by

works that detect or locate threats with the help of monitors inserted in different parts of the system. Chaves et al. [Chaves et al., 2019] insert a DoS monitor inside the router, named CPRD (Collision Point Router Detection). The packets sent through these routers have an extra field that carries the value of the router in which the packet waited the most. When the packet arrives at the target, the latency is analyzed, and if it is above the expected latency of the packet, a DoS suspicion is reported. Once there are a certain number of DoS reports, the platform understands that an attack is happening and activates the firmware to locate and reset the source of the attack.

Kenarangi and Partin-Vaisband [Kenarangi and Partin-Vaisband, 2019] use machine learning sensors fed with voltage-level information of the PDN (power delivery network). These sensors protect the system against power, timing, or electromagnetic SCAs. Thus, the sensors are trained with the transient state values of the PDN, enabling them to detect unexpected behaviors, especially in terms of power and electromagnetic attacks.

Sudusinghe et al. [Sudusinghe et al., 2022] implements an eavesdropping detection technique based on machine learning. The approach is has three phases: (1) design time, in which features to infer NoC traffic and probes are attached to routers to gather traffic information. Then, during (2) training time, the gathered information is utilized to train the ML models, which, once trained, are stored in an IP that will act as a decision unit (DU). At (3) runtime, the DU utilizes the ML models and the NoC traffic collected by the probes to make decisions or even detect eavesdropping attacks.

### Secure Zones

Secure Zone is a countermeasure that explores spatial isolation, i.e., reservation of resources to execute one secure application. Caimi and Moraes [Caimi and Moraes, 2019] propose the *Opaque Secure Zone* (OSZ) technique, executed at runtime. This technique uses software-configurable wrappers to “close” a region of the MCSoC, blocking both incoming and outgoing packets arriving at the OSZ borders. The OSZ closing and opening protocols are done via a control-NoC, separated from the data-NoC, and unavailable at the application level. For each application that requests an OSZ, the platform manager defines the required resources and maps and allocates the application on those resources, securing the application admission phase. During the secure application execution, packets that try to traverse the OSZ are rerouted. The OSZ can be opened to IO packets according to a secure protocol.

As mentioned before, in the firewall section, Azad et al. [Azad et al., 2019] create a secure zone using programmable firewalls to protect the CAESAR-MCSoC. Moreover, the authors also deploy a firewall configuration protocol with encrypted and authenticated configuration packets.

Benhani et al. [Benhani et al., 2019] use secure zones in a Trusted Execution Environment (TEE) based on the ARM TrustZone mechanism in a heterogeneous SoC. In this case, the authors use cryptography to establish secure communication via an AXI Bus.

### Physical Unclonable Function (PUF)

Physical Unclonable Function (PUF) is a hardware unit whose output depends on manufacturing factors, creating unpredictable behavior, thus, considered as impossible to be violated. Siddiqui et al. [Siddiqui et al., 2019] propose a secure boot in FPGAs involving PUF and backend server. The server challenges the PUF in the client, provoking responses that, once verified by the server, send the bitstream to configure the application logic in the FPGA. Kumar et al. [Kumar et al., 2018] propose using a PUF to protect the system against the misuse of test structures to leak confidential data from the cryptographic or critical cores. Thus, the authors create a wrapper that can only be accessed by authenticated test engineers.

PUFs are also utilizes on proposals for secure test environments, Zhang et al. [Zhang et al., 2022] uses PUF to create a secure wrapper for NoC-based SoC that is compatible to IEEE 1500. The original key is encrypted into a stream cipher by the PUF module, and the input key must match or the wrapper is locked, promoting an effective authentication.

### Software-Defined Network (SDN)

SDN in NoC aims to reduce the router complexity by moving the control logic (arbitration and routing) to software, creating a control layer. The SDN Controller has a global view of the communication infrastructure, being able to make decisions according to the current constraints and status of the network.

Ruaro et al. [Ruaro et al., 2020] propose the SDN-SS, an SDN framework for MCSoCs. This work presents a secure SDN management, with protocols to secure the SDN configuration, including secure router configuration with key-exchanging mechanisms.

Ellinidou et al. [Ellinidou et al., 2019] propose the SSPSoC, also based on SDNs, but the target system is a cloud-of-chips, which is a larger dimension of MCSoC architectures with a large number of ICs. Similar to the previous work, the authors' concerns are security, inserting defense mechanisms such as private key derivation, group key agreement, and data exchange phases to provide secure communication.

### Mapping

Task mapping is also used as a security mechanism. Tibaldi et al. [Tibaldi et al., 2021] present a framework to generate heterogeneous SoC by selecting processing and

communication components and after defining the mapping and scheduling of tasks. Design space exploration is done with a *Ant Colony Optimization* heuristic, including security as a constraint, resulting in the application's performance and security improvement. The authors focus on reinforcing security via task mapping and distribution of resources to avoid the influence of public communication flows into private flows.

A Thermal Covert Channel (TCC) attack is a type of security exploit that leverages thermal emissions from electronic devices to transmit information covertly. The fundamental principle behind a TCC attack is manipulating temperature changes in a system to create a covert channel for data transmission. Wu et al. [Wu et al., 2021] propose a countermeasure to TCC attacks by task migration. The authors identify that task distance weakens the thermal signal and propose a task migration algorithm that blocks TCCs based on task distance.

#### Correction

The proposals based on correction in the scope of many-core security are defensive mechanisms against faults caused intentionally by an intruder - as the corruption attack [Gondal et al., 2020]. In that same work, the authors design a specific router, called Junction Routers, enhanced with Error Correcting Codes (EEC). The selected EEC by the authors is the Joint Cross Talk Avoidance Triple Error (JTEC), which can detect and avoid 3-bit errors. As a result, several Junction Routers are placed at the NoC, and they successfully correct the packets that the attacker corrupts.

### 2.3.1 Discussion

Figure 2.5 and Figure 2.6 presents the countermeasures versus the types of attacks. The intersections indicate the number of works that use the countermeasure to defend against a given attack. Note that one work can use a countermeasure against more attacks, counting in more than one intersection. The color represents the number of works found in the literature.

The 2020 research showed DoS as the most reported attack. The most frequent countermeasures against DoS attacks are:

- adaptive routing, change the path when detecting a suspicious behavior;
- secure zones, deviate the traffic that tries to traverse a secure application;
- obfuscation, masks the packet contents to the attacker.

SCAs are present in the threat models, with the timing attack being the most frequent. The countermeasures obfuscation, authentication, routing, firewalls, and secure

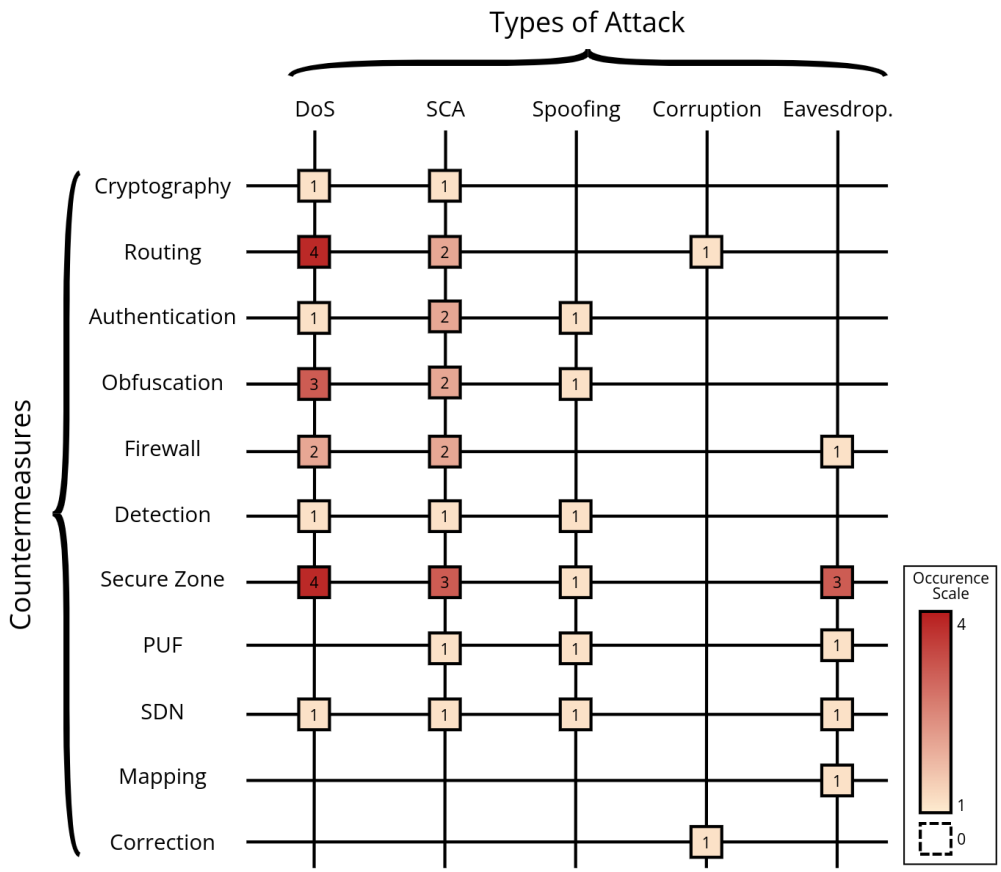



Figure 2.5 – Countermeasure overview according to type of attack from 2020 research.

zones are adopted to avoid or mitigate this attack. Those techniques can control the access to sensitive information, such as timing readings, sensitive traffic path, power dissipation, and cache accesses.

Secure Zones is an effective countermeasure against most attacks. The robustness of this countermeasure is due to the spatial isolation and controlled access through configurable wrappers. Caimi et al. [Caimi et al., 2021] review several secure zone proposals. Besides Secure Zones, SDN appears, with lower numbers, as a defense mechanism against four out of five attack types, showing to be an alternative to increase many-cores' security. The main limitation of SDN is the number of dedicated SDN subnetworks [Ruaro et al., 2020].

Another situation observed in Figure 2.5 is the relationship *correction x corruption*. Only one work mentions this type of attack and countermeasure. One reason for the lack of work related to data corruption is that most countermeasures are associated with the fault-tolerance field. Thus, fault-tolerance techniques may be used together with security countermeasures, as presented in the Avizienis' taxonomy [Avizienis et al., 2004].

Spoofing and eavesdropping are two types of attacks that are more complex and, for this reason, require a more robust attack system, including specific software routines and

access to specific platform features. Therefore, these attacks are not very recurrent in the 2020 research.

However, the 2023 research (Figure 2.6) showed increased spoofing occurences and even more eavesdropping attacks. In addition, cryptography and authentication are the most used countermeasures to avoid sending data to unauthorized entities inside the platform.

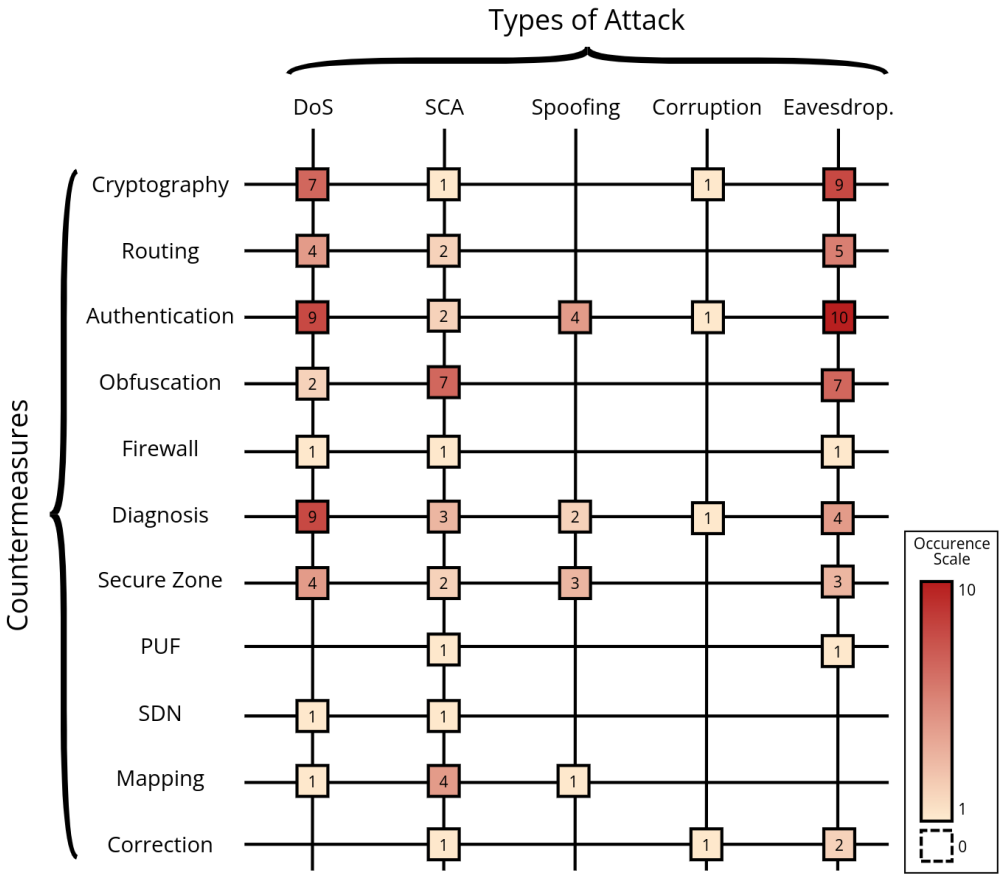



Figure 2.6 – Countermeasure overview according to type of attack from 2023 research.

The protection mechanisms used against DoS also increased. Current works have used cryptography and authentication to validate traffic flows to avoid flooding and confirm IPs' legitimacy to avoid attempts to impair the system functionality. Besides that, more works are focusing on techniques to detect attacks and locate their source, mostly DoS but also other types of attack.

The works of Secure Zone on 2023 research are mostly from our research. Only two other papers, [Benhani et al., 2019] and [Azad et al., 2019], have considered secure zones and were not the focal point of the proposals.

Comparing Figure 2.5 to Figure 2.6, one can see an increase in the total occurrence number. Even though the number of publications on security has increased, it is important to observe that the threat models are getting more complex and countermeasures are evolving, being able to cover more than one attack type.

## 2.4 Criterion 3 - Phase

According to [Caimi et al., 2018a], the application lifetime encompasses three main phases: (a) *Admission*: before execution, the application needs to be loaded into the MCSoC, mapping the tasks and reserving the necessary resources. (b) *Execution*: during the application execution, the computation resources run the algorithms and may use communication resources to inter-task communication; (c) *IO Access*: some applications may need access to external devices connected to an IO port. Thus, it is possible to classify threats and countermeasures according to the execution phase.

### Application Admission

The main requirement of the application admission security is the protection of the source code during its transmission from external devices to the PEs. Defense mechanisms adopted at this phase include secure boot [Siddiqui et al., 2019], application deploy [Real et al., 2018], and source code integrity [Sepúlveda et al., 2018].

The application admission proposed by Real et al. [Real et al., 2018] adopts a logical and spatial isolation of sensitive applications by creating secure zones to mitigate DoS and cache SCA attacks at runtime. The architecture uses the MPSoCSim, a Mesh NoC where each router is connected to a cluster with four processors (with local memory), one shared memory, and one shared bus.

Caimi and Moraes [Caimi and Moraes, 2019] adopt in the OSZ technique mutual authentication based on ECDH protocol to guarantee the authenticity of the entity responsible for deploying the application into the MCSoC (application injector). Furthermore, the source code transferring is protected by encrypting the message and attaching a MAC to it, verified by the receiver PE to guarantee the code integrity.

Siddiqui et al. [Siddiqui et al., 2019] propose the “Multilayered Camouflaged Secure Boot” technique, targeting FPGA devices. The goal is to mitigate the leakage of the bitstream contents by encrypting it with a key only available to remote trusted servers. The FPGA only accepts the bitstream with a correct authentification. In addition, a PUF is used to generate unique keys for each device.

### Application Execution

Most works in the literature focus on protecting the execution phase since this is the phase where applications use the system resources (NoC, processors, NI, memory). This category includes proposals protecting the communication resources, the computation resources, or both.

Examples of defense mechanisms at the communication level: Trust-based Routing [Charles and Mishra, 2020a], that guarantees the delivery of packets avoiding suspicious routers; proposal of a response protocol that indicates the correct packets delivery [Daoud and Rafla, 2019b], Anonymous Routing that obfuscates via cryptography the source, target, and content of the packet, asserting that the information exchanged between tasks is not accessed by other tasks [Charles et al., 2020a].

In terms of protecting specifically the task execution, Zhao et al. [Zhao et al., 2020] discuss a Power Budget attack that is able to alter the values of power given to cores. Therefore, the execution of an application can be severely damaged if such an attack lowers the power resources of the core. As a countermeasure, the authors suggest the monitoring of the power budgets that differ from an average threshold.

Reinbrecht et al. [Reinbrecht et al., 2020] present Guard-NoC, a communication and computation protection against SCAs, specifically Timing Attacks. The protection is based on the *Obfuscation Module*, placed between the NI and the local IC input. This module uses two processes to prevent timing attacks: blinding and masking. The blinding strategy implementation changes the response time of the ICs to have a constant value. Masking is applied to insert delays on the responses, operating as a noise source. Both strategies are effective against the attempts to read or access time measurements of that local IC. In addition to this obfuscation mechanism, the Guard-NoC protects the system's communication via the switching mechanism. This NoC has dual switching, the packet switching is reserved to secure communications. Consequently, the circuit switching is destined for common packet transmission. The separation of secure and common traffic prevents the attackers from injecting malicious traffic directed to collide against secure traffic and infer timing values about a protected communication.

### Communication with IO devices

Relative to the communication with IO, unauthorized access to instructions and data in shared memory and peripherals can compromise the execution of the applications, resulting in attacks of information tampering or information leakage. For this reason, it is important to have security protocols to communicate with IO devices, protecting the many-core against malicious peripherals. One example of a countermeasure focused on protecting the communication with IO devices is the *IO Secure Communication Unit* (IOSCU) (previously detailed on Authentication) [Kornaros et al., 2018]. This work also discusses the security challenges of adding new modules in an SoC-based automotive design.

Another type of IO is external memories attached to the MCSoC. In this case, works that propose defense mechanisms to protect cache level 2 (L2) or lower memory layers, such as [Ge et al., 2019, Reinbrecht et al., 2019], are also related to the IO communication.

### 2.4.1 Discussion

This section demonstrates that most works consider only one of the application execution phases, limited to the application execution (computation or communication protection) and the memory access. The concern about memory access is considered from the communication point of view. Proposals regarding the application admission and the access to peripherals are scarce in the MCSoC research field. A low-cost protocol for secure application admission and communication with external devices targeting MCSoCs is still an open research problem.

## 2.5 Criterion 4 - Cost

Every implementation, either threat or countermeasure, has an innate cost measured in physic attributes (e.g., area or power), application performance (e.g., execution time), or communication efficiency (e.g., latency). This section presents the costs related to the insertion of the security methods.

Table 2.1 summarizes works that presented the costs and trade-offs related to their proposal. The first columns show the authors enumerated from #1 to #22, followed by the proposal characteristics. The other six columns correspond to the overheads: area, power, performance, latency, energy, and critical path delay.

The values represent the overhead of the proposal and the reference for comparison in parenthesis. As some works focused on improving already existent techniques, the values with the minus sign represent a decrease in the overhead. The performance column is a generic term that includes different measurements, such as execution time (ET), bandwidth (BW), throughput (TP), depending on the authors' presentation. Besides that, when there is the indication of *others*, means a comparison between the proposal and another proposal of the literature.

This section aims to present an overview of how the authors evaluate the costs and trade-offs related to countermeasures and threats, and not a direct comparison between the state-of-the-art. Besides that, not all the works are present in the table because: (*i*) did not present the costs clearly, or (*ii*) presented results could not be summarized in this table (e.g., presentation of several graphs instead of tables or measures).

#### Area, Power and Energy

The silicon **area** is the primary metric for evaluating the impact of either the threat or the countermeasure. Most authors present a relative area evaluation, comparing their

Table 2.1 – Cost of the security proposals for MCSoCs.

| # | Work | Proposal | Area | Power | Performance | Latency | Energy | Crit.Path |
|---|---|---|---|---|---|---|---|---|
| 1 | [Charles and Mishra, 2020b] | Cryptography | 2% (NoC)<br>15% (Crypt) | | -5% (crypt. ET) | | | |
| 2 | [Reinbrecht et al., 2020] | Obfuscation<br>Secure Zone | 16% (Router)<br>0.9%(System) | +18% (Router)<br>+0.9% (System) | 2.77%(Blinding)<br>13.45% (Masking) | | | |
| 3 | [Charles and Mishra, 2020a] | Adapt. Routing | 6% (Router) | | -4.7%<br>(vs. Attack) | -43.6%<br>(vs. Attack) | -28.3%<br>(vs. Attack) | |
| 4 | [Charles et al., 2020a] | Adapt. Routing | | | 4% (ET) | | | |
| 5 | [Caimi and Moraes, 2019] | SDN | 6% (NI) | 4% (NI) | | 19.42% (High TP)<br>7.1% (Med. TP)<br>0.9% (Low TP) | | |
| 6 | [Charles et al., 2020b] | Detection | 6%<br>(Router) | 4%<br>(Router) | | | | |
| 7 | [Benhani et al., 2019] | Cryptography<br>Secure Zone | 0.93 - 3.47%<br>(System) | | 15.3$\mu$s<br>(crypt. ET) | | | |
| 8 | [Daoud and Rafla, 2019b] | Adapt. Routing | 10.83%<br>(Router) | 27.78%<br>(Router) | -21.31% (TP) | | | |
| 9 | [Ho et al., 2019] | Obfuscation | 1.89 mm$^2$ | | | | 24.6pJ<br>9 core plat. | |
| 10 | [Daoud and Rafla, 2019a] | DoS HT | 1.98%<br>(Router w/ HT) | 0.74%<br>(Router w/ HT) | | | | |
| 11 | [Caimi and Moraes, 2019] | Secure Zone<br>Adapt. Routing | | | 12% - 15%<br>(ET w/ IO) | | | |
| 12 | [Azad et al., 2019] | Cryptography<br>Secure Zones<br>Firewall | 277.6% (NI)<br>18% (NI) | | | | | 17.80% (NI) |
| 13 | [Ravikumar et al., 2019] | Detection<br>Adapt. Routing | | 1.4 mW | | 200 cc | | |
| 14 | [Raparti and Pasricha, 2019] | Firewall | 2.2 $\mu$m$^2$ | 5.5% (NI) | -48.7%<br>(ET w/ attack) | -67.8%<br>(ET w/ attack) | -47.8%<br>(System) | |
| 15 | [Harttung et al., 2019] | Cryptography<br>Authentication | 4.4%<br>(vs. others) | | -40%<br>(data Rate) | -10%<br>(System) | | |
| 16 | [Chaves et al., 2019] | Monitoring CPRD | 17.7% | 5% | | | | -.02 ns |
| | | Detection CPDD | 23.2% | 9.4% | | | | -.03 ns |
| 17 | [Hussain and Guo, 2019] | Authentication<br>Cripto | -56%<br>(vs. others) | | -36%<br>(BW overhead) | -11.66% | | |
| 18 | [Azad et al., 2018] | Cryptography<br>Firewall | 0.8 mm$^2$ | | | | | 6.46% (NI) |
| 19 | [Zhang et al., 2018] | DoS HT | | | 52%<br>packet loss | | | |
| 20 | [JYV et al., 2018] | Obfuscation | 21.2%<br>(Router) | | | 150% (HT)<br>10% (CT) | | |
| 21 | [Hussain et al., 2018] | Detection | | | -40% (vs others) | 13% (EETD1)<br>19% (EETD2) | -38%<br>(vs others) | |
| 22 | [Caimi et al., 2018b] | SZ<br>Adapt. Routing<br>Mapping | | | 94.2K cc (ET) | | | |

(BW) Bandwidth; (TP) Throughput; (ET) Execution Time; (cc) Clock Cycles.

proposal before and after the implementation. The hardware modules evaluated are the routers and the network interfaces (NI). For example, [Azad et al., 2019] (#12[1]) presented the highest relative overhead, 277.6%, which is expected from a firewall that is a complex mechanism to include in the NI. Other works compare their proposal to the state-of-the-art. [Hussain and Guo, 2019] (#17) reduced in 56% the area of authentication schemes compared to proposals available in the literature. In terms of threat evaluation, [Daoud and Rafla, 2019a] (#10) presented an area overhead lower than 2% when implementing a DoS-HT in a router.

**Power** evaluation is also a performance metric broadly used by the authors in their work. As in the area evaluation, authors measure the power comparing their proposal to the baseline one [Caimi and Moraes, 2019, Raparti and Pasricha, 2019, Reinbrecht et al.,

[1] First column of Table 2.1

2020, Charles et al., 2020a] (#11,#14,#2,#4). In other cases, the authors present the implementation absolute value [Ravikumar et al., 2019] (#13).

Hussain et al. [Hussain et al., 2018] (#21) have **energy** as the major concern, as it proposes a runtime HT detection that is energy-efficient. They claim that always-on countermeasures are expensive in terms of energy. For this reason, their proposal is a countermeasure that is only activated after an HT attack is detected, being powered off otherwise. Results show a reduction of 38% in energy consumption compared against another detection mechanism available in the literature. Ho et al. [Ho et al., 2019] (#9) measured the total energy consumption of a 9-core platform with the countermeasure, resulting in 24.6 pJ overhead.

### Performance

The authors use performance to measure the proposal's impact on the application execution time, frequently using benchmarks. As mentioned, this column also presents measurements that impact the overall performance, such as *bandwidth* (BT), *thropughput* (TP), and other time measurements and comparisons made by the authors. Charles and Mishra [Charles and Mishra, 2020b] (#1) show that incremental cryptography can reduce the encryption time by 5%. Reinbrecht et al. [Reinbrecht et al., 2020] (#2) measured the impact on the execution time of both obfuscation techniques: 2.77% in blinding and 13.45% in masking. Caimi and Moraes [Caimi and Moraes, 2019] (#11) presented an increase of 12 to 15% in the execution time of applications inside an OSZ to perform IO communication.

### Latency and Delay

Latency and delay are measurements used to analyze the impact of the implementation on communication. Those metrics are helpful to evaluate trade-offs, e.g., modifying routing algorithms such as Trust-Based Routing proposed by [Charles and Mishra, 2020a] (#3). In this case, the insertion of extra layers of security increased the packet latency compared to the non-secure routing. However, that same comparison in an attack scenario reduced the latency caused by the attacks by 43.6%. Another example is the proposal of [Harttung et al., 2019] (#15), which measured the extra delay in the delivery of messages due to the message encryption procedures, reaching 10% overhead.

### Critical Path

The critical path is the key attribute that constrains the range of operating frequency of the circuit. Implementing extra circuits, either attacks or countermeasures, may insert more delays throughout the critical path of the circuit, limiting its operating frequency. Chaves et al. [Chaves et al., 2019] (#16) implemented two routers that reduced the critical path delay

by 0.02 and 0.03 ns. Azad et al. also analyze the critical path, as presented in [Azad et al., 2019, Azad et al., 2018](#12 and #18), reducing it by 17.8% and 6.56%, respectively.

### 2.5.1 Discussion

This review analyzed the approaches taken by the authors to evaluate the costs of their proposals. We identified two methods to evaluate the proposals:

1. *relative*: results are compared with other/previous versions of the same platform. For example, compare an IP with and without a firewall. This evaluation gives limited information about the method since it is a function of the baseline implementation.
2. *absolute*: results such as power, performance, and area are given to the implemented modules. This approach is helpful for designers since they can evaluate the impact of adding a given module to the system.

Note that the Table does not present a *Security* column. Security is not directly measurable, lacking in the literature benchmarks to evaluate attacks. Developing benchmarks for evaluating security in NoC-based systems is an open research field.

## 2.6 Criterion 5 - Integration

The countermeasures insertion can be classified according to when its integration on the system occurs, which can be at design time or runtime. Design time proposals assume methods that cannot change at runtime, i.e., it is not possible to configure the countermeasures according to the system state. Despite requiring design-time support, runtime approaches can configure the countermeasures according to the system state, such as changing routing paths or the location of secure zones.

#### Design Time

In design-time approaches, the configuration and activation of the countermeasure is a static process, and it is done during the design phase and is not modified at runtime. An example of design time countermeasure is the incremental cryptography proposed by [Charles and Mishra, 2020b], which is an encryption block placed at the NI, that is always encrypting packets, regardless of the packet length or type. PUF is another example of design time countermeasure, which can only be inserted into the system at design time.

Halder et al. [Halder et al., 2023] propose ObNoC, which obfuscates hardware to avoid reverse-engineering attacks. The ObNoC routers are designed with configurable

switching architectures that are programmed after fabrication and only for authorized holders of the activation packet that activates the functional configuration of the routers.

Runtime

Runtime countermeasure includes methods that can be configured while the system is running. Furthermore, they are adaptable, meaning that the current state and workload of the platform are taken into consideration in the countermeasure decision-making process. Runtime countermeasure is the category of most works. The proposals that include detection and monitoring, such as [Chaves et al., 2019, Kenarangi and Partin-Vaisband, 2019] are always runtime since they are constantly observing the system and making decisions based on what is observed. Proposals that protect the resource allocation, such as secure zones [Caimi et al., 2018a] are runtime since they analyze the availability of the resources to allocate and protect sensitive applications.

## 2.7 Taxonomy Final Remarks

Previous sections proposed a taxonomy for the MCSoC security research field. This proposal contains five criteria related to the works available in the literature: attacks, countermeasures, the application phase, costs, and integration moment. The literature review considered the most recent publications.

In terms of attacks, the research showed that recent proposals explore HTs to perform attacks on MCSoCs, proposing countermeasures to these attacks. The relevance of detecting HT attacks, mainly DoS, is that HTs are difficult to detect and may cause severe damage to the system.

Among the countermeasures, secure zones and SDNs are defense mechanisms that protect the system against a wider range of attacks, being effective against DoS, SCA, Eavesdropping, and Spoofing. It is important to mention that secure zones and SDNs also require a more complex platform in terms of hardware and software to promote spatial isolation.

The security mechanisms must protect all application phases, including its admission, execution, and communication with IO devices. Most works in the literature present countermeasures only for the application execution, with few proposals encompassing this requirement.

One question raised in the cost section is “how to measure security?”. Works presented different methods to measure the efficiency of the security approaches, but most of them are only comparable to their context since the threat models and countermeasures

are very specific, i.e., compared to a baseline implementation. Benchmarks for evaluating security in NoC-based systems are thus an open research field.

The integration of security methods must occur at runtime. The MCSoC must provide the security mechanisms at design time (e.g. encryption IP, isolation wrappers, control NoC), but these must be configurable at runtime to adapt the system according to detected attacks.

In this Thesis, we propose mechanisms based on secure zones, authentication, and detection to protect the MCSoC against malicious IO devices and malicious hardware performing DoS, Spoofing, and Eavesdropping.

Our approaches protecting IO (Objectives *SG1* to *SG4*) are motivated by the lack of works that consider the protection of IO communication, especially considering the use of secure zones.

One can see the concern around hardware trojans on MCSoCs, which was still an open vulnerability even considering the secure zones. Because of that, we propose a detection mechanism of hardware trojans that protects the communication inside the secure zone (Objectives *SG5* and *SG6*).

During this research, we observed a broad range of threats and countermeasures. We proposed a taxonomy to map the research results to help understand the areas and subareas of the MCSoC security research field. We noticed that most papers proposed countermeasures focusing on specific threats, with a limited number of works that approach complex threats integrating several countermeasures. In addition to that, Charles and Mishra [Charles and Mishra, 2022] pointed to the integration of security mechanisms as a research direction in the conclusion of their survey on NoC security and countermeasures:

> **Seamless integration of security mechanisms:** While existing literature has discussed several different threat models, it is naive to think that mitigating one particular type of threat will secure the SoC. For example, defending against eavesdropping attacks does not guarantee that eavesdropping is the only possible attack in that particular architecture. Developing security mechanisms for different threat models is a promising starting point. However, seamless integration of a suite of security mechanisms is required to secure the hardware root of trust. [...] Similarly, how to integrate several NoC security mechanisms and ensuring their inter-operability in hardware, firmware, and software layers is worth exploring further.

Hence, the main original contribution of this Thesis (Chapter 6) is the integration of defense, monitoring, and detection mechanisms that protect the MCSoC from a diverse set of threats.

## 2.8 Comparative Analysis and Positioning within the State-of-the-Art

This section discusses related work on MCSoC security. During the taxonomy we placed and discussed several works, however most of they focus on specific counter-

measures against specific attacks. In addition, many of them assumed threat models very different from the ones in this Thesis. For those reasons, for Thesis placement, we focused on proposals that integrate multiple countermeasures and propose systemic level security.

In a previous study, Fiorin et al. [Fiorin et al., 2007] emphasized the need of a security framework designed to gather data from monitors integrated into network interfaces (NI) or routers strategically positioned within critical areas of the NoC. The authors propose to monitor: (*i*) buffer occupancy, (*ii*) anomalous behavior of power manager; (*iii*) unauthorized access to secure memory locations; (*iv*) violation of execution of critical routines. These authors propose in [Fiorin et al., 2008] the adoption of firewalls integrated into the NI to manage memory accesses using a lookup table containing the access rights. The authors only evaluate the firewall area. In more recent work, Fiorin et al. [Fiorin et al., 2013] propose the insertion of a configurable Probe device inside the NI that can detect events and collect values about throughput, latency, resource utilization, and message characteristics. After detecting events, a message to the Probe Management Unit (PMU) reports the detected set of events that can trigger runtime management functions. The probe module was evaluated for area, energy, and traffic overhead.

Meng et al. [Meng et al., 2023] propose a framework for systematically detecting security violations in SoC designs resulting from vulnerabilities in NoC communication. The threat model includes message misdirection, message mutation, delivery prevention, and network congestion. The proposed framework, SEVNOC, extracts a control-flow graph of the design that enables analysis of security properties through state exploration. The framework does not detect attacks at runtime. The authors' goal is to detect vulnerabilities in the RTL design using a symbolic approach.

Sharma et al. [Sharma et al., 2021] analyze the security aspects of MPSoCs, discussing several defense mechanisms known in the literature, such as secure zones, firewalls, and key agreement, and then expand the discussion to the Cloud of Chips scope. Furthermore, the authors propose a software-defined network-on-chip (SDNoC) as an alternative that can reserve resources, avoiding congestion and harmful paths. The paper provides a broad view of MPSoC security and how effective are the defense mechanisms against DoS, Hardware Trojan, and Side-channel attacks. However, the proposal does not include runtime monitoring of threats, which results in a gap in detection and countermeasures once a threat is detected.

Ruaro et al. [Ruaro et al., 2020] also adopt SDN to establish a programmable path based on different policies, such as power, QoS, and security. In addition, the authors propose a secure path configuration based on key authentication that avoids DoS and flooding attacks since packets that fail the authentication are discarded. However, the authors point out that their approach is still vulnerable to HT attacks.

Kumar et al. [Kumar et al., 2021] propose a methodology to protect NoCs against HTs. The authors propose a 3-tier approach that includes a Trojan cognizant routing al-

gorithm (TCRA), a Trojan detection and diagnosis module, and a Trojan-resilient network interface. The detection and diagnosis module is responsible for identifying and locating HTs in the system, while the network interface provides a secure communication channel between the NoC and the external world. The authors used a NoC simulator (NoCTweak) to test the TCRA under different scenarios, including single and multiple HTs. The results of the experiments show that the proposed approach effectively mitigates the impact of HTs on NoCs. The TCRA outperforms other methods regarding average throughput, packet delivery, and free link availability.

Table 2.2 – Positioning within the state-of-the-art in security frameworks for NoC-base many-cores.

| Author | Security Location | Defense Mechanism | Monitoring | Detection | Countermeasure |
|---|---|---|---|---|---|
| [Fiorin et al., 2008] | NI, to protect external shared memories | Data Protection Unit (DPU), for memory access control | Memory access rights | Correctness of access rights | Packet discarding; negative acknowledgment to the initiator |
| [Fiorin et al., 2013] | NI | — | Throughput, Latency, Resource Utilization | — | — |
| [Meng et al., 2023] | SoC | Security analysis at design-time of CFGs (control-flow graph) | — | — | — |
| [Sharma et al., 2021] | SND for MPSoC and Cloud of Chips | SDNoC-based security, security-aware routing | — | — | — |
| [Ruaro et al., 2020] | NoC (SDN) | Secure SDN configuration; sub-network authentication | Key authentication for path configuration packets | Wrong authentication key | Packet discarding |
| [Kumar et al., 2021] | Router and NI | Trojan cognizant routing algorithm (TCRA), Trojan-resilient network interface. | — | Error detection code (ECD) | Routing algorithm and ECD |
| **This Thesis** | PEs, NoC and NI with IO devices | Opaque Secure Zones (**OSZ**), authentication, adaptative routing, secure NI for IO devices | Session protocol in OSZs, master-slave comm. protocol, packet authentication | Missing packets, unexpected data, fail key auth., access attempts | Reroute, key renewal; packet discarding, **warnining for a security manager** |

Table 2.2 provides a qualitative comparison of the works discussed in this section. The column **Security Location** denotes the system components endowed with security mechanisms. Most proposals focus primarily on protecting the communication infrastructure, encompassing NoC and NI components. In addition to employing these components, our work uses PEs to monitor and configure NoC and NI peripherals at runtime.

The column **Defense Mechanisms** presents the security mechanisms adopted by the authors. Our approach is distinct due to adopting opaque secure zones, which reserve

communication and computation resources for a specific application, together with secure mechanisms for communicating with peripherals.

The **Monitoring** column lists the policies used for system monitoring. Here appears a gap in current literature, as fundamental mechanisms are firewalls or authentication, lacking systemic mechanisms. The **Detection** column is a consequence of the monitoring methods. As discussed later, our work monitors multiple events simultaneously, enabling the detection of a broader range of security events.

Lastly, the **Countermeasure** column shows that the most common countermeasure is packet discarding, followed by rerouting. Besides incorporating these countermeasures, our work notifies a Security Manager about suspicious events. This enables this it to know the system's status and implement suitable measures to ensure its secure operation.

The reviewed proposals on security for NoC-based systems addressed frameworks and mechanisms to treat different security threats. Despite these advances, the integration of security mechanisms still needs to be improved, enabling a security manager to make decisions using monitoring data to mitigate threats more effectively. This integration is the primary goal of our work, aiming to create a comprehensive framework for security management.

# 3. BACKGROUND KNOWLEDGE

This Chapter explains the concepts required to follow this Thesis. Section 3.1 presents the baseline NoC-based MCSoC. Section 3.2 presents the Opaque Secure Zones (OSZ), the central security method used in this Thesis. Section 3.3 presents the legacy communication proposal between OSZs and IO devices, which will be modified to mitigate security issues. The Secure Network Interface for Peripherals (SNIP) is detailed in Section 3.4. Finally, in Section 3.5, we present examples of attacks that Hardware Trojans can carry out.

## 3.1 Baseline platform

The baseline MCSoC utilized in this work is the Hermes MultiProcessor System (HeMPS) [Carara et al., 2009]. The main HeMPS platform features are presented on Figure 3.1:

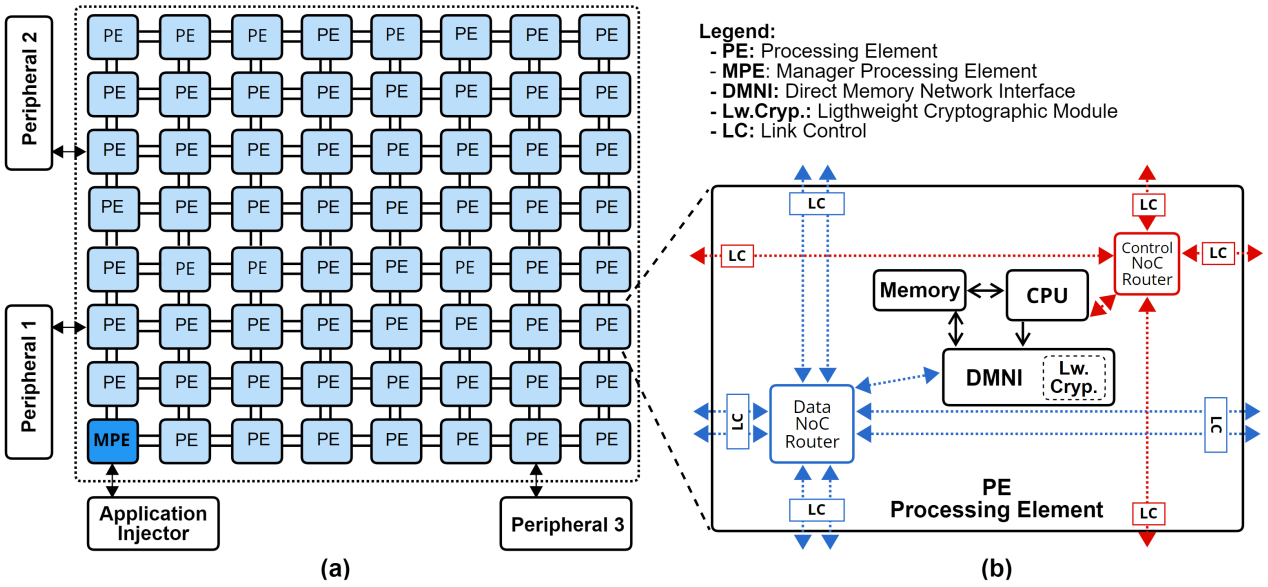



Figure 3.1 – NoC-based MPSoC. Link controls (LC) are added to the control signals of NoCs links, enabling to isolate ports individually.

- NoC-based system: the platform contains two NoCs. A PS (Packet Switch) data NoC and a control NoC. Both adopt 2D-mesh topology. The data NoC uses duplicated physical channels, enabling the adoption of distinct routing algorithms per physical channel (detailed in Section 3.1.1). The control NoC uses broadcast as the default transmission mode (detailed in Section 3.1.2). The two NoCs are completely disjointed without hardware or software dependence in their accesses.

- Homogeneous system: all PEs have the same hardware architecture (Figure 3.1(b)) with a PS NoC router, a broadcast NoC router, a private memory, a MIPS-like processor, a DMNI (Direct Memory Network Interface) module, a set of link control cells connected to each router link.
- Distributed memory: each PE has a true dual-port scratchpad memory for instructions and data while a message-passing API executes the communication between PEs.
- Applications are modeled as a Communication Task Graph (CTG). The CTG is a model to represent functional parallelism, where an application is composed of independent parts and thus is divided into tasks [Rauber and Rünger, 2013]. A graph node represents each task in a CTG, and the graph edges represent the communication between these tasks (Appendix B).
- Peripherals are connected to the boundaries of the MCSoC [Ruaro et al., 2018], at unused ports of the mesh NoC (e.g., South ports of bottom routers). Therefore, this results in a regular floorplan for PEs, with peripherals distributed along the platform perimeter.

This platform requires at least one peripheral, the Application Injector (*AppInj*). This peripheral transmits the applications' source code to the PEs through the data NoC.

The control flow signals of all links contain link control modules (LC). The function of the LC cells is to isolate a given link. The granularity of the isolation is at the link level. For example, blocking only the west link and continuing to transmit through the other links is possible. The activation of the LCs occurs by the OS using a memory-mapped register at each PE. Each bit of the LC register enables/disables a given link. Thus, the LC's area overhead is negligible since its implementation requires a small number of gates, a register, and an FSM.

### 3.1.1 Data NoC

The data NoC transfers *data messages*, exchanged by applications. The data NoC extends the NoC Hermes [Moraes et al., 2004] adopting duplicated physical channels, flit width equal to 16 bits, input buffering, round-robin arbitration, credit-based flow control, wormhole packet switching, simultaneous support for distributed XY routing and source routing (SR).

Duplicated physical channels ensure deadlock avoidance and full routing adaptivity. The number of virtual or replicated channels required to avoid deadlocks is a function of the network topology. For example, two virtual or replicated channels are sufficient to avoid deadlocks in a 2D-mesh topology [Linder and Harden, 1991]. Due to the duplicated physical

channel adoption, the flit width is half of the original in the Hermes NoC to minimize the area overhead.

The standard routing mode between PEs is the distributed XY routing algorithm. The data NoC also supports SR, which is essential to determine alternative paths to circumvent broken paths due to an *OSZ* or to avoid infected routers. The mechanism to find an alternative path to use in the SR is presented in Section 3.1.2.

A *data* packet, from the NoC point of view, has a header and a payload (Figure 3.2). The *packet header* content controls the data NoC behavior, such as, routing, open and close internal switching, and arbitration. While in [Carara et al., 2009] the packet header have two fields (target and payload size), we adopt three fields to support the SR, the rerouting mechanism and the communication with peripherals: (*i*) the source/target address with data (D) or peripheral (P) packet flag; (*ii*) the XY or SR field that indicate the turns on each router when use SR or the source/target address when use XY routing and; (*iii*) the payload size.

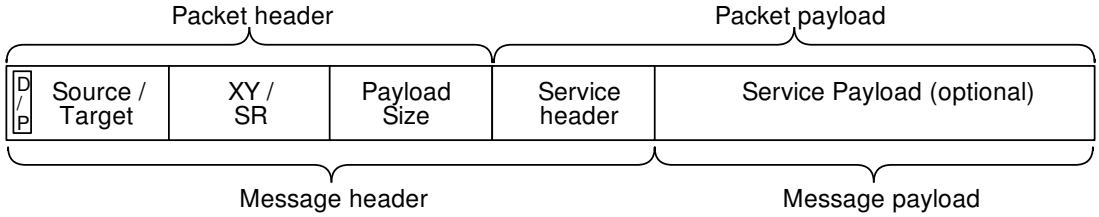



Figure 3.2 – Packet and message structures - a flag (D/P) in the target address field differentiates *data* packets from *peripheral* packets.

The data NoC differentiates *data* packets from *peripheral* packets. Data packets are those exchanged by tasks running in PEs, and peripheral packets are those transferred between a task and a peripheral. A *peripheral* packet arriving in a boundary PE goes to the peripheral, not the DMNI.

From the task point of view, a message is used by the kernel with two fields: (*i*) the *message header* to control the data exchange between tasks or peripherals. The message header has data such as: producer task ID, consumer task ID, service (e.g., message delivery, request for a message, task mapping, task allocation), message timestamp. (*ii*) The *message payload* is an optional field with data to transmit to the task or peripheral. It may contain, for example, user data or the object code of a task.

### 3.1.2 Control NoC - BrNoC

The *BrNoC* [Wachter et al., 2017] is a dedicated NoC, decoupled from the data NoC. The BrNoC has the same topology as the data NoC, enabling the control of each port individually (e.g., the North port in the dedicated NoC has an equivalent North port in the data NoC). The default transmission mode is broadcast because it enables to reach PEs

in case of disabled links, to notify several PEs with one message, and to transmit with low latency control messages.

When a given port receives a message in a broadcast, it is processed and broadcasted to the neighbor routers (ports N, S, E, W), except to the port it came from. The broadcast acts as a wave traveling through the NoC. According to the transmission mode, the message may be transmitted to the port connected to the NI (local port). The BrNoC supports five transmission modes:

- ***brTgt*** (broadcast with a target): a specific PE is the target of this message. The message is broadcasted to all routers, but only the PE with the target address consumes it. This mode may be used to find a new path after a message is discarded and notify a specific PE to execute some action. The broadcast ensures that the message will be delivered even if a link/router is faulty or disabled.
- ***brAll*** (broadcast to all PEs): all PEs consume the message. Therefore, all PEs are interrupted, and the message type defines the action the PE should execute. This mode may be used to freeze the tasks of a given application.
- ***brWt*** (broadcast without a target): all *BrNoC* routers consume the message, without notifying the NIs. This mode executes actions related to the *BrNoC* management, such as clearing specific data structures.
- ***brApp*** (broadcast to application): a specific application is the target of this message. The message is broadcast to all routers, but only the PEs running tasks of that application consume it. The verification is based on a memory-mapped register with the application number *AppReg*. This mode is exclusive for secure applications and is used on key management functions.
- ***unicast***: this message is an answer to a ***brTgt*** message. The ***unicast*** message follows the path defined by the ***brTgt*** message, in the reverse order to reach the source PE (backtrack process). This mode may be used to return a new path. Due to the limited payload size, each *BrNoC* router in the path sends a unicast message to the source router so that the path can be completely received.

Figure 3.3 presents the internal architecture of a *BrNoC* router, for a 2D-mesh topology. The router contains two control FSMs (Finite State Machines), two round-robin arbiters and a centralized CAM (Content Addressable Memory) memory. In addition, routers have a small area footprint since they do not have input buffers (the CAM acts as a buffer shared by all input ports, storing all flits received for all ports), and each flit encapsulates a single message.

Figure 3.4 details the flit structure (37 bits in blue) and one CAM row (51 bits, 37 from the flit plus 14 extra control bits). Each CAM row stores the flit contents (to enable

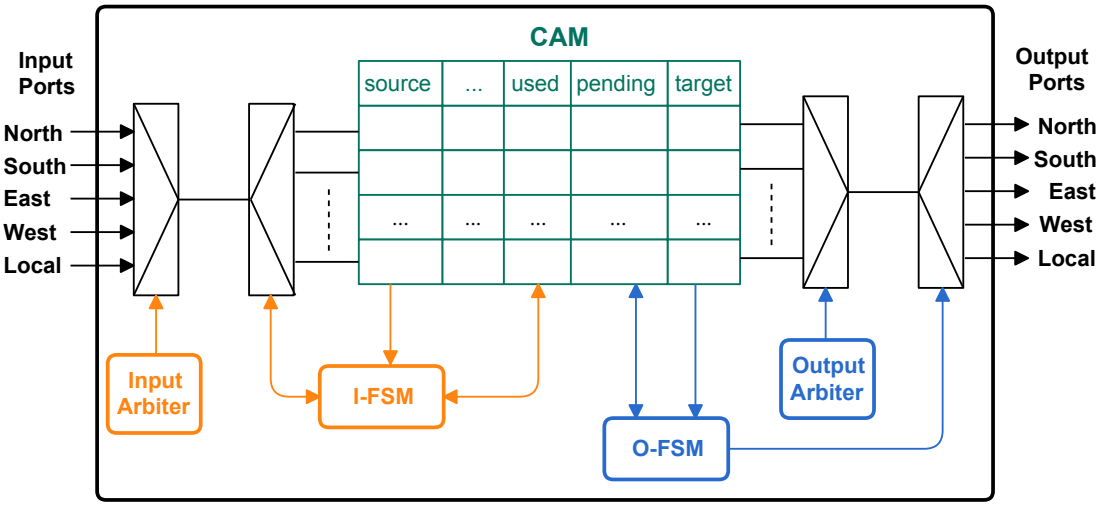



Figure 3.3 – *BrNoC* architecture.

the broadcast) and control fields. The *flit* structure contains the fields: *message ID* (identification); *source address*; *target address*; message *type* (defines the action to execute and the transmission mode); message *payload*. The *tag* to search in the CAM is the tuple *{msg ID, source address}*. Each brNoC link contains the flit structure plus the *req*, *ack* and *nack* signals.

The CAM size definition (number of rows) occurs at design time, and it is not a function of the system size, ensuring scalability. Smaller CAMs can increase the delay in handling the messages, while larger CAMs reduce this delay at the cost of larger silicon area. The payload size may increase at design time to support services requiring larger data to transmit. The payload size is also a trade-off between the amount of data to transmit and the silicon area.

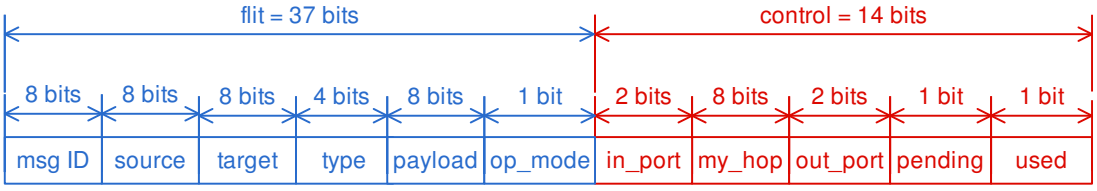



Figure 3.4 – Message (*flit*) and one row of *BrNoC* CAM memory.

The control structure of one CAM row contains the fields: *in_port*, *my_hop*, *out_port*, *pending* and *used*. The *pending* field signalizes the presence of a message to be handled. The *used* indicates that the row is in use. The *in_port* stores the port identification from where the message comes from. The ***unicast*** mode uses the fields *my_hop* and *out_port*.

The control NoC has two operation modes (*op_mode* field): *global* and *restrict*. The *global* operation mode enables the control messages to pass through the LCs, even if they are enabled. This operation mode allows PEs inside a secure zone to exchange messages with the MPE. The *restrict* operation mode observes the status of the LC, i.e., if a

control message hits an activated LC, the message is discarded. This mode enables a path discovery mechanism by the control NoC.

The I-FSM receives incoming messages and, if necessary, stores the message in a CAM row. A handshake protocol (*req*, *ack*, *nack*) controls the I-FSM which is initially in an idle state, waiting for incoming messages (*req* asserted in a given port). The *input arbiter* chooses an input port to handle. Three conditions may assert the *ack* signal: ($c_1$) the *tag* is not in the CAM, and there is space in the CAM; ($c_2$) the *tag* is in the CAM; ($c_3$) failed or isolated port, where an LC force the *ack* signal. The assertion of the *nack* occurs when the *tag* is not in the CAM, and there is no space in the CAM. The router receiving the *nack* unsets the *req* and tries later (action discussed in the O-FSM). When the condition ($c_1$) is satisfied, the I-FSM stores the flit in the CAM and sets the control bits accordingly. Condition ($c_2$) ensures that requests to already visited routers are discarded, avoiding cyclic transmissions (i.e., livelocks) and the end of the broadcast when all routers are visited.

The O-FSM handles the messages stored in the CAM, using the same handshake protocol. The *output arbiter* chooses a row to handle, according to the asserted pending fields. All broadcast modes propagate the message to the neighbor routers, except the *in_port*. According to the broadcast mode, the message also goes to all local ports (***brAll***), or to the local port that matches the router address with the target field (***brTgt***). The *pending* field is cleared when all broadcasted ports answer with an *ack*. If some broadcasted port answer with a *nack* the arbiter selects another CAM row, enabling the selection of the current row again. An example of message type using ***brWt*** propagation is the CLEAR, responsible for freeing a CAM row, by clearing the *used* field. The ***unicast*** message uses the *in_port*, *my_hop* and *out_port* fields to answer a ***brTgt*** message. The ***unicast*** message forwards the message to the port defined in the *in_port* field.

Path discovery using brNoC

Figure 3.5 presents a step-by-step example of a new path finding using the control NoC. In this scenario, Router 1 communicates with Router 15 (XY path), but an *OSZ* interrupts the communication. Using the control NoC, Router 7 starts a TARGET_UNREACHABLE message to Router 1 (not shown in Figure 3.5).

When the message reaches Router 1, it starts a SEARCH_PATH message to find a new path to Router 15 (Figure 3.5.a, red arrows).

Next, Routers 0, 2, and 5 receive the message through ports East, West, and South, respectively. Then, these routers broadcast the received message to their neighbors (Figure 3.5.b). As Router 6 has the LC activated in ports South and East and the SEARCH_PATH message uses the restrict mode, the message is ignored in Router 6 (the input *req* signal is masked, and the output *ack* signal is forced to high).

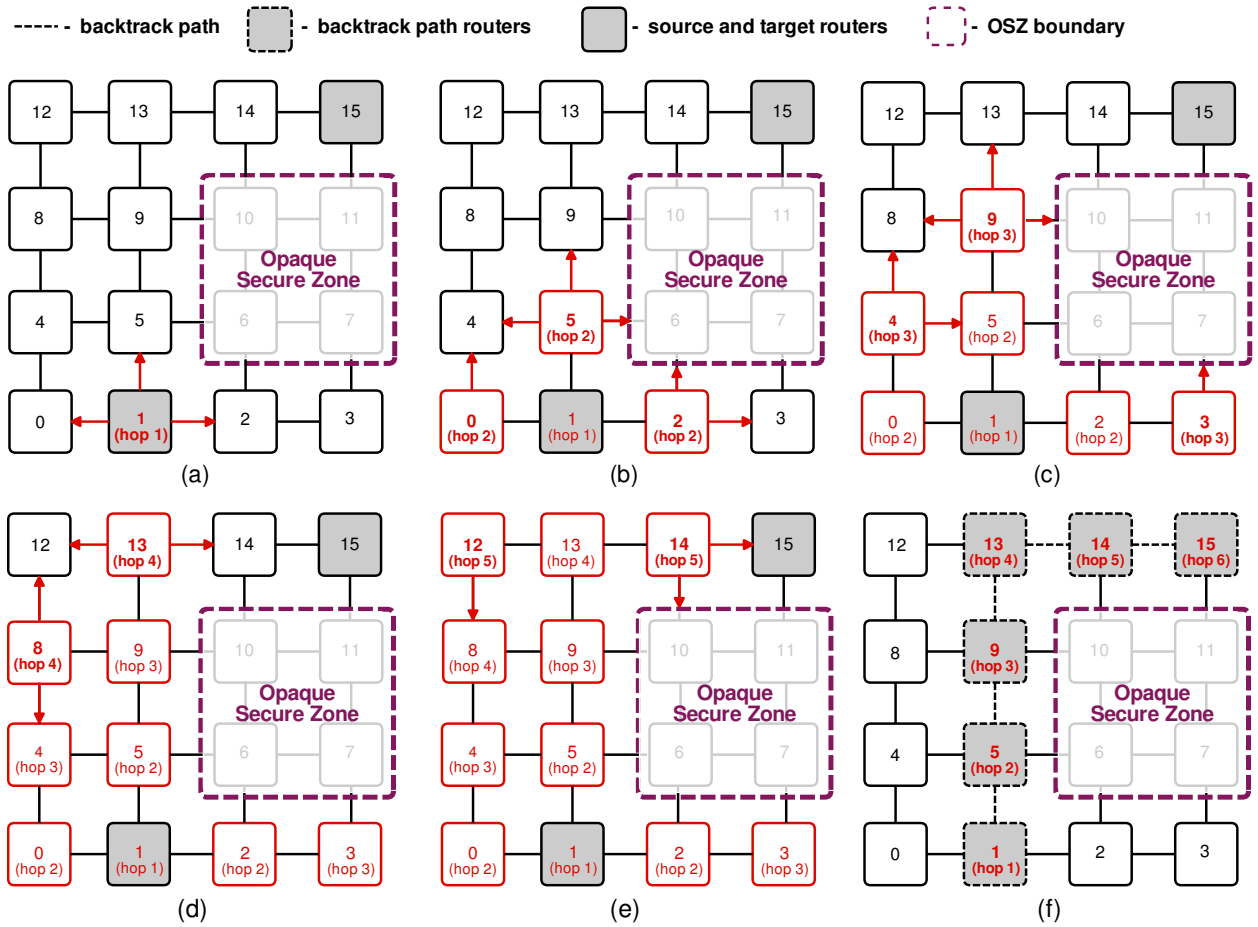



Figure 3.5 – Example of path discovery using the *BrNoC*.

In Figure 3.5.c, Routers 3, 4 and 9 receive the message from ports East, South, and South, respectively, and broadcast to their neighbors. Note that the message sent by Router 4 is discarded in Router 5 because this router has already received the message from the same source (*msg ID*/*source address* stored in CAM). Router 7 ignored the message because the LC is active in the ports South and East, and Router 10 because the LC is active in the ports West and North.

Next, Figure 3.5.d, the message is received in ports East and South of Routers 8 and 13, respectively, and sent to their neighbors. The Router 4 discard the message because it has received the message previously. Router 10 ignores the message from Router 9 because the LC is active in ports West and North.

In the next hop (Figure 3.5.e) the message is received in Routers 12 and 14. Router 12 sends it to Router 8, which ignores the message because it has received it previously. Router 10 ignores the message from Router 14 because the LC is active in ports West and North. Finally, in Figure 3.5.f, the SEARCH_PATH message reaches the target, starting the answer step, with BACKTRACK messages.

In the answer, each router in the path sends a BACKTRACK message to the source router. Initially, Router 15 sends a BACKTRACK message to Router 1 through the West port

(information stored in the *in_port* field). Next, Router 14 propagates the first message, and then transmits a new BACKTRACK message to Router 1, with the payload having the contents of the *out_port* field. Each router in path repeats this process, propagating the previous BACKTRACK messages and sending a new one. The *my_hop* field controls the process, finishing when the source router receives all BACKTRACK messages (*my_hop*=1). Therefore the source router receives a number of BACKTRACK messages equal to the number of hops in the path to the target. Each one of these messages contains the port to reach the destination router in the payload. For example, the source PE (Router 1) receives the following *out_port* values from the BACKTRACK messages: [W E E N N N].

When Router 1 receives a BACKTRACK message, the PE is interrupted to compute a hop of the source routing path to Router 15. After receiving all BACKTRACK messages, Router 1 computes the source routing path and resends the lost message. All subsequent packets to this destination use the source routing path, which is stored in an OS structure. The process to find a new path to a given target is executed once but can be repeated when faults on the current path are detected.

### 3.1.3 Software Model

Scalability at the hardware level comes from PEs executing several tasks in parallel, using the NoC to transmit multiple flows concurrently. However, large systems require high-level management for controlling the deployment of new applications, monitoring resource usage, managing task mapping and migration, and executing self-adaptive actions according to systems constraints. The management of HeMPS occurs in the Manager PE, which has a different kernel from the other PEs.

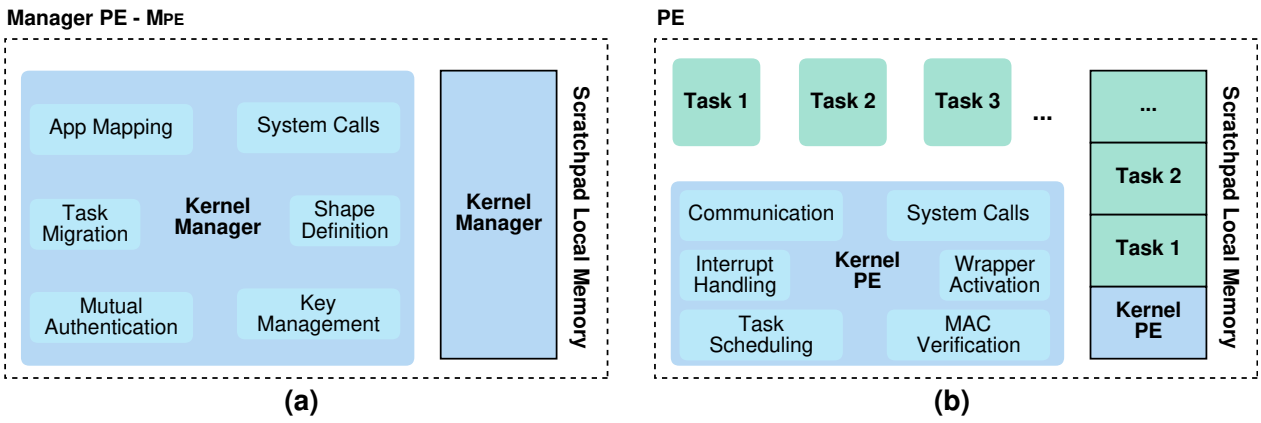



Figure 3.6 – Overview of the kernels: (a) Manager PE kernel controls the system and does not execute users' tasks; (b) Regular PE kernel manages users' tasks.

At the Manager PE level, the local memory is reserved for the kernel without executing the user's tasks. The Manager PE executes activities as task mapping, task migration, monitoring, authentication, and key management (Figure 3.6(a)).

At the regular PE level, a multi-task kernel acts as an Operating System (OS). The platform adopts a paged memory scheme to simplify the kernel design. Examples of actions executed by the kernel include task scheduling, inter-task communication (message passing), and interrupt handling (Figure 3.6(b)).

Both kernels are written in C language. Only a small part of the code is written in assembly language, which is responsible for executing context saving and handling hardware and software interruptions.

Applications are written in C language. They are modeled as task graphs A = $< T, P, D, S >$, where T = $\{t_1, t_2, ..., t_m\}$ is the set of application tasks corresponding to the graph vertices; P = $\{p_1, p_2, ..., p_n\}$ is the set of peripherals corresponding to the graph vertices. The D set represents the application descriptor which contains the communicating pairs $\{(t_i, t_j), (t_i, p_r), (t_j, p_s), ..., (t_m, p_n)\}$ with $(t_i, t_j, ..., t_m) \in$ T, $(p_1, p_2, ..., p_n) \in$ P. A pair $(t_i, t_j)$ denotes the communication from task $t_i$ to task $t_j$ ($t_i \rightarrow t_j$), and a pair $(t_i, p_r)$ denotes the communication from task $t_i$ to peripheral $p_r$ ($t_i \rightarrow p_r$). The S value indicates if the applications execute in normal mode (value 0) or secure mode (value 1). Figure 3.7 presents an application following this model.

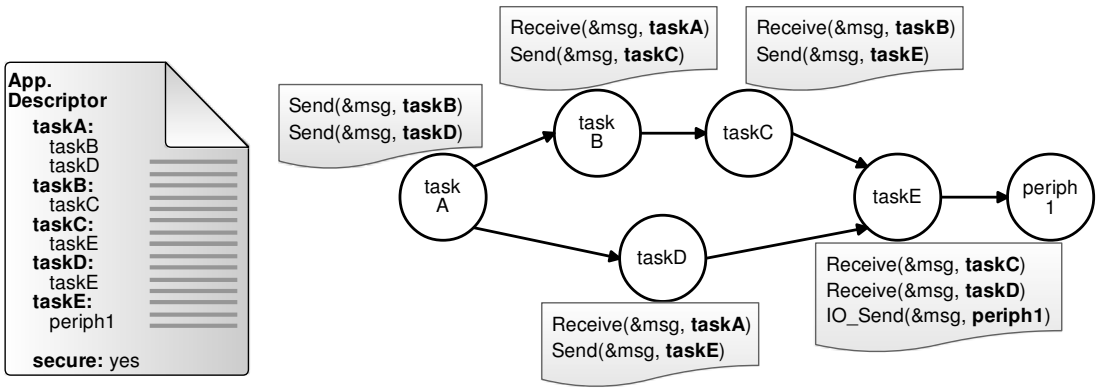



Figure 3.7 – Application task graph example.

Tasks communicate using message-passing (MPI-like) primitives. The API provides two primitives: a non-blocking *Send*() and blocking *Receive*(). The main advantage of this approach is that a message is only injected into the NoC if the receiver requests data, reducing network congestion. To implement a non-blocking *Send*(), a dedicated memory space in the kernel, named *pipe* [Carara et al., 2009], stores each message written by tasks. Within this work, the pipe is a kernel memory area reserved for message exchanging, where messages are stored in an ordered fashion and consumed according to it. Each pipe slot contains information about the target/source processor, task identification, and the order in which it is produced.

At the lower level, the kernel communicates with the data NoC with *data_request* and *data_delivery* packets. The *pipe* and a message buffer enable packet retransmission to inter-task and inter-manager communication, respectively. The support for IO communication uses a second API, with *IO_Receive*() and *IO_Send*() primitives, using a master/slave communication model.

## 3.2 Opaque Secure Zone

Resource sharing is an essential feature of MCSoCs. Different applications may execute in the same processor, share the NoC links, and shared memories. This feature, resource sharing, is the source of issues related to security.

Security methods deployed at design time enable the adoption of sophisticated and robust algorithms to provide solutions to the security problem since they do not have limitations related to the execution time of the heuristics. However, design-time methods do not apply to dynamic workload scenarios. Thus, these methods are limited to scenarios where the workload is known beforehand without changing during the system lifetime.

Secure Zone (SZ) is a runtime approach adopted to limit resource sharing. It is possible to classify such proposals using a set of orthogonal criteria [Caimi and Moraes, 2019]:

- **Creation time:** the definition of the SZ occurs at design time or runtime.
- **Shape:** the SZ may be discontinuous or continuous, with a rectangular or rectilinear shape.
- **Communication sharing:** The SZ may allow flows belonging to sensitive applications to share NoC links, or the flow inside the SZ is forbidden to other applications.
- **Computation sharing:** the SZ may allow tasks belonging to sensitive applications to share the same processor or apply resource reservation to sensitive applications.
- **Methods:** the methods used by the SZs include cryptography, routing algorithms, spatial and temporal isolation, rerouting.

Figure 3.8 presents examples of SZs. Discontinuous SZs ($SZ2$) require more effort to prevent attacks (encryption or routing schemes) due to the exposure of the flows. In contrast, continuous SZs can imply internal fragmentation when using a rectangular shape due to the reservation of resources without effective use ($SZ1$). A rectilinear shape ($SZ4$) prevents internal fragmentation but needs dedicated routing mechanisms to avoid flows crossing the boundary of the region.

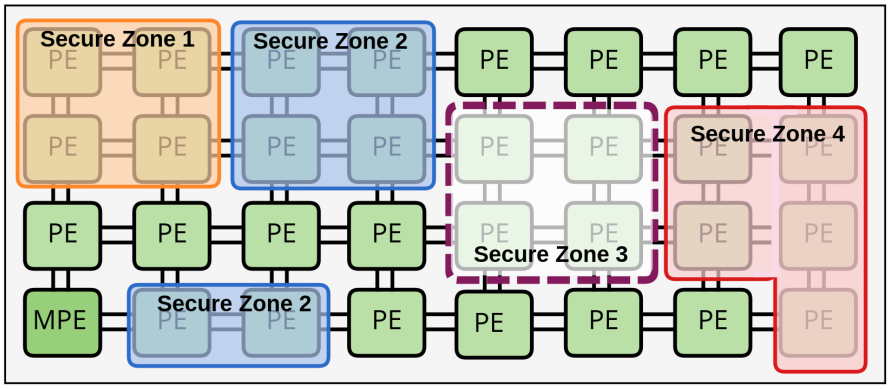



Figure 3.8 – *SZ*1: continuous and rectangular, *SZ*2: discontinuous, *SZ*3: continuous, rectangular, and opaque, *SZ*4: continuous and rectilinear. Source: [Caimi and Moraes, 2019].

Continuous SZ (*SZ*1 and *SZ*4) still exposes the communication to attackers because flows belonging to other applications can transverse the SZ, allowing DoS, HT, and timing attacks.

While some works define a secure zone by building routing paths that never cross through certain PEs [Fernandes et al., 2016], the **Opaque Secure Zones** (OSZs) creates rectilinear shapes at runtime without computation and communication resource sharing (*SZ*3, in Figure 3.8). They are named **Opaque** due to the isolation of the communication by setting barriers that block traffic at the secure zone borders. The PEs of the *OSZ* are reserved for running a single secure application. The only resource-sharing exception is communication with I/O devices, which is further discussed on Section 3.3.

According to the previous classification, **Opaque Secure Zones** (OSZs) are created at runtime and have a rectilinear shape without computation and communication resource sharing. The PEs of the *OSZ* are reserved for running a single secure application (*SZ*3, in Figure 3.8). The only resource-sharing exception is communication with I/O devices, which is further discussed on Section 3.3.

### OSZ API

The OSZ method promotes the spacial isolation of applications by allocating, at runtime, a specific region of the system to execute an application with security constraints, hereinafter referred $App_{sec}$.

**Definition 1.** *Secure Application ($App_{sec}$)* is an application allocated and executed using security mechanisms.

The OSZ method starts when the entity responsible for deploying new applications into the system, the Application Injector, requests the execution of a new $App_{sec}$. The OSZ method has four main phases:

- **Shape Creation**.
  Defines a set of rectangular shapes, with enough PEs to execute all $App_{sec}$ tasks. The shape definition is a function of the number of $App_{sec}$ tasks, and the number of tasks that PEs can execute. For example, a 4-task $App_{sec}$ with PEs executing 1 task, possible shapes are: 4x1, 1x4, 2x2, 3x2 (with 2 PEs in excess).
- **OSZ Location**.
  This phase searches the entire system for a set of free PEs, according to the shape set defined in the previous step. Free PEs are PEs not executing tasks. One constraint of the OSZ method is to avoid sharing the $App_{sec}$ with other applications (spatial isolation). Without a region with free PEs, this phase evaluates if task migration can free a set of PEs to map $App_{sec}$ in a given shape.
- **OSZ Setting**.
  Once defined the region that will receive $App_{sec}$, the system manager executes the task mapping in this region, and the Application Injector sends the object code of each task (including a MAC to ensure the task integrity). After this step starts the third OSZ method phase: OSZ setting. This phase comprises three actions: (1) activate the LC to close the secure region; (2) remove PEs in excess from the region, if any; (3) notify the system manager that the OSZ has been closed.
- **OSZ Unsetting**.
  When $App_{sec}$ finishes, all tasks send a message to the system manager, which will send control messages to the PEs to release the OSZ. This phase comprises two actions: (1) clear the task memory space to avoid data leakage; (2) open the LC.

The complete sequence diagram and in-depth explanation of the OSZ API functions is found on Appendix D

## 3.3 IO Communication

The OSZ proposal [Caimi, 2019] supports communication with IO devices. Chapter 4 discusses the pros and cons of the proposed IO↔OSZ communication, presenting a new method to solve issues of the original proposal.

As mentioned in Section 3.1.3, the original OSZ support for IO communication uses a dedicated API with a master/slave communication model, with *IO_Receive*() and *IO_Send*() primitives. The PE is the communication master, and the peripherals are the communication slaves. At the lower level, the kernel communicates with the data NoC with `IO_REQUEST`, `IO_DELIVERY`, and `IO_ACK` packets. The *IO_Receive*() primitive sends the `IO_REQUEST` from the PE side, and the peripheral answers with an `IO_DELIVERY` packet. The

*IO_Send*() primitive sends `IO_DELIVERY` from the PE, and the peripheral answers with an `IO_ACK` packet.

However, in terms of hardware, an OSZ blocks all incoming and outgoing messages. Thus, IO communication requires selective control to send packets to peripherals and receive packets from peripherals. This control is performed by the **LC control module**, which is able to configure masks to the LC, allowing the controlled passing of packets through the OSZ borders. The masks are memory-mapped registers but can also be configured through data packets, so PEs not close to the border can open the LC without interrupting other PEs.

Figure 3.9 presents an example of a PE inside an OSZ communicating with a peripheral using the default XY routing algorithm. For each message exchange with a peripheral, the communicating PE first sends two configuration messages to the boundary of the OSZ, to set the LC mask registers. When the mask configuration message arrives at the target router, the LC control module intercepts the message (i.e., this message is not consumed by the PE) to set the mask value (input mask or output mask value). The mask configuration messages contain the direction (input or output) and the port side to mask (e.g., north). Once an EOP (end-of-packet) is received, the opened port (input or output) that received the packet is closed. This mechanism ensures that the secure zone receives only one packet for each request.

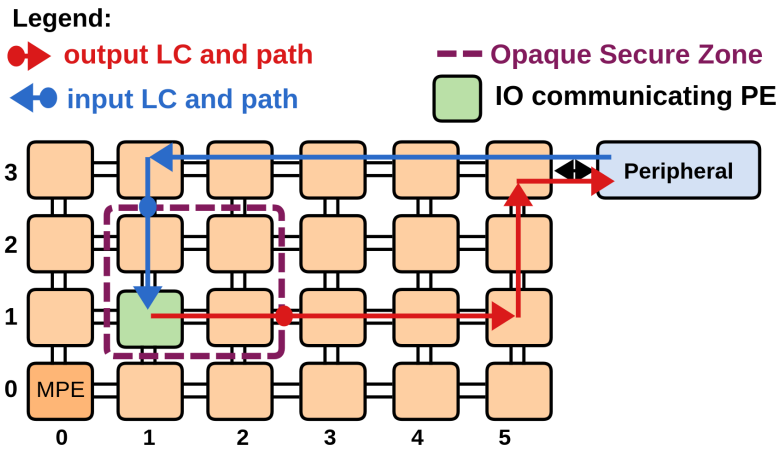



Figure 3.9 – Example of IO communication through an OSZ. Source: [Caimi and Moraes, 2019].

The data NoC differentiates the API with a flag in the header field (D/P flag explained in Figure 3.2). This feature enables to block all data packets arriving at the boundary of the OSZ (in both directions) and to apply selective management of IO packets.

## 3.4 Peripheral Interface

A Network Interface (NI) in a NoC-based system is the bridge between on-chip components, such as processors, memory modules, and dedicated hardware modules, with the NoC [Aghaei et al., 2020]. Its primary function is to manage data exchange by converting incoming and outgoing traffic into a format compatible with the NoC infrastructure. To accomplish this, the NI offers flow control, buffering, routing, and protocol conversion, enabling seamless communication within NoC-based systems and contributing to their overall efficiency and scalability.

In addition, the NI may also enable communication with input and output modules, such as accelerators or shared memories, herein named *IO devices*. These IO devices often use standard protocols for data exchange. The NI integrates these modules into the NoC infrastructure by adapting to their requirements and ensuring proper data encoding and decoding (e.g., an IO device with AXI protocol [ARM, 2013]). This may require protocol converters, specialized buffer management, and tailored QoS policies to maintain a high-performance data exchange with IO devices. Furthermore, NIs also offer configurable interfaces that facilitate the integration of new hardware modules, fostering adaptability and extensibility in NoC-based systems.

Incorporating security as a fundamental requirement in the design of NIs is essential to protect against threats and ensure the integrity of communication between on-chip components and IO devices [Charles and Mishra, 2022].

Therefore, we adopted the “Secure Network Interface with Peripherals” (*SNIP*) [Comarú et al., 2023], which integrates security mechanisms to safeguard communication with internal components in many-core systems.

The SNIP has six main modules, as illustrated in Figure 3.10. Two modules, **Packet Handler** and **Packet Builder**, enable simultaneous communication to and from the NoC. The Packet Handler stores the sensitive data for communication with the application in the **Application Table**, while the Packet Builder retrieves it when necessary. FIFO **buffers** hold data sent to or received from the IO device until consumption. The **Key Generator** produces and updates authentication keys. The next subsections detail these components, except the input and output buffers.

### 3.4.1 Application Table

The main function of the Application Table is to record the applications that are allowed to access the IO device, as well as the information needed to authenticate and answer packets sent by these applications. Each line of the table corresponds to a different commu-

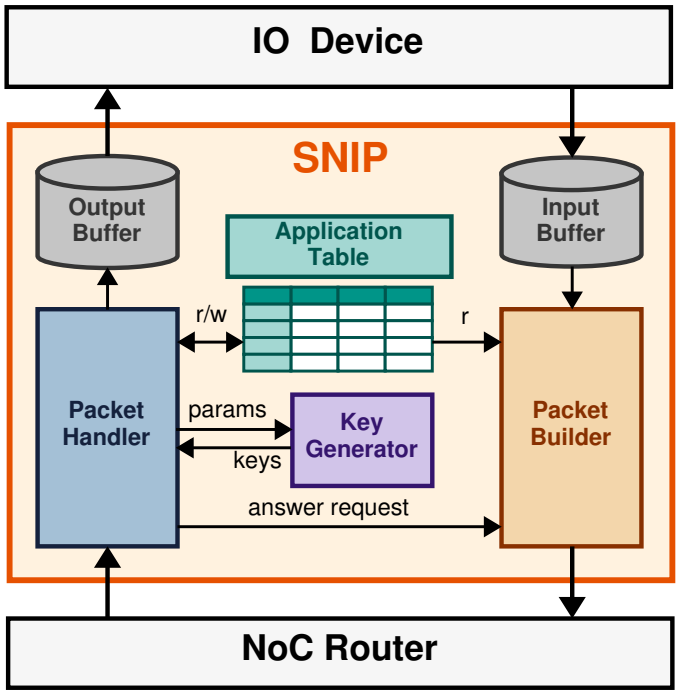



Figure 3.10 – SNIP architecture and interfaces. Source: [Comarú et al., 2023].

nicating application. This application granularity reduces the table size (and, consequently, area) compared to a table with task granularity. Figure 3.11 illustrates the fields available in the table.

| line | valid | appID | k1 | k2 | path_to_SZ | path_size |
|---|---|---|---|---|---|---|
| #1 | | | | | | |
| #2 | | | | | | |

Figure 3.11 – Application Table with two lines, each corresponding to a different application allowed to interact with the peripheral. [Comarú et al., 2023].

- **Valid**: flag used to signal whether the table slot is being used to store an application's information or not.
- **AppID**: ID of the application allowed to access the peripheral.
- **K1 and K2**: keys used by the authentication protocol to assert if a given packet was not tampered.
- **Path_to_SZ**: sequence of turns a packet has to take in the network to reach the OSZ. This field is required to send messages to the application through source routing.
- **Path_Size**: contains the size of the **Path_to_SZ**, which may have up to 6 flits.

The SNIP handles paths from one up to six flits, making the *Path_to_SZ* the largest field on the table. To avoid passing large busses through the table interface, *Path_to_SZ* is broken into six segments, each one corresponding to a flit in the path. Only one segment can be read or written at a time.

The Application Table works as a Content-Addressable Memory (CAM). To access the selected line, we must inform the *appID* of the application we are looking for. The table itself searches the correct line and makes it available for reading or writing. In a CAM, the information used to distinguish each line (here, the *appID*) is called a *tag*.

Furthermore, the table offers two separate interfaces. The primary (read-write) interface is connected to the Packet Handler, while the secondary (read-only) interface is connected to the Packet Builder. This separation enables the SNIP to send and receive messages simultaneously.

### 3.4.2 Packet Handler

The SNIP acts as a slave to the system since it waits for incoming packets to define its action. The Packet Handler is responsible for receiving packets from the NoC and responding appropriately. It executes all the decision-making, acting as a manager to the other components.

Each packet arriving at the SNIP contains two header flits (target and payload size), followed by a set of flits corresponding to the message header and, optionally, the payload flits. The packet handler reads the message header, storing relevant flits (e.g., *service*, *appID*, *path*) into registers to avoid buffering the packet that could cause NoC contention.

There are four different services the packet handler treats:

**IO_INIT:** this service is used by the Manager PE to set the ***k0*** value of the SNIP, which is the key utilized to obfuscate data between the MPE and the SNIP throughout the platform execution.

**IO_CONFIG:** used to register a new application in the SNIP table, thus granting it authorization to access the IO device. The received packet contains the ***appID***, the pair of authentication keys **{*k1,k2*}** and the ***path_to_SZ***. Following the authentication protocol, *appID* is obfuscated using *k0*. By the end of handling, a new line has been allocated in the Application Table, where all values are stored.

The SNIP is accessible to the applications through the IO Communication API, discussed in Section 3.3. The packets themselves were modified to contain the information needed for the authentication process, that is further explained in Section 4.3.

**IO_REQUEST:** is sent by the application to request data from the peripheral. The packet encodes the tuple **{*appID,k1,k2*}**. When the packet is received by the SNIP, the Application Table performs a crypto search. If a matching application is found, the packet is said to be authentic. Only if the incoming message is successfully authenticated, the SNIP answers the application with an `IO_DELIVERY` packet containing the data requested. This outgoing packet is sent through the path configured in **Path_to_SZ** and also contains the authentication flits, for verification on the application side.

**IO_DELIVERY:** carries data the application wants to convey to the peripheral. It also contains the tuple **{*appID,k1,k2*}**. The authentication process is the same as the performed for the `IO_REQUEST` service. If the authentication succeeds, the data contained in the packet is relayed to the peripheral, and the SNIP answers the application with an acknowledge message (`IO_ACK` service).

### 3.4.3 Packet Builder

For the SNIP to communicate with applications, it needs to be able to answer to incoming messages. The Packet Builder module is responsible for assembling those answers and sending them through the data NoC.

Once the Packet Handler decides to send a message to an application, it notifies the Packet Builder through the signal ***answer_request*** (Figure 3.10) and informs the packet parameters: *service* defines the type of message to build, while *appID* specifies which application to send it to.

### 3.4.4 Key Generator

The Key Generator creates the keys used in the Authentication Protocol. This module produces $\{k1, k2\}$ keys, using a Linear-Feedback Shift Register (LFSR) as a pseudo-random key generator. Although an LFSR is not considered the most robust method to generate pseudo-random numbers, it provides a distributed and area-efficient way to generate the authentication keys. For the `IO_CONFIG` service, the LFSR uses *appID* as a seed, and $k1$ is obtained after $n$ rounds in the LFSR and $k2$ after $p$ more rounds.

## 3.5 Hardware Trojan - HT

In Section 2.2.1, we presented an overview of contemporary research involving Hardware Trojans (malicious hardware). This Section describes HT types and effects in the context of this Thesis.

HTs can be classified into two categories according to their activation mechanism: always-on Trojan and trigger Trojan [Shakya et al., 2017]. Always-on HTs are easily detected using, e.g., side-channel analyses. HT triggers include time and physical condition (internal triggers) or user input and component output (external triggers). Trigger HTs are not easily detected since they are usually in sleep mode, being wake-up by some event.

Weber et al. [Weber et al., 2020] detail the insertion of an HT in an open-source NoC. Figure 3.12 shows the attacks explored in [Weber et al., 2020]. The system contains two hardware layers, the trusted Processing Elements (PEs) and the untrusted NoC. Distinct PEs execute tasks A and B ($T_A$ and $T_B$), being an application with security requirements, and $T_Z$ references a malicious task. $T_A$ sends data to $T_B$, represented by the green arrow.

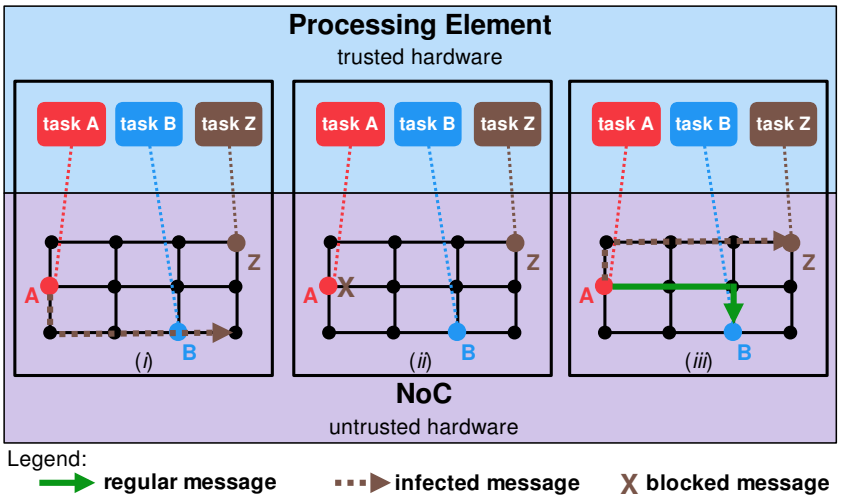



Figure 3.12 – Representation of attacks that an HT may execute. (*i*) misrouting, (*ii*) local port blocking and (*iii*) packet duplication. Source: [Weber et al., 2020].

The reported attacks include:

- **Misrouting** - Figure 3.12(*i*). $T_Z$ configures the HT to misroute the $T_A$ outgoing packets. Consequently, $T_A$ transmits packets to a PE, which is not waiting for data, or to an invalid address. These packets are not consumed, resulting in a DoS attack that obstructs other flows and may block the entire NoC operation.
- **Local-port blocking** - Figure 3.12(*ii*). The application is interrupted, and the result may be catastrophic for critical applications, as in autonomous driving.
- **Packet duplication attack** - Figure 3.12(*iii*). During the attack, packets generated by $T_A$ goes to $T_B$ and also to $T_Z$, leaking sensitive data for a malicious task.

# 4. SEMAP - SECURE MAPPING WITH ACCESS POINT

This Chapter presents the *second original contribution* of this Thesis, the **Secure Mapping with Access Point (SeMAP)**, a proposal enabling the mapping of multiple OSZs simultaneously, protecting secure applications ($App_{sec}$) against unauthorized accesses, and ensuring the availability of paths to the IO devices. The main publication related to SeMAP is:

> SeMAP - A Method to Secure the Communication in NoC-based Many Cores
> Faccenda, Rafael Follmann; Comarú, Gustavo; Caimi, Luciano Lores; Moraes, Fernando Gehm.
> IEEE Design & Test, vol. 40(5), pp 42-51, October 2023.

The goal of this Chapter is to fulfill objectives *SG1* to *SG4*. These objectives seek to establish secure communication between IO devices and $App_{sec}$ running within an OSZ. Given that the peripherals are located outside the OSZ, it becomes necessary to create openings in the OSZ to enable the flow of incoming and outgoing messages with these IO devices. These openings are named Access Points, as defined in Definition 2.

**Definition 2.** *Access Point (AP)* is a controlled OSZ opening, enabling an $App_{sec}$ to communicate with IO devices.

Figure 4.1 illustrates an MCSoC with an $App_{sec}$ that communicates with a peripheral. This Figure also presents concepts used in this Thesis: peripherals (IO device and Secure Network Interface for Peripherals - SNIP), PE, path, AP and OSZ. Note that an $App_{sec}$ communicate with IO devices. The term “peripheral” refers to an IO device and an SNIP responsible for managing the security aspects of the IO communication.

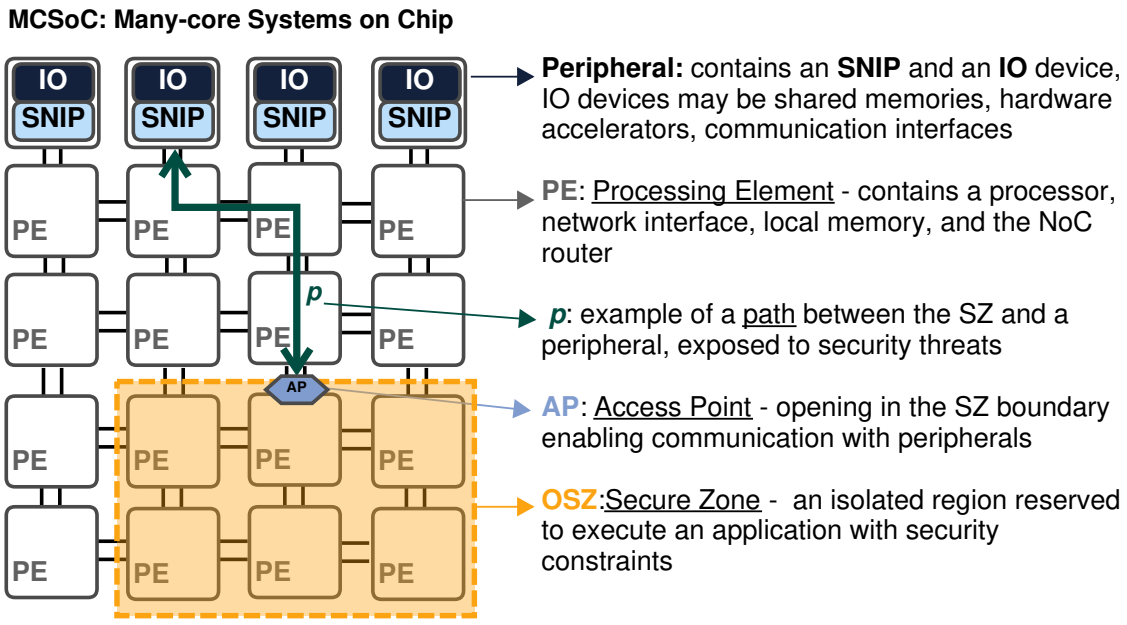



Figure 4.1 – MCSoC with an OSZ and IO communication terminology.

At the same time that the AP enables the communication of an $App_{sec}$ with IO devices, it introduces vulnerabilities that attackers can exploit. Assuming as trusted the control NoC, MPE, SNIP and OS, the threats are as follows:

- Forged packets entering the OSZ through the AP can cause attacks such as DoS (denial-of-service) [Charles et al., 2020b], spoofing [Rout et al., 2020], and data corruption. The effects of these attacks include performance degradation up to the complete application hang.
- Malicious packets to peripherals may execute DoS, spoofing, eavesdropping, and data corruption. Besides modifying the application data stored, with unpredictable effects, the intruder may steal sensitive information stored in the IO device.
- The exposed path is prone to Hardware Trojans (HTs) and side-channel attacks (SCAs). The HT may access the packet content, corrupt the original data, execute an eavesdropping attack, misroute, or block packets [Daoud and Rafla, 2019b, Manju et al., 2020, Ahmed et al., 2021].
- The IO device itself may be malicious and execute DoS attacks or transmit the application data to an intruder.

Therefore, to prevent DoS attacks, spoofing, and eavesdropping, not allowing malicious packets to enter into the OSZs or peripherals, nor malicious IO devices performing unauthorized data injection, we follow the security principles defined in [Caimi and Moraes, 2019]:

1. Differentiate the PE↔PE communication from the PE↔Peripheral: this differentiation prevents malicious applications from trying to inject packets into OSZs.
2. Master-slave communication: the PEs inside the OSZ must initiate all IO transactions. Thus, the PEs of the OSZ discard all unexpected packets.
3. IO Packets must be signed to ensure authenticity and that they come from/to the correct peripheral.
4. Avoid unreachable resources, i.e., an OSZ may not block access to peripherals.

The remainder of this Chapter is organized as follows: First, an analysis of the previous method for IO communication is presented, along with an introduction to SeMAP in Section 4.1. This is followed by a detailed description of SeMAP resource management implementation in Section 4.2. Next, Section 4.3 discusses the protection of message exchanges. The Chapter concludes with a presentation of results in Section 4.4 and final remarks in Section 4.5.

## 4.1 MCSoC Partitioning for Security

The proposal presented in [Caimi and Moraes, 2019], discussed in Section 3.3, named *Dynamic SZ* (DSZ) focuses on mapping flexibility. OSZs can be mapped at any region of the MCSoC, respecting the restriction of not blocking paths to peripherals by always keeping a distance of one PE between OSZs. Figure 4.2 illustrates a system that follows the DSZ method with three mapped secure zones. Note the absence of the SNIP module in this proposal.

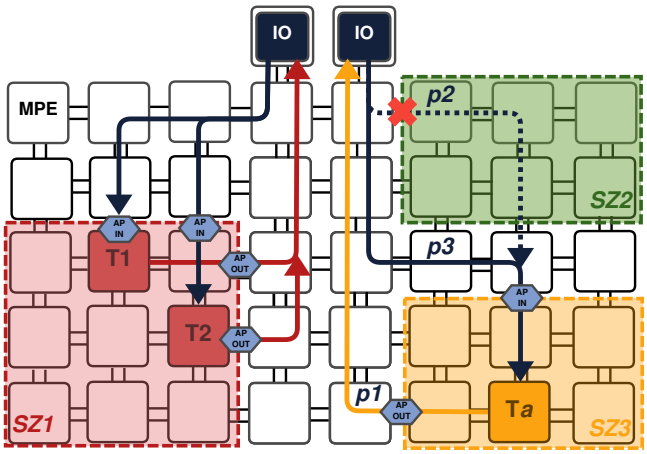



Figure 4.2 – Example of Dynamic SZ method, adapted from [Caimi and Moraes, 2019].

In the DSZ method, the APs are opened for each communication transaction, and their position is defined based on where the packet will pass following the XY routing. The task that starts an IO communication opens two unidirectional APs (AP IN and AP OUT in Figure 4.2), transmitting a control packet to each OSZ border to be opened. The border router, then, receives this control packet and sets the AP to allow the crossing of one data packet. Once the data packet passes through the AP, it is immediately closed. This method ensures that only one packet traverses the AP per transaction, minimizing attack attempts. On the other hand, if multiple tasks communicate with peripherals, several APs can be opened simultaneously (*SZ*1 in the Figure), increasing the attack surface. Packets to traverse the APs must meet two conditions: (*i*) be IO packets; (*ii*) match the key shared by the IO device and the $App_{sec}$. The packet is discarded otherwise.

The mapping flexibility brings the *masking effect*. Consider Figure 4.2 with only *SZ*1 and *SZ*3 mapped and running in the system, communicating with IO devices as shown by the arrows *p*1 and *p*2. When *SZ*2 enters in the system, it blocks path *p*2, which connects the IO device to *SZ*3. Thus it is necessary to compute a new path using source routing. The PE closest to the IO device computes the new path (*p*3), transmitting it to the IO device for subsequent data transmissions. This approach adds a security threat, as it involves a PE not related to $App_{sec}$, allowing it to know the location of the AP and use this information to initiate an attack.

Despite the DSZ application mapping flexibility, there is a restriction related to the task mapping inside the OSZ, named *alignment effect*. The DSZ does not allow two or more tasks on the same X or Y coordinate to communicate with peripherals because the DSZ authorizes only one transaction per AP. If two tasks are aligned, both activate the same AP, but only one packet passes through it, thus blocking one of the tasks.

As an alternative to solve the aforementioned issues, SeMAP restricts the $App_{sec}$ mapping and allows only one bidirectional AP per OSZ. The goal is to have a single aperture for all the IO transactions to reduce the attack surface. Besides that, the System Manager reserves rows and columns to run only applications without security constraints, named *Gray Areas* (GA) – Definition 3.

**Definition 3.** *Gray area (GA)* is an MCSoC region where applications without security requirements are mapped. The set of PEs in the GA must provide paths to all peripherals connected to the system.

The routers in the GA never receive an OSZ, which means that communication flows will not be affected by the arrival of new applications, therefore solving the *masking effect* of the DSZ and ensuring reachability. The PEs outside the gray area are part of the *Restricted Area*, where the OSZs are mapped.

Figure 4.3 illustrates an example of MCSoC with GA. The gray-colored tiles represent the gray area PEs. In addition, three $App_{sec}$ mapped in the restricted area with at least one side juxtaposed to the GA, guaranteeing a path to the IO devices. The peripherals are attached to the north side of the system for the sake of simplicity. SeMAP does not restrict peripherals to a given system side but requires the peripherals to be connected to a router belonging to a GA.

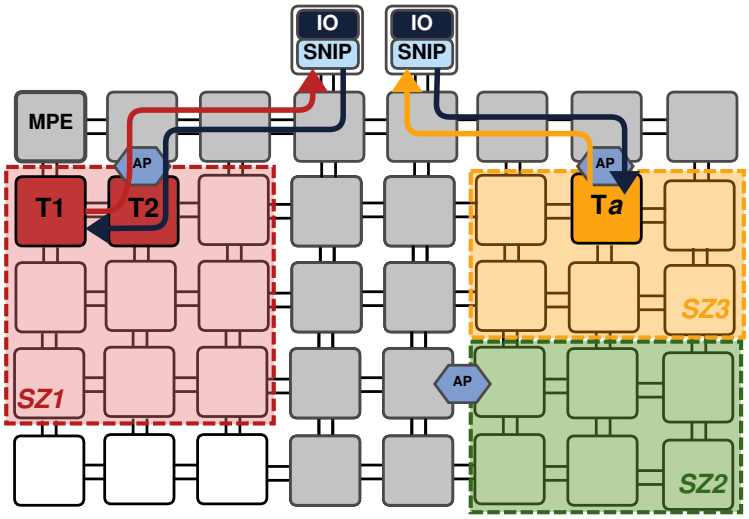



Figure 4.3 – Gray and restricted areas. Three $App_{sec}$s mapped on the restricted area, each with an Access Point (AP). The path AP↔Peripheral is defined by source routing.

Next sections detail the SeMAP method, starting from the resource management that now needs to consider the GA to map OSZs, applications, and tasks (Section 4.2), followed by the mechanisms to protect the message exchange between IO and $App_{sec}$, which include the SNIP and the authentication protocol (Section 4.3).

## 4.2 Resource Allocation with Gray Area

Due to the proposal of gray and restricted areas, it is necessary to create new algorithms that define the shape and location of OSZs. The task mapping must also consider the AP location to reduce the hop count to the IO devices the $App_{sec}$s communicate with.

The GA must be continuous, with at least one row and one column of the MCSoC, including the border side(s) of the MCSoC where peripherals are connected. Figure 4.4 exemplifies 3 different GA configurations. Following these constraints, any packet that arrives at any GA router can find a path to the peripherals.

The GA is defined at the system startup according to the expected workload. Once the MPE selects the rows and columns of the GA, it is impossible to change the regions reserved for secure and non-secure applications at runtime.

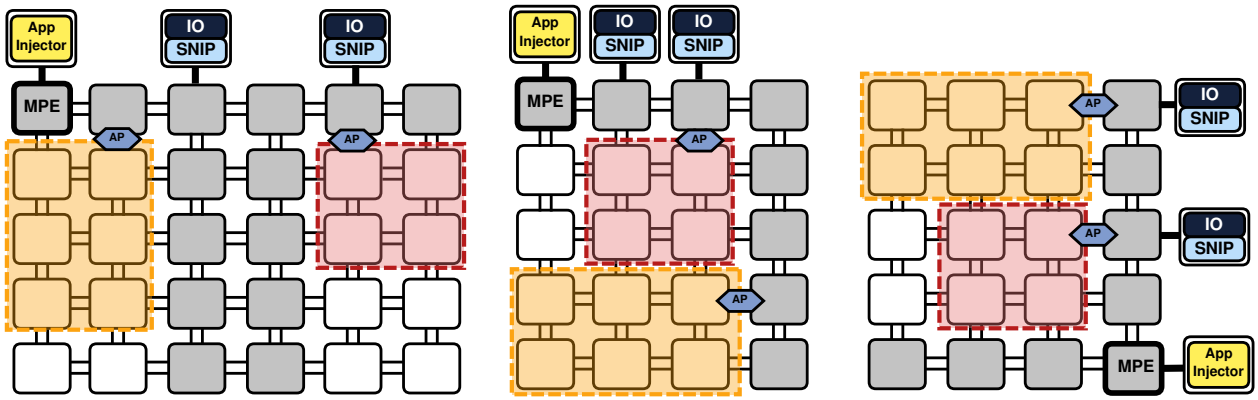



Figure 4.4 – Examples of gray and restricted areas.

### 4.2.1 OSZ Shape, Location and Mapping

The definition of the OSZ shape now prioritizes shapes having the width of the restricted area. This method improves system utilization, avoiding PEs without access to the gray areas. Figure 4.5(a) shows an example of two applications mapped into a restricted area: one with three tasks (orange) and the other with four tasks (red). Figure 4.5(b) shows possible shapes for a 3-task application: 3x1 and 2x2 with one PE in excess. Since the

restricted area is 3x3, the first shape is 3x1 for having the same width as the restricted area. The 2x2 is still valid and can be mapped if the 3x1 shape does not fit in the system. Figure 4.5(c) presents the possible shapes for a 4-task application: 2x2, 4x1, and 1x4. In this case, the only option is the 2x2 since the latter two options do not fit in the 3x3 restricted area. One can see that the second application isolated the bottom right PEs from the GA, which is why the preferred shapes are the ones with the same width as the restricted area.

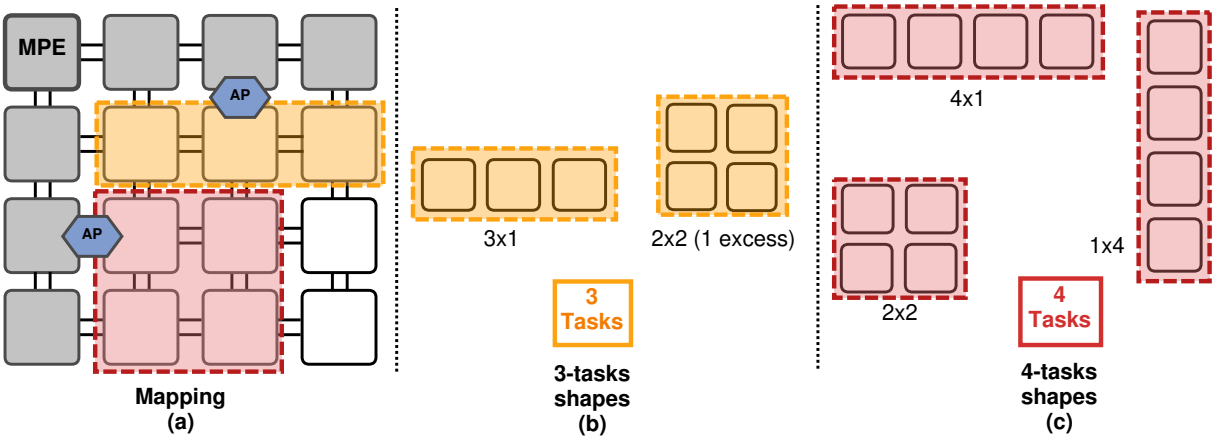



Figure 4.5 – Illustration of OSZ shapes in a restricted area.

The process for selecting the location of the OSZ employs a Sliding Window Search (SWS) algorithm, as detailed in Chapter 3. This algorithm begins its search from the row or column closest to the peripherals and furthest from the intersection of the GA, defined as the point where a GA row and column intersect. The SWS algorithm aims to identify a suitable region for placing the OSZ, focusing on areas with available PEs adjacent to the GA. As depicted in Figure 4.6(a), the SWS initiates its search near the top row, which is the peripherals' side. The search progresses horizontally (indicated by the horizontal orange arrow), moving from left to right. Upon encountering the GA column, the SWS alters its course and proceeds vertically downwards (as shown by the vertical orange arrow). The algorithm opts for an alternative shape if the initial shape selected for the OSZ does not fit within the available space. In cases where it is not feasible to map the $App_{sec}$ due to space constraints, the application will be scheduled for mapping at a later time, specifically after the completion of another application.

After defining the OSZ shape and coordinates, the next step is task mapping. The system manager knows the tasks that communicate with peripherals. The system manager uses this information to map first these tasks near the AP and then the remaining tasks. In Figure 4.6(b), blue arrows represent the direction in which the manager maps the tasks with IO communication and yellow arrows represent the mapping of the remaining tasks. It is important to note that the rectangle reserved to map the application ensures minimal hop count between communicating tasks, as its size in terms of PEs is equal or close to the number of the $App_{sec}$ tasks.

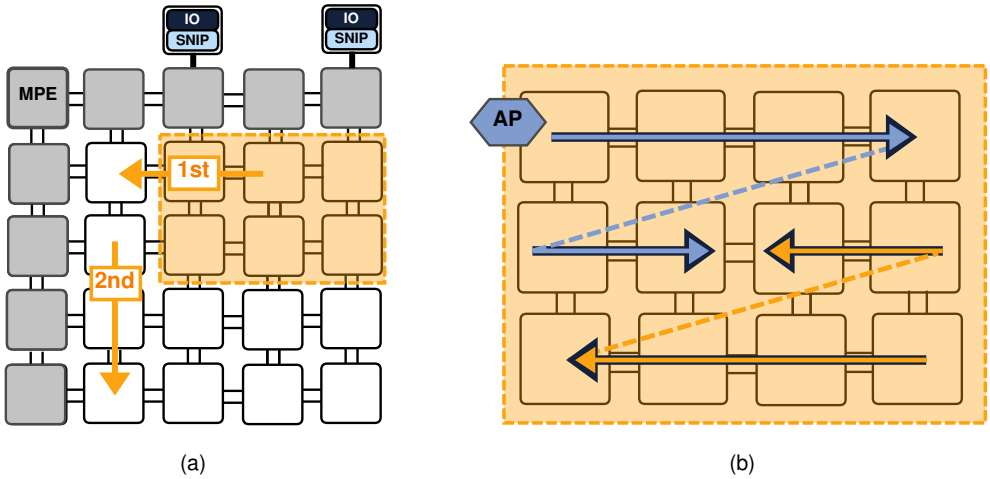



Figure 4.6 – (a) Sliding Search Window (SWS); (b) directions of task mapping inside the OSZ.

### 4.2.2 Access Point (AP) Definition

The AP can be mapped at any router port in the OSZ border adjacent to the GA. The AP is mapped in the OSZ top-left or top-right router by default, according to the GA position. If the OSZ is at the top of the restricted area, the AP is mapped at the north port of the top-middle router. Figure 4.5 presents both cases: the AP of orange OSZ is in the middle-top position, while the purple OSZ is a default case, with the AP at the top-left position.

Note that this is the first default positioning of the AP, meaning that there is the possibility to change the AP location periodically or whenever suspicious behavior is detected.

### 4.2.3 IO Path configuration

PEs inside the OSZ does not use the default XY routing algorithm to reach the peripheral because the SeMAP forces them to follow a route that passes through the AP and GA. Thus, the packet follows the source routing algorithm, with each hop to take already defined by the source.

The first communication of a given task with a peripheral fires a path configuration heuristic. Since the OS knows the AP, the gray area routers, and peripheral locations, it computes the path to the peripheral, passing through the AP. The MPE is in charge of calculating and configuring the returning path (peripheral→AP) to the application, explained further in Section 4.3.

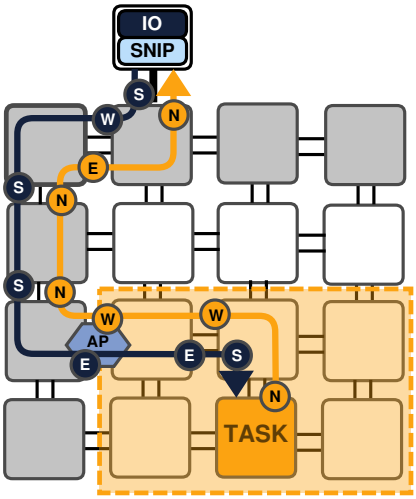



Figure 4.7 – Example of SR path from/to task to/from peripheral through AP.

First, the OS computes the path PE→AP, then the path AP→peripheral, according to the gray area shape. The algorithm builds the route by going first to North until reaching the AP horizontal coordinate. Next, it goes to East or West according to the AP location. After reaching the AP, the route follows the gray area, in the same manner, going North first, then East or West towards the peripheral.

Figure 4.7 illustrates an example of a path computation between “Task” and peripheral. There are two paths: path A, the yellow arrow from Task→IO, and B, the black arrow from IO→Task. The circles show each of the ports taken for each of the paths. Therefore, path A is [N, W, W, N, N, E, N], then, path B is the opposite in reverse order [S, W, S, S, E, E, S].

### 4.2.4 DSZ and SeMAP comparison

This Section presents the performance of applications considering the two methods for communicating with peripherals: DSZ and SeMAP. Figure 4.8 presents the applications mapping with their respective paths in two scenarios to evaluate the methods of communication with peripherals. In the first scenario, (a) and (b), the system receives the DTW (Dynamic Time Warping) application at startup. At *5 ms*, a new application enters the system (MPEG decoder). Note that the DTW DSZ (Figure 4.8(a)) has two paths broken (red arrows) by the MPEG, firing two path search computations (due to the “masking effect”). At *9 ms*, a PC (Producer-Consumer) is mapped, also blocking two DTW paths. A second scenario, (c) and (d), is evaluated, swapping the DTW with MPEG. In the first scenario (DTW in the left corner), the IO communication volume (number of messages exchanged with the peripherals) is higher.

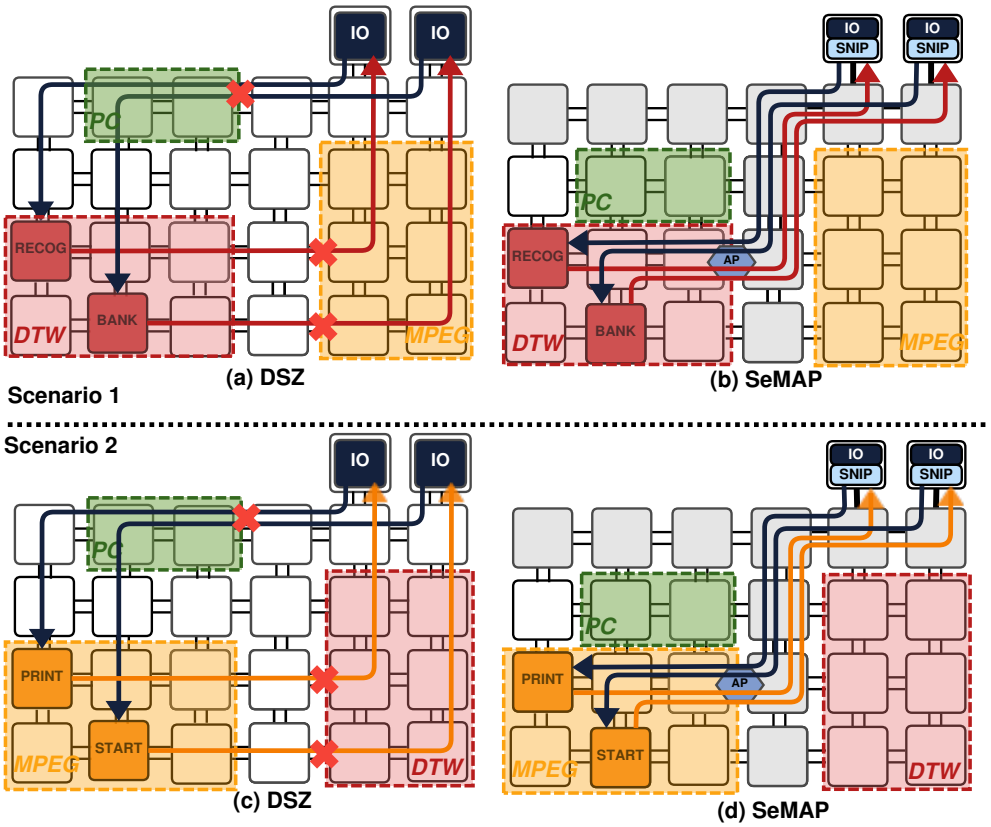
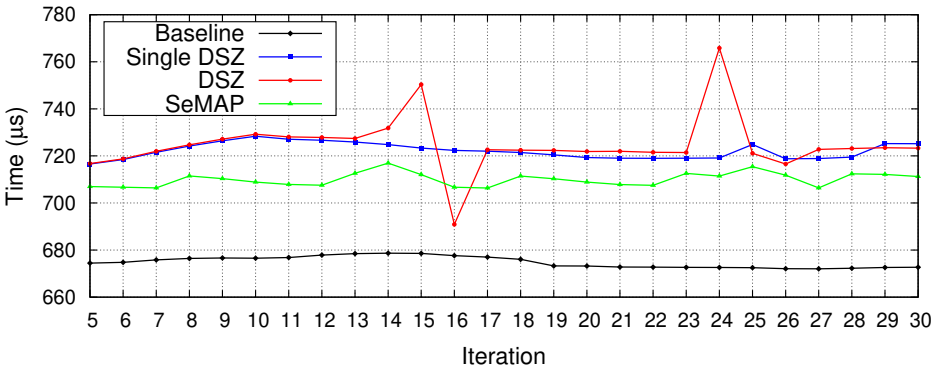



Figure 4.8 – Application mapping to evaluate the methods to communicate with peripherals.

Figure 4.9 presents the iteration latency for DTW and MPEG applications. The y-axis is the time required to execute each application iteration (in $\mu s$), and the x-axis is the iteration number. Graphs omit the first five iterations, considering these as the warm-up period. Each graph has 4 curves:

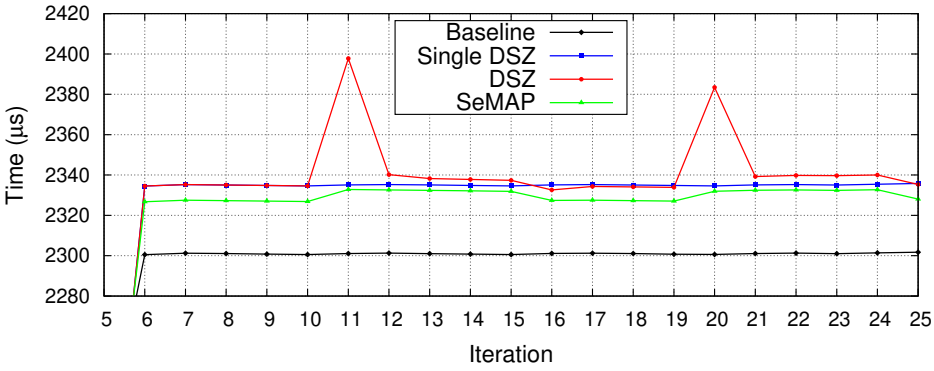
- **Baseline** (black line): execution of the applications without communication with peripherals. Input data is assumed to be stored in the local memories, and results are also stored in the local memories. Simulating the baseline MCSoC aims to evaluate the overhead due to the communication with peripherals.
- **Single DSZ** (blue line): only the reference application (DTW or MPEG) executes in the system. The goal of simulating the DSZ approach without other applications is to evaluate the DSZ method in the absence of the masking effect.
- **DSZ** approach (red line) using evaluation scenario presented on fig. 4.8(a).
- **SeMAP** approach (green line) using evaluation scenario presented on fig. 4.8(b).

Comparing **SeMAP** and **Single DSZ** versus **Baseline**, the latency per iteration increases 1.2% (MPEG) to 7.3% (DTW) when there is IO communication (average values). The latency increases due to the: (*i*) non-minimum paths; (*ii*) master-slave communication




Execution time:
18.26 ms (baseline)
18.44 ms (single DSZ)
18.51 ms (DSZ)
18.37 ms (*SeMAP*)

(a) DTW iteration latency.




Execution time:
13.82 ms (baseline)
13.91 ms (single DSZ)
13.94 ms (DSZ)
13.98 ms (*SeMAP*)

(b) MPEG iteration latency

Figure 4.9 – Iteration latency using the baseline MCSoC, DSZ and *SeMAP*.

protocol, i.e., all transactions started by the $App_{sec}$; (*iii*) management of APs in the DSZ method (opening and closing of APs at each transaction).

**SeMAP** reduces the latency per iteration (0.33% for MPEG and 2.7% for DTW, best cases) compared to **Single DSZ** because it does not need to manage APs. On the other hand, there is a latency per AP to start the application execution, as it is necessary to compute the paths for each AP (25$\mu$s@100MHz - average value per path).

The masking effect, observed in **DSZ** (red curves), increases the iteration latency when a new application enters the system, blocking the PE→Peripheral communication, requiring to reroute the broken paths. In both scenarios, the masking effect only affects few iterations, since this mechanism is activated once, being the alternative path taken for the subsequent communications. The valley observed in the first scenario is due to the application pipeline behavior, i.e., while a task is blocked waiting for a new path, the other tasks continue to run. The performance degradation increases in scenarios with a larger number of broken paths,

**SeMAP** is immune to the masking effect, presenting a small latency increase (0.2% to 1.8%) when a new application enters the system. The network traffic increases when a new application is admitted due to the transmission of the object code of the tasks. This increase in network traffic explains the slight increase observed in latency.

In terms of overall application execution time, SeMAP has a minimal overhead - 1.3% for DTW (**Baseline** versus **DSZ**) and 0.5% for MPEG (**Baseline** versus **SeMAP**).

In addition to the advantage related to not recalculating paths due to the masking effect, SeMAP presents advantages that justify its adoption:

1. single bidirectional AP per $App_{sec}$, reducing the attack surface;
2. simplified OSZ internal mapping due to the absence of the alignment effect;
3. AP management by the kernel is simplified, as there is no need to control the opening and closing of APs for each IO transmission;
4. management of APs by the kernel is simplified, as there is no need to control the opening and closing of APs;
5. APs are configured through memory-mapped registers and not by control packets;
6. there is no need to use a PE near to the peripheral to compute the path to the OSZ, also reducing the attack surface.

Two potential weaknesses of SeMAP were identified. First, a communication bottleneck could occur at the AP, given that all IO communications go through the same router link. We observed that this issue did not appear in the experiments, given that communication with IO devices has a reduced rate, corresponding to the search for data to be processed and subsequently sending the processing to the peripherals. Second, fragmentation of the restricted area for $App_{sec}$. It is possible to defragment the system by using task migration. Such defragmention is out of the scope of the Thesis, being a possible future work.

## 4.3 Securing the Message Exchange

The previous section detailed the system configuration defined at system startup and the runtime mapping of secure applications ($App_{sec}$) into OSZs. Once an $App_{sec}$ is mapped into the system, the MPE releases its execution. This section delves into the secure mechanisms to enable communication with IO devices. The Authentication protocol is the key mechanism to enable this communication, elaborated in Section 4.3.1, which incorporates authentication flits into the packets. While confidentiality could be achieved by encrypting the payload, this aspect is out of the scope of this proposal, as it can be addressed using established lightweight cryptography algorithms. Despite the obfuscation of authentication keys via XOR operations, these keys are periodically renewed, as detailed in

Section 4.3.2. Section 4.3.3 discusses the hardware implementation of the APs, designed to support the Authentication protocol. This section concludes with Section 4.3.4, which describes the IO API and the services used in communication with IO devices.

### 4.3.1 Authentication Protocol

The primary goal of the Authentication protocol is to generate key values — ***k0***, ***k1***, and ***k2*** — which are used in packets to enable the OS, AP and SNIP to verify the trustworthiness of the source and the legitimacy of the data. These keys are generated using a Linear-feedback shift register (LFSR) and are obfuscated with XOR operations, allowing them to traverse the GA, the non-secure area of the MCSoC. The protocol encompasses four phases: *Initialization*, *Application Deploy*, *Communication*, and *Key Renewal*.

#### Initialization

The *initialization* phase occurs at system startup. The MPE generates unique keys, named ***k0***, for each PE and SNIP in the system and sends them to their respective PEs and SNIPs. In this same process, the LFSR polynomial is configured. Since this action occurs when there is no other application or traffic in the system, these values can be transmitted without encryption, exempting the use of complex key distribution mechanisms such as Diffie-Hellmann [Diffie and Hellman, 1976], which would result in software and hardware overheads. Applications and IO devices do not have access to ***k0***, guaranteeing the confidentiality and integrity of these keys.

#### Application Deploy

The deploy starts when the MPE receives a `New_App` message from the $App_{inj}$. Then, the MPE executes the mapping heuristic, and manages the OSZs - steps previously detailed in Section 4.2.1. Figure 4.10 presents the sequence diagram of the Application Deploy phase, fired when the $App_{inj}$ requests the execution of an $App_{sec}$.

The MPE maps the $App_{sec}$ and sends the mapping result to the $App_{inj}$, which transmits the application object code (protected by a Message Authentication Code mechanism [Caimi and Moraes, 2019]) to the selected PEs in the restricted area. In parallel, the MPE randomly generates the tuple $\{appID, n, p\}$, where *appID* is a unique application identifier and $\{n, p\}$ integer values. The MPE transmits `Task_Release` messages to all $App_{sec}$ PEs with two initialization flits, $i1$ and $i2$ (Equation 4.1), with the tuple obfuscated by ***k0***.

$$\mathbf{i1} = appID \oplus k0_{PE_x} \qquad \mathbf{i2} = (n \,\&\, p) \oplus k0_{PE_x} \tag{4.1}$$

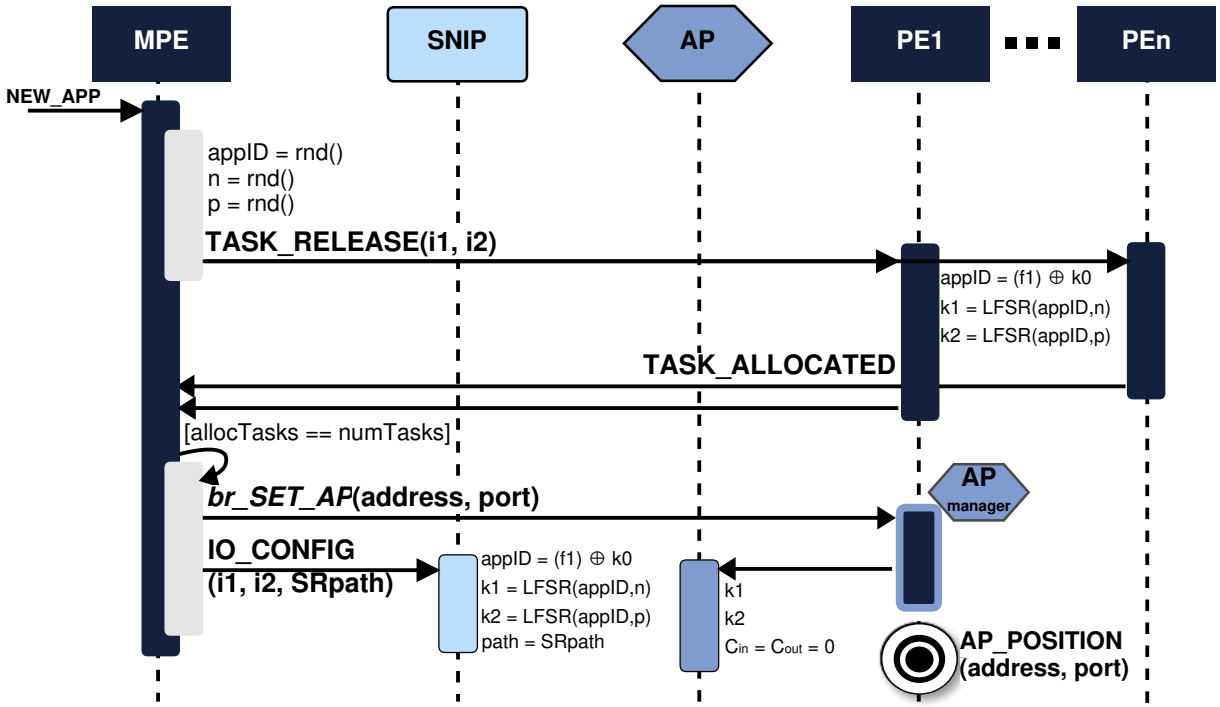



Figure 4.10 – Sequence diagram of the Application Deploy phase.

PEs restore $\{appID, n, p\}$ upon receiving the initialization flits: $appID$ = **i1** $\oplus$ $k0_{PE_x}$; $n = MSB(\textbf{i2} \oplus k0_{PE_x})$; $p = LSB(\textbf{i2} \oplus k0_{PE_x})$. The $appID$ is the seed for the LFSR. The $\{n, p\}$ values correspond to the number of shifts in the LFSR to generate the authentication keys $\{k1, k2\}$. The reason to use an LFSR for key generation comes from its simple hardware implementation and linearity. At the end of this step, all PEs have the same $\{k1, k2\}$ keys, without transmitting them through the NoC. The PEs notify the MPE through a `Task_Allocated` message the correct object code reception and keys generation.

The MPE executes four actions after receiving all `Task_Allocated` messages:

1. elects a PE as “AP manager”, transmitting `br_Set_AP` via control NoC to it, with the port to place the AP;
2. configures the SNIPs with $\{appID, n, p\}$ and the path from the SNIP to the AP (`IO_config`), also using $i1$ and $i2$ (Equation 4.1) with the SNIPs $k0$;
3. sends through the control NoC a message to block the OSZ links (not included in the figure);
4. sends through the control NoC a message to start $App_{sec}$ (not included in the figure).

The “AP manager” configures the AP through memory-mapped registers (MMR), sets $\{k1, k2\}$, and resets the transaction counters $\{C_{in}, C_{out}\}$. The “AP manager” also broadcasts the AP address for all PEs executing $App_{sec}$ (`AP_position`).

At the end of the *Application Deploy* phase, all PEs and peripherals of $App_{sec}$ have the authentication keys $\{k1, k2\}$, so they can start the packet exchange.

Communication

This phase corresponds to authenticated communication between $App_{sec}$ and peripheral, through the SNIP (Section 3.4). Due to the master-slave communication method, tasks are responsible for starting the communication. Tasks may execute two services: `IO_delivery`, to send data to a peripheral; `IO_request`, to read data from a peripheral. Both services must include the tuple $\{appID, k1, k2\}$ encoded in two flits (Equation 4.2).

$$\mathbf{f1} = k1_{PE} \oplus k2_{PE} \qquad \mathbf{f2} = appID \oplus k2_{PE} \tag{4.2}$$

The SNIP authenticates the received packet by retrieving the *appID* with the $k1$ value stored at the SNIP (Equation 4.3).

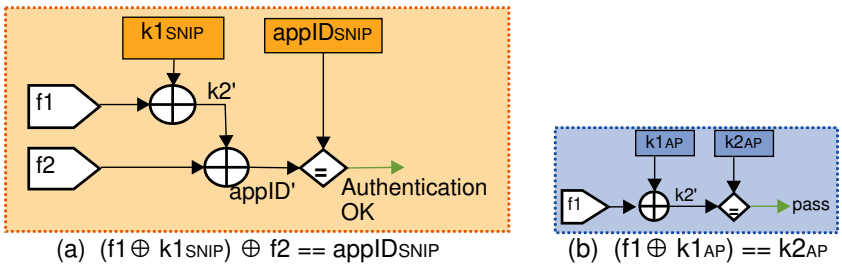
$$(\mathbf{f1} \oplus k1_{SNI}) \oplus \mathbf{f2} == appID_{SNI} \tag{4.3}$$

Then, the SNIP searches for a line in its internal table that matches the received $appID_{SNI}$, which could result in:

1. *No match*: the SNIP packet handler discards the packet, avoiding DoS and spoofing attacks;
2. *Valid AppID and the SNIP are not executing any transaction*: the packet handler reserves the SNIP for the transaction with the IO device (reading or writing).

The answer packet, SNIP$\rightarrow App_{sec}$, has to pass through two authentication locations, at the AP and the target PE. The AP extracts $k2$ from the first flit. If it is equal to $k2_{AP}$ ($k2$ value stored in the AP), the flit enters the OSZ. Otherwise, the AP discards the packet. This lightweight XOR verification avoids attacks such as spoofing and DoS. After reaching the target PE, the operating system (OS) extracts the AppID ($\mathbf{f2}_{SNIP} \oplus k2_{PE}$) from the packet and verifies if the retrieved AppID matches the stored AppID.

Figure 4.11 presents the two authentication circuits: (a) authentication in the SNIP – Equations 4.2 and 4.3, and (b) the verification of k2 made in the AP.




(a) (f1 ⊕ k1SNIP) ⊕ f2 == appIDSNIP (b) (f1 ⊕ k1AP) == k2AP

Figure 4.11 – Lightweight authentication modules.

Persistently using flits {f1, f2} with unchanged values over an extended period presents an opportunity for malicious activities, such as an eavesdropping attack perpetrated by a Hardware Trojan (HT). However, it is important to note that merely acquiring {f1, f2} is not sufficient for launching an attack. Due to the use of source routing, an attacker would not have access to the precise addresses of the SNIP and AP. Moreover, the AP enhances security by monitoring the number of packets received through transaction counters. To further increase the security of the system, the authentication mechanism periodically renews {k1, k2}, as a precautionary measure, even in the absence of detected threats.

### 4.3.2 Key Renewal

Transaction counters $\{C_{in}, C_{out}\}$ complement the authentication protocol. These counters are located in the AP, $C_{in}$ counts the number of packets entering the OSZ, while $C_{out}$ counts the packets exiting the OSZ. To ensure secure communication, the condition $C_{in} < C_{out}$ must always be satisfied at the AP, in accordance with the master-slave communication protocol. The process for renewing keys is triggered by two events: (*i*) $C_{out}$ reaching a predefined threshold (set to 64 in the current implementation); (*ii*) detection of a malicious packet, characterized by a packet that contains the correct key but lacks the IO flag (a flag indicating that the packet was generated by the IO API) or an unexpected value in $C_{in}$. Upon detecting a malicious packet, the AP notifies the MPE.

Figure 4.12 illustrates the key renewal process initiated when the predefined threshold value is reached. This process begins with the AP interrupting the “AP Manager” to generate new keys. To ensure synchronization, the “AP Manager” first notifies all PEs within the OSZ to complete any ongoing IO transactions and temporarily halt subsequent IO communications. Once all PEs in the OSZ acknowledge the “AP Manager”, it generates two new random numbers, $\{n, p\}$. These numbers are then transmitted to all PEs in the OSZ (KEY_EVOLVE message through the control-NoC) for the generation of new keys $\{k1, k2\}$ using the LFSR with the previous $k2$ key serving as the seed. Concurrently, the “AP Manager” also transmits these new $\{k1, k2\}$ keys to the AP and resets the transaction counters

$\{C_{in}, C_{out}\}$. This initial phase of the key renewal process occurs entirely within the OSZ, thereby mitigating any security risks associated with the keys.

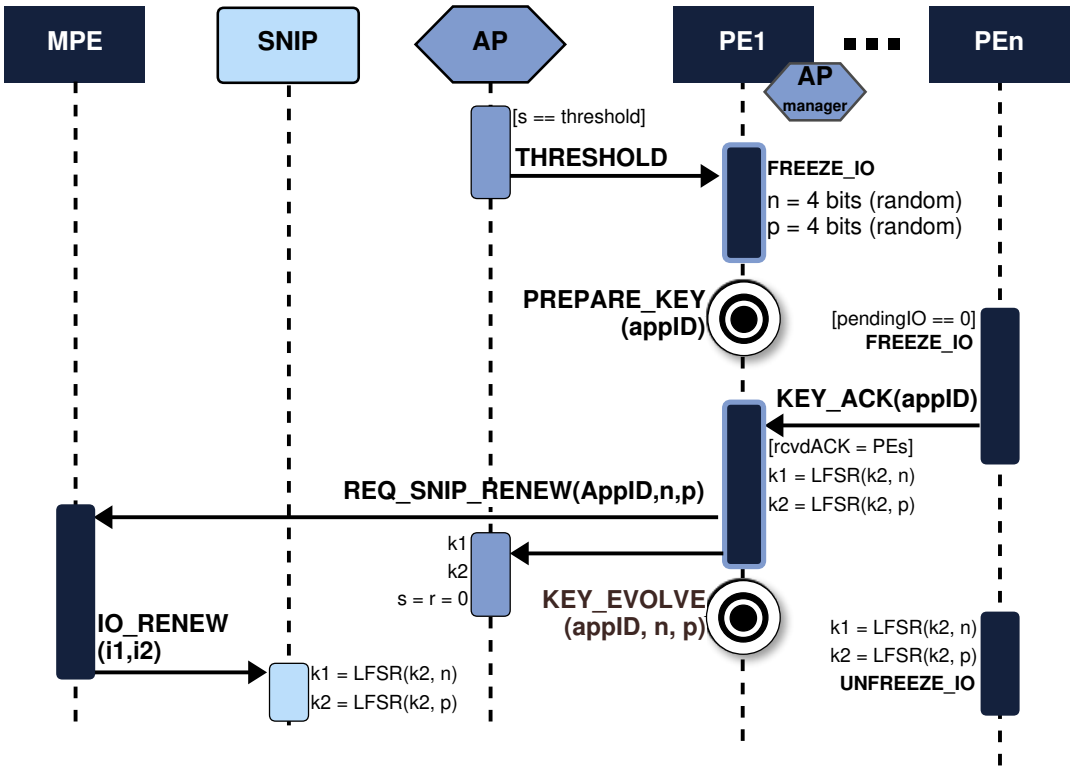



Figure 4.12 – Key Renewal sequence diagram

The final step in the Authentication protocol involves the “AP Manager” sending a REQUEST_SNIP_RENEW message through the control NoC, with $\{AppID, n, p\}$, to the MPE. It is necessary to make this request through the MPE because the SNIP does not have an interface with the control NoC. The MPE then send an IO_RENEW packet through the data NoC, using k0 key to obfuscate the contents of $\{AppID, n, p\}$, following the same procedure as in the Application Deploy phase (as outlined in Equation 4.1). Upon receiving this packet, the SNIP searches for the corresponding $\{AppID\}$, retrieves the stored k2 key from its table, and then generates new keys using its LFSR. For this generation, the LFSR uses the retrieved k2 as the seed and the $\{n, p\}$ values from the packet as the number of shifts in the LFSR.

### 4.3.3 Access Point Architecture

As explained in the previous section, the AP is the primary security barrier for packets attempting to enter the OSZ. This section details the implementation of the security mechanisms within the AP. The AP is designed as a hardware module, integrated into all

router ports on channel 0[1], except the local port, to minimize the area overhead. The operation of the AP is controlled by memory-mapped registers (MMRs). Figure 4.13(a) illustrates the "secure router" configuration, with 4 APs and 9 Link Control (LC) modules. Each LC is responsible for enabling or disabling the links. The system uses five registers to control these modules: *k1reg*, *k2reg*, *THR*, *apReg* (defines the AP to activate), and *LCreg* (defines the LC(s) to activate).

When the MPE defines an OSZ, the PEs in the OSZ border activate all LCs of the OSZ boundary, except the one connected to the AP, discarding incoming and outcoming traffic. To discard a packet, regardless of its direction, `AND` gates mask *tx/rx* signals, and `OR` gates activate the credit control flow signal.

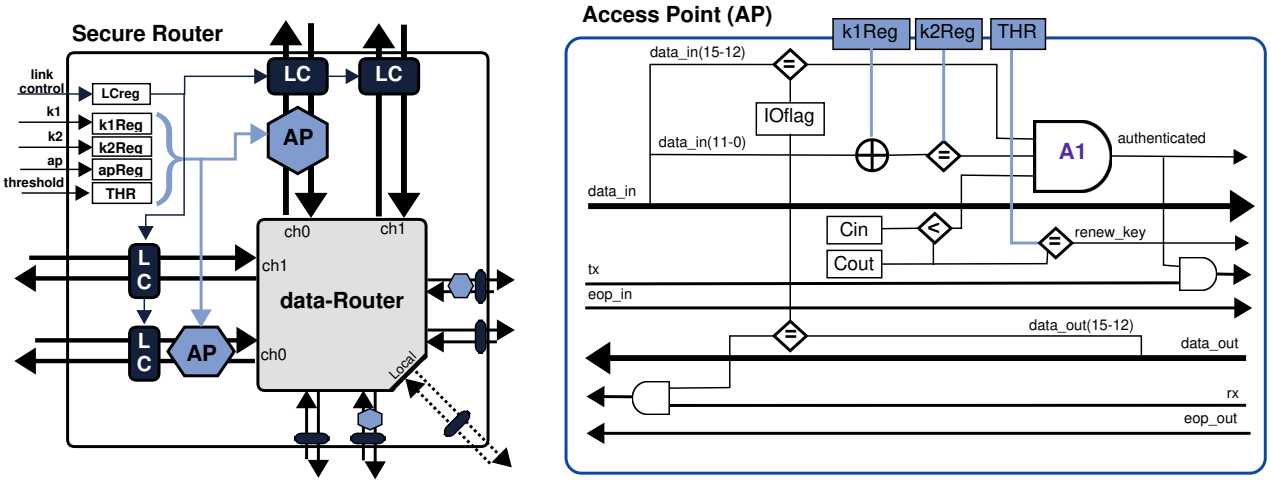



Figure 4.13 – Secure Router and Access Point architecture.

The primary role of the is to enable or block communication. Figure 4.13(b) illustrates the AP implementation for verifying packet authenticity and managing access to the OSZ. In this diagram, the elements depicted as blue squares – *k1Reg*, *k2Reg*, and *THR* – are inputs derived from the MMMRs, as shown in Figure 4.13(a). The registers *k1Reg* and *k2Reg* hold the values of the authentication keys, while *THR* represents the threshold at which key renewal is triggered. Additionally, the registers $C_{in}$ and $C_{out}$ are responsible for tracking the count of packets entering and exiting the OSZ, respectively.

The first packet flit has a tuple {packet type, obfuscated *k*2}. The following flits contain the SR path and payload. For packets entering the OSZ, there are three authentication layers (three inputs in the AND gate A1):

1. *Packet from a peripheral*: The system differentiates communication from PE↔PE and PE↔IO using a 4-bit identifier in the packet header. When the first flit of the packet is received at the *data_in* point, its four MSBs are compared against the *IOflag* to verify if it is a packet from a peripheral.

[1]The data NoC has duplicated physical channels (Section 3.1.1). Only one channel receives an AP.

2. *Successful authentication*: For a packet to be authenticated successfully, $k2$ must correspond with the *k2Reg* value. This process involves the first flit of the packet arriving at *data_in*, where its 12 LSBs undergo an XOR operation with *k1Reg*. The result of this operation is then compared with the *k2Reg* to confirm authentication.

3. $C_{in} < C_{out}$: Due to the master–slave communication protocol, the number of received packets cannot exceed the transmitted packets.

Besides that, Figure 4.13(b) also shows that whenever the $C_{out}$ register is equal to the value of the threshold, the signal *renew_key* is activated and interrupts the PE to perform the periodic key renewal. $C_{in}$ increments when the packet satisfies the above conditions and enters the OSZ. Outgoing packets traverse the AP and $C_{out}$ increments. There is no need to verify their authenticity at the AP (only at the SNIP).

### 4.3.4 IO API

Table 4.1 summarizes the services supported by the IO API. The first column refers to the name of the service, the second column refers to who generates the packet with the service, and the third column refers to the function related to the service.

Table 4.1 – Services supported by the IO API.

| **Service code** | **Packet Source** | **Function** |
|---|---|---|
| IO_INIT | Manager PE | Packet received at system startup with the initialization key – $k0$ |
| IO_CONFIG | Manager PE | Configure a line of the SNIP Application Table with $\{appID, path, k1, k2, status\}$ |
| IO_RENEW | Manager PE | Renew the *appID* keys $\{k1, k2\}$ receiving parameters $\{n, p\}$ |
| IO_CLEAR | Manager PE | Clear and deallocate the SNIP Application Table row indexed by *appID* |
| IO_WRITE | Application | Write data into an IO device<br>Application waits an IO_ACK from SNIP |
| IO_READ | Application | Request data from an IO device<br>Application waits an IO_DELIVERY packet |
| IO_ACK | SNIP | Acknowledgment related to a write operation in an IO device |
| IO_DELIVERY | SNIP | Data from an IO device |

The IO_INIT service is generated by the MPE only at system startup, to generate the $k0$ keys for each system component. The IO_CONFIG, IO_RENEW and IO_CLEAR services are related to SNIP management, with the MPE kernel being the only component that generates these packets. The last four services are related to communication between $App_{sec}$s and peripherals.

## 4.4 Results

This section is dedicated to assessing the impact and effectiveness of the security measures proposed in SeMAP. It aims to demonstrate their efficacy in mitigating attacks such as Denial of Service and Spoofing. To achieve this, the results provide an analysis and discussion, contrasting scenarios where SeMAP is implemented against those without such security measures.

Experiments use a 4x4 system, running a secure application (5-task MPEG benchmark), with two tasks communicating with two IO devices, one to read input data and the other to output the results.

The first evaluation considers the cost of the authentication process, key renewal, and overhead for starting the application. Figure 4.14 shows the iteration number executed by the application on the X-axis and the iteration latency on the Y-axis. The baseline curve corresponds to the latency without security mechanisms. The K0 curve corresponds to the execution considering only the authentication process (without key renewal), K5 and K10 correspond to THR equal to 5 and 10 transactions, respectively. The execution time overhead, w.r.t. the baseline system is 1.43%, 2.56%, 1.95% for K0, K5, and K10, respectively. As expected, frequent key renewal (K5) increases the latency overhead, with an average overhead smaller than 3% considering other benchmarks. The average overhead to start the application is 97.08 $\mu$s (9,708 clock cycles @ 100 MHz), with an increase of 1.6% in the total execution time due to the OSZ and AP configuration.

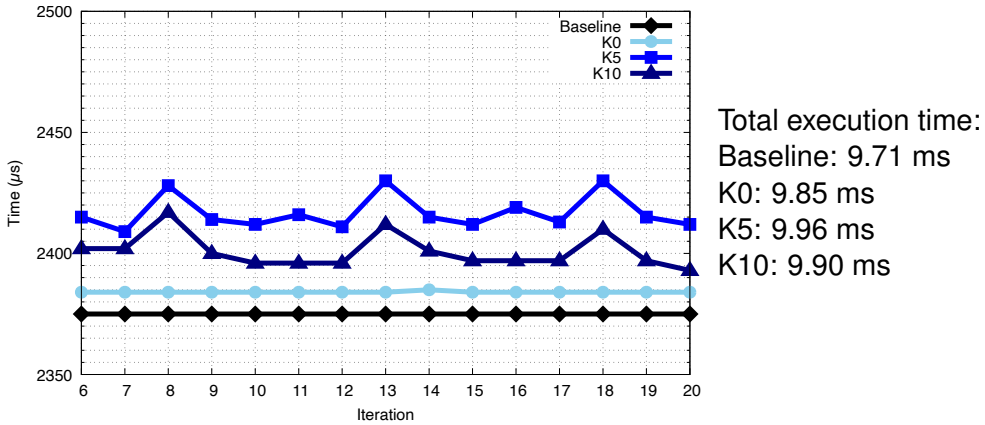



Figure 4.14 – Impact of authentication and key renewal frequency.

In a second evaluation, a malicious task injects invalid packets into the AP coordinate to perform a DoS attack, with 26-flit packets at every 20 $\mu$s@100MHz, corresponding to 1.3% of the link bandwidth (Figure 4.15, iteration 8-14). Despite the reduced injection rate, the baseline system has a 31.6% increase in iteration latency, given that invalid packets arrive at a PE, and the OS must treat and discard them (on average, there are 116 interruptions at each iteration). In the SeMAP proposal, the AP discards the packets without affecting the

iteration latency. This result shows that: (1) it is possible to generate a DoS attack not flooding the NoC but inducing frequent interruptions in processors; (2) the effectiveness of protecting applications against DoS attacks using the proposed mechanisms.

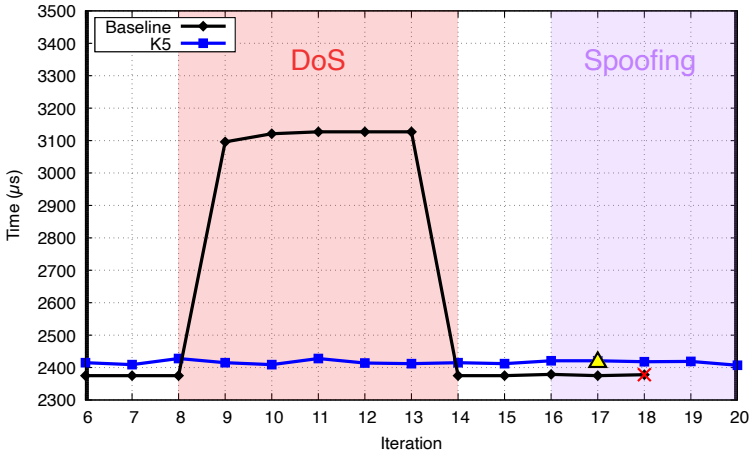



Figure 4.15 – DoS and spoofing attacks on the baseline and K5 (key renewal after five IO transactions) systems. The first five iterations are the simulation warmup period.

Figure 4.15, iteration 16-20, shows the effect of a spoofing attack with forged 26-flit packets at every 75 $\mu$s@100MHz. In the baseline system, the application hangs in this experiment if the forged packet arrives while the task waits for an IO packet (iteration 18). For SeMAP, we assume that the forged IO packet arrives at the AP satisfying condition $C_{in} < C_{out}$, with a forged f1. In this case, the attack is detected at iteration 17. The packet arrives at the PE, with the OS discarding it due to the failed f2 authentication, starting a key renewal, and re-executing the IO transaction. The cost is negligible, 18 $\mu$s. For an attack to succeed, it is necessary to get the exact time the application is waiting for an IO packet and correctly forge AppID, k1, k2. Given the reduced time between key renewals, the attacker does not have enough time to perform a spoofing attack successfully.

### Area Evaluation

Table 4.2 presents the results of the logic synthesis, using Cadence Genus - 28 nm technology library, to assess the hardware impact of the security elements added to the system. The synthesis includes the Secure Router, Control Router, and SNIP. Each processing element (PE) has two routers (data and control routers), a processor, local memory, and a network interface. The SNIP is configured with an application table with four rows, and input/output buffers for 16 slots for 16-bit flits. The Data Router has two disjoint channels, with 16-bit flits and 8-flit input buffers. The Control Router has a CAM size with 8 slots.

Table 4.2 shows that the SNIP has a low area overhead, representing 55.6% of the data router area. Comparing the baseline router area (17,973 $\mu m^2$) to the new communication infrastructure (control and secure router – 26,766 $\mu m^2$), the area overhead corresponds to 48.8% in the communication infrastructure.

Table 4.2 – Synthesis results - 28nm FDSOI - CADENCE GENUS 21.12-s068.

| **Synthesis Results** | SNIP | Control Router | Secure Router – Figure 4.13(a) | | | |
|---|---|---|---|---|---|---|
| | | | Data Router | 1 AP (Avg) | All LCs + Regs | Total |
| Cell Count | 3,666 | 3,203 | 7,049 | 169 | 50 | 7,776 |
| Cell Area ($\mu m^2$) | 8,665 | 4,656 | 13,768 | 291 | 86 | 15,019 |
| Net Area ($\mu m^2$) | 2,470 | 2,090 | 4,204 | 81 | 472 | 5,000 |
| **Total Area** ($\mu m^2$) | **11,135** | **6,747** | **17,973** | **372** | **558** | **20,019** |
| Timing Slack (ps) @500 MHz | 373 | 29 | - | - | - | 136 |
| Total Est. Power (mW) | **3.95** | **2.69** | **-** | **-** | **-** | **7.46** |

The communication infrastructure represents no more than 20% of the PE area (in [Rovinski et al., 2019], the NoC represents 7.7% of the PE area). Thus, it is expected that an increase of 50% in the communication infrastructure corresponds to an increase in the PE area between 5% an 10%.

## 4.5 Final Remarks

In this Chapter, we introduced SeMAP, a methodo designed to enable secure applications ($App_{sec}$) to communicate securely with IO devices. The primary security concern SeMAP addresses is the vulnerability inherent in the exposed path between the OSZ and the IO device. SeMAP addresses this vulnerability through a combination of hardware and software components:

- Hardware Components: APs and SNIPs. These components safeguard the communication endpoints.
- Software Components: Logical partitioning of the MCSoC into gray and protected areas, the implementation of an authentication protocol, and the mechanism for key renewal.

The SeMAP proposal effectively addresses issues related to communication with peripherals, i.e. entities *outside* the OSZ. While the OSZ is designed to be “closed”, with dedicated processing and computing resources exclusively allocated to a specific application ($App_{sec}$), the potential for internal attacks within the OSZ, possibly produced by Hardware Trojans (HTs), cannot be disregarded. The next Chapter shifts the focus to this *inside* perspective of the OSZ. It proposes a solution aimed at securing the communication of tasks belonging to $App_{sec}$, thereby strengthening the internal integrity of the OSZ against such threats.

# 5. SESSION MANAGER

This Chapter presents the *third original contribution* of this Thesis, the Session Manager [Faccenda et al., 2021], fulfilling objectives *SG5* and *SG6*. This mechanism is designed to monitor, detect, and recover the system against attacks or faults that disrupt the packet delivery via the data-NoC (e.g., an HT inserted in a router or link that drops packets). The motivation for the Session Manager is the system vulnerability related to HTs. The Security Manager (set of security procedures executed at the MPE) does not know the presence of an HT, and in this case, an OSZ may be created with infected routers. The main publication associated with the Session Manager is:

> Detection and Countermeasures of Security Attacks and Faults on NoC-Based Many-Cores
> Faccenda, Rafael; Caimi, Luciano; Moraes, Fernando Gehm.
> IEEE Access, v.9, pp. 153142 - 153152, November 2021.

The objective of the Session Manager, protocol executed at the PE level, is to supervise the sending and receiving of packets in the data-NoC. Every time the PE sends a data packet, it also sends a control packet via control-NoC simultaneously to the same target. This control packet carries the communicating pair unique identifier, named *session key*, or $K_s$, which enables the packet receiver to verify its authenticity using $K_s$ and confirm the data-packet arrival. If the target PE detects any violation of the session protocol, it activates the recovery protocol that requests the source PE to resend the data packet, avoiding the original path. As the data NoC supports source routing, the control NoC creates a new path, circumventing the affected region.

The following sections describe the threat model (Section 5.1), the Monitoring Protocol (Section 5.2), the Detection Protocol (Section 5.3), the Recovery Protocol (Section 5.4), the evaluation of the Session Manager (Section 5.5), and the final remarks (Section 5.6).

## 5.1 Threat Model

As mentioned above, one identified vulnerability of the OSZ is the presence of HTs that can block routers inside of the reserved area. Figure 5.1 illustrates an example of an HT executing a DoS attack. Figure 5.1(a) presents a 3x3 system with two communicating tasks: T1 and T2, and a deactivated HT in a router of the path. Figure 5.1(b) illustrates the activation of the HT that blocks the router. Consequently, the router is now unable to receive and send packets, inducing the DoS attack (Figure 5.1(c)).

In the scope of OSZs, the scenario represented on Figure 5.1 can happen inside a secure zone, as the System Manager has no information about the presence of an infected resource. Even without the possibility of having a malicious application running in the OSZ,

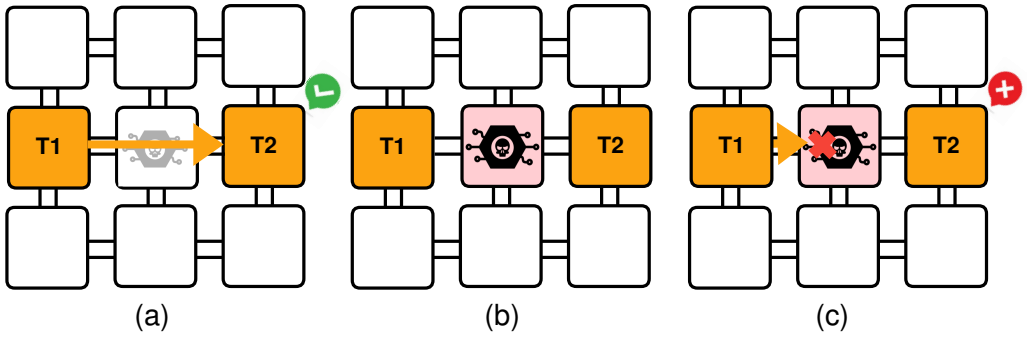



Figure 5.1 – Hardware Trojan affecting communication. (a) T1 communicating with T2, inactive HT in between. (b) HT activation. (c) HT blocking the packet transmission from T1 to T2.

internal conditions can trigger the HT, such as a timer or another physical condition of the system.

Therefore, HTs can attack OSZs internally, regardless of the triggering model. Assuming the Data NoC as a 3PIP, it can be infected by HTs, requiring countermeasures to avoid or mitigate the attacks. The *threat model* of the Session Manager assumed as trusted the control NoC, PE, and OS. Thus, considering the following HT attacks [Philomina, 2021]:

- Packet Loss: one or more of the routers are dropping packets. Therefore, the target never receives the message, and the tasks are left waiting, blocking their execution. This is a DoS-type attack and is also known as a *blackhole* attack [Daoud and Rafla, 2019b].
- Packet Misrouting: one or more routers change the packet header, sending it to the wrong destination. Consequently, the target PE stays blocked, waiting for the requested packet, as in the previous attack.
- Port Blocking: one or more ports of the routers cannot send or receive packets, making them stall and causing contention. In this case, the blocking can be temporary, affecting only the packet latency, or it can also be permanent, blocking the application.

## 5.2 Message Exchange Monitoring

Message exchange monitoring is responsible for detecting problems related to attacks during the packet transmission. The monitoring process of the Session Manager starts by establishing a *session* (Definition 4). Figure 5.2(a) presents the sequence diagram with each step of the protocol, from its creation up to its end.

**Definition 4.** *Session*: establishment of a virtual connection between a producer-consumer pair, using the control NoC. The session is defined by a unique identifier, known only by the communicating pair.

**Definition 5.** *Session key* ($K_s$): unique identifier for a communicating pair, represented by the tuple $\{rnd, ID_p, ID_c\}$, being *rnd* a random number, $ID_p$ the producer task identifier, and $ID_c$ the consumer task identifier.

The OS of each PE has a table to store $K_s$s. In fact, $K_s$ identifies a given session, not following the "key" concept adopted in cryptography. We decided to keep this nomenclature since it provides security to the *session*.

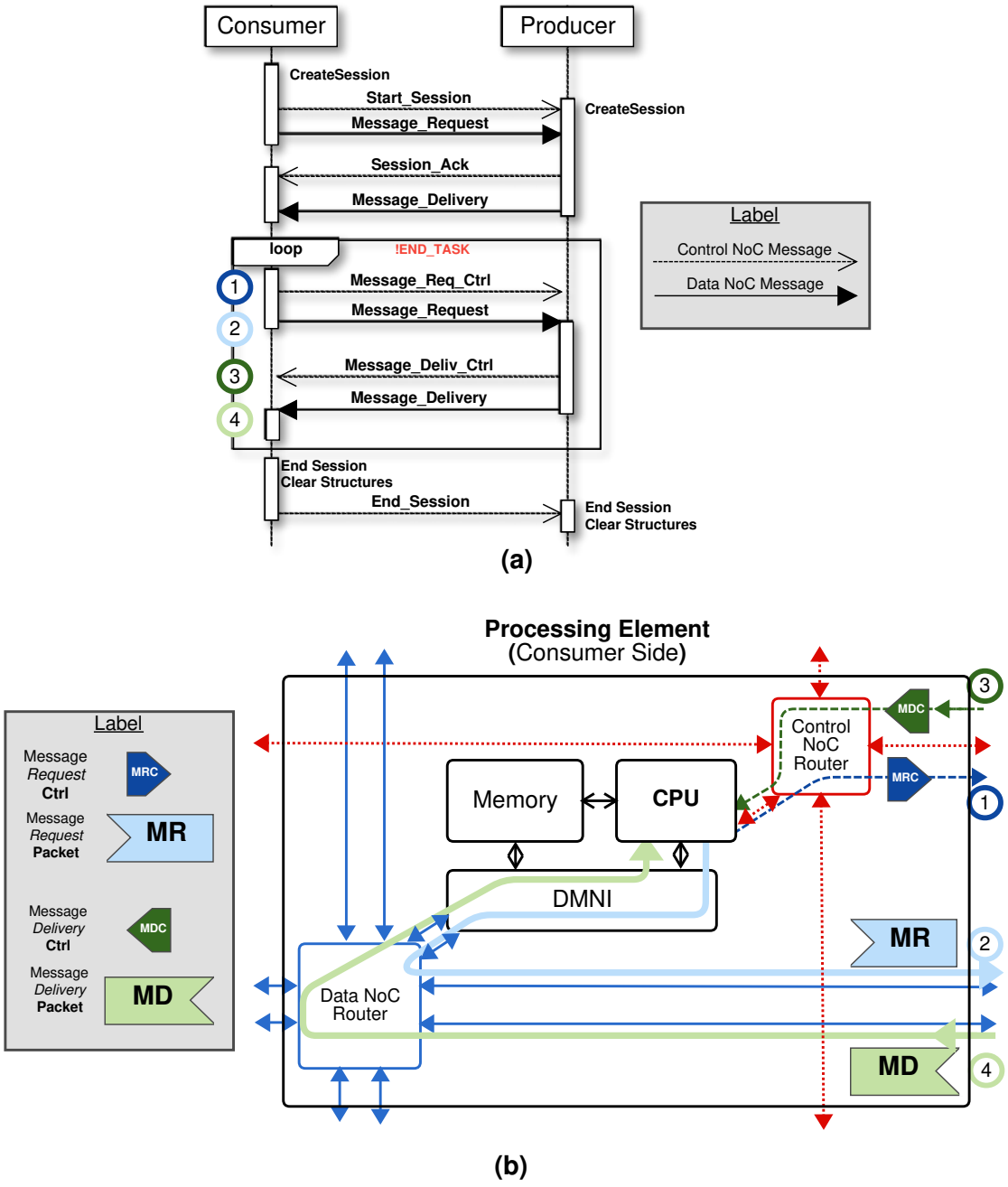



Figure 5.2 – (a) Sequence diagram of the Session Manager operation between the consumer and producer tasks. (b) PE internal organization and the path taken from each step of the protocol. Numbers 1-4 indicate each of the protocol steps temporally (a) and spatially (b).

The consumer task starts the session protocol in the first `Message_Request` packet to be transmitted in the data NoC. Before sending the `Message_Request`, the kernel creates the $K_s$ (Definition 5) with the producer and consumer identifiers and *rnd*. Then, through the control NoC, the consumer sends a `Start_Session` packet to the producer, with $K_s$ in its payload. When receiving this packet, the producer then creates the session at the producer side using *rnd* and the task identifiers from the received $K_s$. When the producer delivers this first requested message, it also transmits the `Session_Ack` control packet confirming the successful session creation.

Following the sequence diagram of Figure 5.2(a), after the successful session creation on both sides of the communication, the tasks exchange messages as depicted inside the *loop* section of the diagram:

1. `Message_Req_Ctrl`(`MRC`): Control message from the consumer task to the producer task, indicates to the producer task an `MR` en route. Contains Ks and the sequence number of the transaction to verify the correctness of the packet.
2. `Message_Request`(`MR`): Data packet holding the `Message_Request` service that asks for the data from the producer task.

The OS only handles `Message_Request` if both messages arrive. Once confirmed the arrivals, the OS searches for messages to the requesting task stored in the pipe. If a message is found, the delivery starts immediately, else the OS registers the `Message_Request`, and as soon as the data is ready, it is sent to the consumer task. The delivery is also composed of two messages:

3. `Message_Deliv_Ctrl`(`MDC`): Control message from the producer task to the consumer task, indicates to the consumer task an `MD` en route. Contains the same information of the `MRC` .
4. `Message_Delivery`(`MD`): Data packet with `Message_Delivery` service that carries the data from the producer task to the consumer task.

The data packet (`MR` or `MD`) enables the OS to retrieve part of $K_s$: $\{ID_p, ID_c\}$ – steps 2 and 4. The control packet (`MRC` or `MDC`) contains the *rnd* value – steps 1 and 3. To validate a data packet, the OS retrieves from the $K_s$ table a line matching the received $\{ID_p, ID_c, rnd\}$. The OS accepts the data packet *iff* the received values match with some line of the $K_s$ table.

Packets transmitted in both NoCs may arrive in any order. If the data packet arrives before the control packet, the OS stores it up to the reception of the corresponding control packet. The same scenario occurs in the opposite reception order.

Step four then goes back to step one, until there are messages to be transmitted. Once the consumer finishes its execution, it closes the current session sending an

`End_Session` message via control NoC to all tasks that produce data to this task. The producers close the session on their side when they receive this `End_Session` message, clearing all values used by the protocol.

Two important issues in a message exchange protocol are: (*i*) correct reception order; (*ii*) network congestion by transmitted but not consumed packets. The message exchange protocol adopted in the OS of the platform addresses these two issues. Data transmission does not freely inject packets into the network, it stores them in the OS pipe until a consumption request, that is `MR` packet. Thus, a packet injected into the network is consumed by the receiver, ensuring message ordering and avoiding network congestion.

## 5.3 Detection

Three situations signalize a suspicious behavior to the OS: (*i*) a mismatch when comparing the $K_s$ from data and control packets; (*ii*) an unexpected packet arriving at the data NoC without a previous message request; (*iii*) a timeout in the reception of the data or control packet.

The Session Manager, thus, handles the attacks of the threat model (Section 5.1) as follows:

- **Packet Loss**: the receiver PE knows that a data packet should arrive due to the reception of a control packet. The Session Manager uses a dynamic timeout mechanism to detect this type of event.
- **Packet Misrouting**: for the receiver PE, the effect is similar to a packet loss. However, the misrouted packet goes to a PE that was not expecting data, and as this PE did not received a control packet, it discards the packet and signalizes the incorrect reception.
- **Port Blocking**: this attack may be permanent or intermittent. If it is permanent, it is similar to a packet loss. If the attack is intermittent, the data packet may arrive with a latency higher than a threshold, which is also detected by the timeout mechanism.

To increase the ability of the method to detect attacks and faults, the control NoC may embed in the payload other parameters, such as a Message Authentication Code (MAC), that could detect corrupted packets.

### 5.3.1 Dynamic Timeout

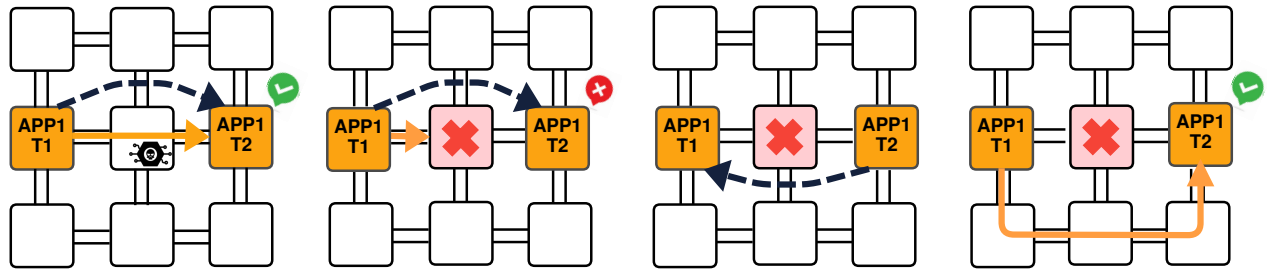
The detection of attacks adopts a timeout mechanism. This timeout is implemented in hardware as a counter that interrupts the OS when reaches zero. The threshold value

that is loaded into the counter is dynamic and it is different for each session. The value is computed according to the Session average latency. The initial value is estimated based on the number of hops between the communicating pair. The warm-up iterations use this value in the first five iterations while registering the latencies of those communications. After the warm-up period, the average latency is used as the threshold and is updated at each iteration.

The *Timeout Monitor* is activated whenever a `MRC` or `MDC` is received. Then it loads the threshold value of the respective session into the timeout counter, which will interrupt the processor if the corresponding `MR` or `MD` does not reach the PE, triggering the Recovery Protocol.

## 5.4 Recovery Protocol

Figure 5.3 presents the Session Manager Recovery Protocol. After detecting an attack or fault through the *Timeout Monitor*, the target PE starts the recovery process. Here, the control NoC also plays a major role in finding a new path. The target PE sends a **TARGET_UNREACHABLE** message to the source PE via the control NoC, informing the $K_s$ of the affected session. Then, the source sends a **SEARCHPATH** service, which uses the broadcast characteristic of the control NoC to find a new path. Then, the control NoC finds this new path via a **BACKTRACK** service, that returns to the source (that previously sent the **SEARCHPATH**).




(a) Successful message transmission.

(b) Data transmission interrupted in the data NoC.

(c) Request for packet retransmission using the control NoC.

(d) Successful retransmission using source routing.

**Dashed** arrows: packets transmitted in broadcast using the control NoC.
**Straight** arrows: packets transmitted through the data NoC.

Figure 5.3 – Representation of the message recovery protocol using the control NoC.

As there is no information related to the HT or fault precise location, the method searches a new path that avoids the routers of the broken path (previous path, which did not deliver the packet correctly). To accomplish this goal, the **SEARCHPATH** message carries two extra configurations: (*i*) the blocked output port at the source PE; (*ii*) the blocked input

port at the target PE. Defining those two blocked ports, the path search method avoids the routers used in the broken path. The hop number of the new path can be non-minimal, as presented by the examples.

Figure 5.4 illustrates two examples of the new path discovery with blocked ports. Red arrows represent the broken paths, which are the ones that are not transmitting the packet correctly – detection by the timeout mechanism. The triangles with a red outline are the ports blocked in the **SEARCHPATH**. The black triangles correspond to the available ports for the new path. The orange arrows indicates the new path. In Figure 5.4(a), the broadcast was not transmitted by the source east port, reaching the target west port due to the blocking of the north port. In this case, the number of hops remains minimal. In Figure 5.4(b), the broadcast excluded the source east port, and reached the target north port. In this case, the new path has four hops and is non-minimal, considering that in the original path, the routers were aligned and had only two hops of distance.

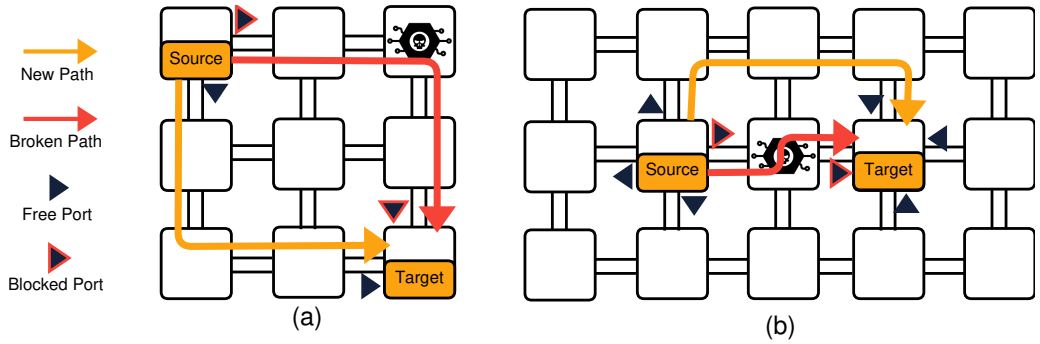



Figure 5.4 – Two examples of Hardware Trojan affecting communication and the Session Manager finding alternative paths.

With the new path, the source PE searches for sent packets with the $K_s$ information received in the **TARGET_UNREACHABLE**, including whether the lost packet was a `MR` or `MD`. Then, the packet retransmission occurs now with source routing (SR). All subsequent packets to the target PE, even those not from the same session, will use this path up to the detection of a new event. Besides that, once defined the new path, there is no additional overhead in the producer-consumer communication, excepting a slight increase in the latency if the SR path is longer than the previous one.

Note that the focus of the Session Manager is not the location of the HT(s) or faulty router(s). The method uses a path search approach that avoids the previous path, not considering any information about specific routers to avoid.

The current implementation of the recovery protocol has a relevant limitation: the non-registration of blocked ports. If the new path also passes through an infected router, the *TimeoutMonitor* will fire another **TARGET_UNREACHABLE** to the source PE informing that the packet still has not arrived. Then, the recovery protocol acts the same way as before, blocking the ports of the used route. Using the example of Figure 5.4(b), the **SEARCHPATH** would block now ports North for both source and target PEs, releasing the previously blocked

ports. The resulting new path would be the original one, with an HT. This limitation makes the protocol resend the packets indefinitely between these two routes.

A solution would be to launch the **SEARCHPATH** blocking the ports of all the previous broken paths, but it would severely limit the path options. As an example, it would fix the issue on Figure 5.4(b) by finding a route using the south ports, but would not work on Figure 5.4(a) because the routers are placed on the corners, having only two other neighbors. For this reason, a more elaborate solution is required, enhancing the path construction algorithm to include information about the system to avoid certain routers besides blocking the ports.

## 5.5 Results

This section presents the costs related to the Session Manager in terms of application execution time overhead (Section 5.5.1), overhead on the session handling routines (Section 5.5.2), and analysis of the recovery protocol impact on applications (Section 5.5.3).

Results are gathered simulating seven applications: DTW, MPEG, MWD, MPEG4, Dijkstra, VOPD, and AES in a 4x4 *MCSoC*. These applications are used in the many-core research community [Costa et al., 2021, Kashi et al., 2021]. Appendix B presents the CTGs of the applications used in this Chapter. The decision for a 4x4 *MCSoC* is related to the size required for the applications to run in single-task mode, i.e., one task per PE.

The applications have distinct communication models, such as pipeline and master-slave, to present a broader view of the impact of the Session on application execution. The MPEG application follows a pipeline communication model. The AES application follows a master-slave model, with one task responsible for distributing the computation to other tasks. The AES application is parameterizable in the number of slave tasks (4 or 8) and 16-byte blocks to encrypt (8 up to 512). For example, AES 4 requires two iterations to encrypt or decrypt a 128-byte message (8 blocks of 16 bytes), while AES 8 requires only one iteration for the same workload.

The CPU, memory, DMNI, control NoC, and the OS are considered reliable entities. Third-party entities, such as the data-NoC and applications, are considered unreliable.

### 5.5.1 Application Overhead Results

This section presents the overall impact of the Session Manager on the application execution time. In this step, no attacks are considered. Table 5.1 compares the application execution time between the baseline system and the one with the Session Manager. The

first column corresponds to the Application. Each line of AES 4 and AES 8 corresponds to a different number of blocks to be encrypted, from 8 to 512.

Table 5.1 – Applications execution time with and without the protocol.

| Application | | Baseline (ms) | Session (ms) | Overhead (%) |
|---|---|---|---|---|
| DTW | | 36.29 | 37.58 | 3.55 |
| MPEG | | 22.68 | 23.54 | 3.79 |
| VOPD | | 3.19 | 3.94 | 23.51 |
| MWD | | 2.51 | 3.05 | 21.51 |
| MPEG4 | | 23.14 | 29.61 | 27.96 |
| Dijkstra | | 5.65 | 7.56 | 33.81 |
| AES 4 | 8 | 2.72 | 3.21 | 18.01 |
| | 16 | 3.98 | 4.79 | 20.35 |
| | 32 | 6.49 | 7.75 | 19.41 |
| | 64 | 11.58 | 13.96 | 20.55 |
| | 128 | 21.70 | 26.19 | 20.69 |
| | 256 | 41.99 | 50.79 | 20.96 |
| | 512 | 72.59 | 89.69 | 23.56 |
| AES 8 | 8 | 3.13 | 3.81 | 21.73 |
| | 16 | 4.14 | 5.09 | 22.95 |
| | 32 | 6.16 | 7.65 | 24.19 |
| | 64 | 10.28 | 12.90 | 25.49 |
| | 128 | 18.31 | 23.29 | 27.20 |
| | 256 | 34.55 | 43.82 | 26.83 |
| | 512 | 68.05 | 86.23 | 26.72 |

Two applications follow the pipeline communication model, *MPEG* and *DTW*. The execution time overhead for these applications corresponds to 3.79% (*MPEG*) and 3.55% (*DTW*). This small performance overhead is due to the communication model. Only one message is exchanged between each communicating task pair per iteration, requiring one session synchronization per iteration.

The CTG of the remaining applications follows a master-slave model (*AES*, *Dijkstra*), or has a complex CTG with many inter-task dependencies (*MWD*, *MPEG4*, *VOPD*). For these applications, the execution time overhead ranges from 18.01% to 33.31%. The execution time overhead for these applications is higher due to the number of messages the master task(s) needs to synchronize. The protocol synchronization directly affects the execution time. When a task has many communication dependencies, the synchronization delay is propagated and impacts the sending or receiving of subsequent messages to the next tasks.

The AES application illustrates this effect of protocol synchronization. The increase in the number of slave tasks, from 4 to 8, implies a higher execution time overhead due to the large number of messages to synchronize. The number of iterations the application executes

(the number of blocks to be handled divided by the number of slave tasks) shows the effect of the synchronization propagation. The AES_4 stabilizes the execution time overhead at approximately 24% from 16 iterations (64 blocks). For AES_8, the execution penalty also stabilizes at 16 iterations, but for 128 blocks, at approximately 27%.

These results define the protocol execution time overhead according to the communication characteristics of the application. *We consider that the overhead of up to 33.81% on application execution time is an acceptable cost considering the security and fault tolerance benefits added by the communication session protocol.*

### 5.5.2 Kernel Overhead

The Session Manager increases the computation of the OS to handle the protocol packets. There are two new algorithms: handling the control (`MRC` and `MDC`) packets. In addition, the routines to handle `MR` and `MD` packets were modified to insert the verification with the control NoC.

To analyze the impact of the protocol in the OS, Table 5.2 shows the time (in clock cycles) taken by the OS to handle the messages with the Session Manager, compared to the baseline implementation, considering 128 iterations of the application. The values refer to the AES 4 with 128 blocks simulation, considering two cases:

- Case 1: control packets arrive before data packets, observed in the AES Slaves.
- Case 2: data packets arrive before control packets, observed in the AES Master. This happens because the master receives the `MR` and `MD` messages from the four slaves almost simultaneously. The OS prioritizes the handling of the data NoC packets to avoid network congestion.

Table 5.2 – Average overhead (in clock cycles) for each service compared to the baseline implementation.

| Service | Case | Baseline (Data) | Session (Data+Control) | Overhead |
|---|---|---|---|---|
| REQUEST | Case 1 | 443.0 | 679.7 | 53.44% |
| | Case 2 | 466.7 | 810.2 | 73.61% |
| DELIVERY | Case 1 | 227.0 | 324.9 | 43.14% |
| | Case 2 | 222.7 | 373.0 | 67.48% |

Results show that the second case presents a higher overhead for both services because data packets arrive before control packets. In this case, the packets arriving at the

data NoC need to be stored and then retrieved when the control packets arrive to validate them.

This experiment also showed the difference between services: the REQUEST takes longer than the DELIVERY. This happens because the REQUEST is also responsible for sending the packet (i.e., execute the DELIVERY) if the producer already has the packet ready in the pipe when the REQUEST arrives.

Table 5.2 shows that message exchange transactions have a relatively high percentual overhead, but small if we consider it in clock cycles (less than 400 cycles in the worst case). Thus, applications with a pipeline model (such as MPEG) have a minor execution time penalty adding the proposed protocol, as shown in Table 5.1. On the other hand, due to the serialization in handling messages in applications with a master-slave communication model (such as AES), this overhead accumulates, explaining the higher runtime overhead.

### 5.5.3 Recovery Costs

After detecting a suspicious behavior, the recovery process starts, as detailed in Section 5.4. The recovery process adopts a rerouting mechanism. With a new path established, all subsequent packets use this path. Thus, the overhead of the recovery process occurs once, when detecting the suspicious behavior.

Two scenarios are simulated to evaluate the impact of the rerouting and packet recovery mechanisms: one with a pipeline application (MPEG) and the other with a master-slave application (AES4), both applications having five tasks. The tasks are mapped inside an OSZ that encloses six routers (yellow-highlighted area), one of them infected by an HT. Figure 5.5 illustrates the MPEG and HT mapping, alongside the XY routes for the `MESSAGE_DELIVERY` and `MESSAGE_REQUEST` packets, which are the green arrows on (a) and blue arrows on (b), respectively. The HT activation interrupts three flows: numbers 3, 5, and 8 of Figure 5.5.

Each application executes ten iterations, with the HT configured to block all ports of the infected router at 3 ms. Figure 5.6 shows the time taken for each iteration for both applications. Each graph has three curves:

- *Baseline*, execution without the Session Manager;
- *Session*, execution with Session Manager, without the activation of the HT;
- *Attack*, execution with Session Manager, the HT activation at 3 ms, and the time spent for the recovery process.

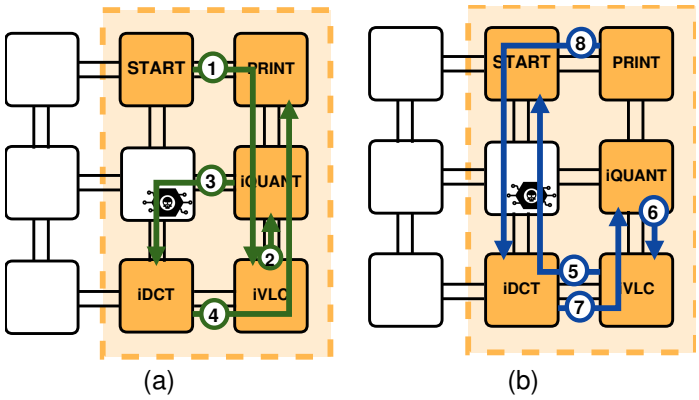



Figure 5.5 – Task mapping of the MPEG application with the inter-task communication on a 3x3 Mesh NoC with a 2x3 OSZ (yellow area). (a) Message Delivery packets. (b) Message Request packets.

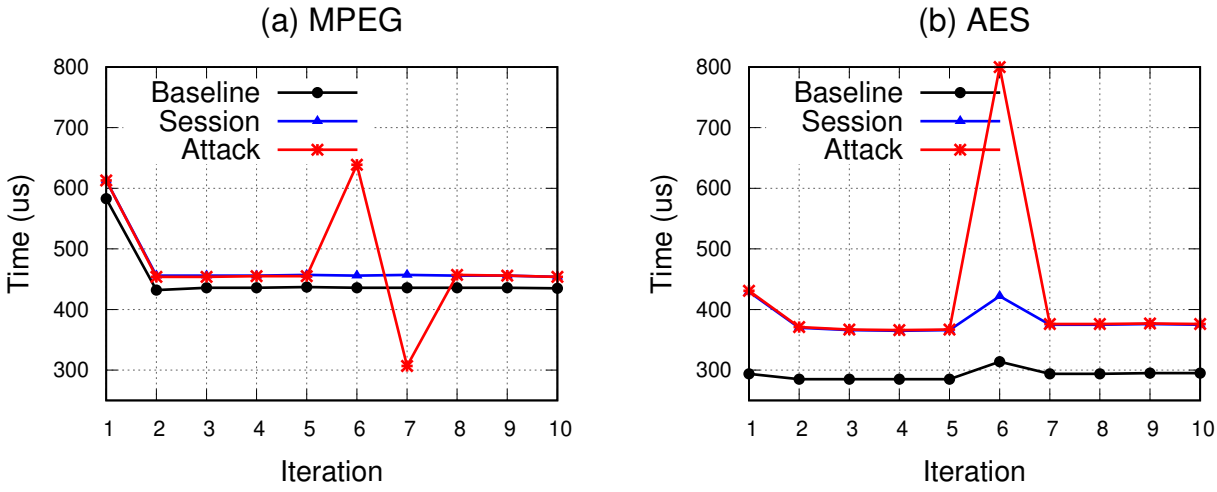



Figure 5.6 – Impact of the Recovery Protocol on applications (a) MPEG (b) AES. X-axis: iteration number, y-axis: iteration latency.

Figure 5.6(a) illustrates the MPEG application. The application stalls at iteration 6, firing the recovery process in parallel at different PEs. The next iteration, after the recovery process, executes faster. Due to the pipeline structure of the application, data remains buffered in the producer PEs. Once the new path is established, the data is transferred to their targets. Figure 5.6(b) illustrates the AES application. This application also stalls at iteration 6. Due to the master-slave communication model, it is not possible to buffer intermediate data. Therefore, it is necessary to finish the recovery process to restore the original latency.

As shown in Figure 5.6, the recovery process overhead happens once. After the recovery process, the HT is still active, but applications are unaffected. Note that the latency after the attack is the same for the ‘session’ and ‘attack’ scenarios. The latency is the same because the new paths have the same number of hops as the original ones.

Table 5.3 presents the execution time for each scenario. The overheads using the Session Manager are according to the ones presented in table 5.1, varying with the communication model. The MPEG application increases its execution time by 0.8% when it is necessary to reconfigure the paths due to the HT attack. The additional overhead of the AES application is 8,85% for the execution of 10 iterations. These overheads reduce as the number of executed iterations increases.

Table 5.3 – Applications execution time with and without the protocol.

| App | Baseline (ms) | Session (ms) | Attack (ms) |
|---|---|---|---|
| MPEG | 5.03 | 5.26 (4.57%) | 5.30 (5.37%) |
| AES | 4.52 | 5.56 (23.01%) | 5.96 (31.86%) |

This technique is an alternative option for those seeking HT location and isolation. The method can detect anomalous behaviors and create a new path that avoids the original path through rerouting.

## 5.6 Final Remarks

This Chapter presented the Session Manager, an original method to detect security attacks and faults in the communication architecture. Proposals available in the literature seek to add security mechanisms to the NoC itself, which may be faulty or under attack. Thus, to avoid instrumentalizing the data NoC itself, we use a control NoC with broadcast transmission to find alternative paths after detecting the attack or fault. The proposed method does not need to precisely locate the source of the problem, which is the objective of most works found in the literature. The cost of our proposal is the increase in the execution time. On the other hand, the Session Manager operates efficiently to recover from attacks, such as packet loss, packet misrouting, and port blocking, caused by HTs or failures in NoC links.

The Session Manager does not exclude methods that locate HTs or faulty routers. Our proposed method, combined with those methods, can simplify the rerouting procedure. Instead of avoiding all routers in the original path, only infected or faulty routers could be avoided in the new path. Furthermore, once an attack is detected, the System Manager can use this information in the next decisions regarding the allocation of secure applications.

Thus, in the context of MCSoCs, this Chapter presented a step towards a unified method to deal with security threats, manufacturing faults, aging, or application constraints (e.g., QoS). Although this work focuses on securing the communication against HT attacks, the method is general and applicable to fault tolerance and QoS constraints.

# 6. FRAMEWORK FOR SYSTEMIC SECURITY MANAGEMENT

This Chapter introduces the *fourth original contribution* of this Thesis: a framework that consolidates all previously discussed security mechanisms, fulfilling objectives *SG7* to *SG9*. This comprehensive framework sends security warnings for the System Manager, also including a set of decision heuristics and countermeasures designed to enhance the system's security. The principal publication associated with this framework is:

> A Comprehensive Framework for Systemic Security Management in NoC-based Many-cores
> Faccenda, Rafael; Comarú, Gustavo, Caimi, Luciano; Moraes, Fernando Gehm.
> IEEE Access, v.11, pp. 131836-131847, November 2023.

The set of defense mechanisms implemented on the platform, described in previous Chapters, include:

**D1** Opaque Secure Zones (Section 3.2);

**D2** Authentication keys, used in the AP $\leftrightarrow$ SNIP communication;

**D3** MAC, protects the binary codes;

**D4** Source routing, obfuscate the source and target address;

**D5** Key renewal, ensures the periodic change in the authentication keys;

**D6** Gray and restricted areas (Chapter 4);

**D7** SNIP, a NI that authenticates the communication with IO devices.

A recurrent observation in the literature is the absence of systemic and integrated security mechanisms that simultaneously monitor, detect and mitigate a broad spectrum of threats in real-time. Based on that, we propose a comprehensive framework for security management that combines the aforementioned defense mechanisms with monitoring methods at several locations, allowing the detection of threats. Threat detection fires countermeasures and generates security warnings to a Security Manager.

The proposed framework adopts an actuation loop based on Monitoring-Detection-Countermeasure, as illustrated in Figure 6.1, which protects the system during the execution of applications with security requirements. This framework integrates into the many-core defense mechanisms, summarized above, with distributed monitoring methods (Section 6.1) that enable the detection of threats and activation of countermeasures (detailed in Section 6.2).

Monitoring mechanisms (**M1-5**) observe system resources and generate warnings (**W1-6**) in case of suspicious behavior. Based on the severity of the alerts, the system triggers countermeasures, which can be local (**C1-5**) or system-level actions. System-level countermeasures are triggered upon detecting a more complex attack (**A1-3**).

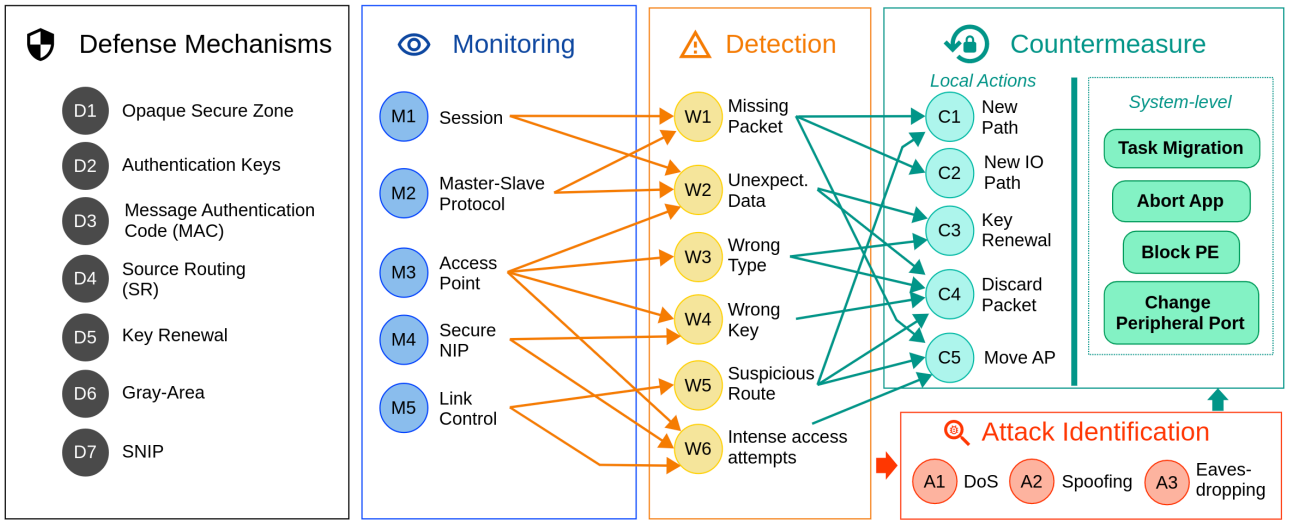



Figure 6.1 – Security management framework overview.

The remainder of the Chapter is organized as follows. Section 6.1 presents the monitoring and warning generation. Section 6.2 explains the warnings and the countermeasures applied by the system. Section 6.3 evaluates the framework in different attack scenarios and evaluates the area overhead due to added hardware mechanisms. Section 6.4 evaluates the proposed framework with an aggressive attack campaign, presenting the countermeasures executed by the framework Section 6.5 concludes this Chapter and points out directions for future work.

## 6.1 Monitoring and Detection of Suspicious behavior

To protect internal communication within the OSZ, we adopted the session protocol (**M1**) (Chapter 5, [Faccenda et al., 2021]). This protocol includes sending control messages via the control NoC alongside the data messages to monitor the arrival of packets inside the OSZ. Packets are only accepted upon receiving both control and data packets that confirm the source and target of this packet. The session protocol can raise two warnings: **W1** (Missing Packet), when only the control message arrives or the data packet is delayed beyond a certain time threshold; **W2** (Unexpected Data) when a data packet arrives without the control message, making it impossible to confirm the source of the message.

The communication API (Application Programming Interface) with IO devices of SeMAP adopts a master-slave protocol, which also works as a monitoring element (**M2**). Any IO transaction must always start from the $App_{sec}$ (master), and the peripheral must always answer this request (slave). PEs monitor the packets' arrivals. Whenever a packet arrives without being requested, the PE raises **W2** (Unexpected Data), or if an answer packet from an IO takes too long to arrive, the PE sends an **W1** (Missing Packet).

The Access Point (AP) (**M3**) monitors all packets trying to enter the OSZ. The AP can raise four warnings:

- **W2** – Unexpected Data: When a packet tries to enter the OSZ without being requested. The AP has two counters, $C_{in}$ and $C_{out}$, which count the number of packets entering and leaving the OSZ. Due to the master-slave communication protocol, the number of received packets cannot exceed the transmitted packets, i.e., $C_{in} < C_{out}$. Whenever this condition becomes false, the AP generates the warning.
- **W3** – Wrong Packet Type: The SNIPs add an identifier in the packets signalizing that it is generated by a peripheral. This warning is raised if the packet does not have this identifier. This monitoring avoids packets generated by applications running on PEs to try to enter the OSZs.
- **W4** – Wrong Authentication Key: The AP executes a lightweight authentication protocol that verifies the packet's authenticity. The warning is raised when the keys do not match, signalizing a forged packet.
- **W6** – Access Attempts: This warning signalize a potential DoS attack. A counter in the AP monitors the number of received packets within an interval defined according to the application profile.

Thus, a packet only enters the OSZ, passing through the AP, satisfying three conditions: (*i*) $C_{in} < C_{out}$; (*ii*) successful authentication; (*iii*) packet from a peripheral.

The SNIP also monitors packets (**M4**). The SNIP authenticates packets, sending warning **W4** if the authentication fails. In addition, the SNIP can raise **W6** if the number of packets received within a time window is larger than a given threshold.

The last monitoring element is the Link Control (LC) (**M5**). Packets should not arrive at enabled LCs due to the routing method, which circumvents the OSZs. Thus, LCs generate a **W5** (Suspicious Route) warning for any packets arriving at an activated LC, especially if the packet is trying to leave the OSZ (this only may occur if an HT infected the secure application). LCs also generate warnings when packets attempt to enter the OSZ from a router outside the boarder, signalizing a possible DoS attack (**W6**).

## 6.2 Countermeasure

Countermeasures are actions that reinforce system security upon the detection of suspicious behavior. Such actions are divided into two groups: local and system-level, which include immediate actions executed upon receiving warnings and actions taken based on broader systemic information, respectively.

The local countermeasures are triggered by receiving warnings, as depicted on Figure 6.1. Depending on the warning severity, a single warning activates the countermeasure, or it is necessary to receive a set of notifications to trigger it. One exception is the Packet Discarding (**C4**) that can happen imediately upon the detection of a suspicious behavior, i.e. packet arriving at a blocked LC.

The **C1** (New Path) action triggers the computation of a new routing path for a message that did not reach the final target detected by the Session protocol (**W1** from **M1**), or detected by the Master-slave protocol (**M2**) in the case of an IO communication.

New IO Path (**C2**) is a countermeasure triggered by **W1** (Missing Packet) emitted by any PE that initiates an IO communication but does not receive the answer within a given time window (**M2** Master-slave protocol). This process is also a tool that can be requested by other countermeasures that affect the IO paths, such as **C5** (Move AP).

Key renewal can be periodic or reactive. The periodic key renewal is part of the authentication method to enhance its security, therefore listed as a defense mechanism (**D5**). Although the keys are not transmitted in plaintext, unauthorized access to the flits with the keys could enable a brute-force attack. The reactive key renewal (**C3**) is a countermeasure to refresh the keys, triggered by **W2** or **W3**. For example, a packet with correct keys arrives at an AP but without a request (**W2**), or wrong type (**W3**). Even though the AP blocked the packet, a key renewal must occur since the authentication keys were correctly forged.

Packet Discarding (**C4**) is the fourth local countermeasure. This is the most frequent countermeasure action due to the OSZ method. The AP of the OSZ discards packets that fail authentication at any layer: whether due to the absence of request (**W2**), wrong type (**W3**), or incorrect key (**W4**). Additionally, activated LCs discard packets that attempt to cross it (**W5**). Moreover, the SNIP discards packets arriving with incorrect keys, and PEs still may discard packets considered suspicious.

Move AP (**C5**) countermeasure is triggered when the MPE receives a **W6** warning from an AP, meaning that heavy traffic, probably malicious, on the AP is affecting the IO communication of an $App_{sec}$. **W1** also may trigger the Move AP to change the route of a packet that could not reach the peripheral during an IO communication. The MPE then elects new AP location and triggers **C3** to refresh the keys and **C2** to recalculate the IO path since the AP coordinate changed. The Suspicious Route warning (**W5**) collected throughout the application execution time can be used as information to avoid mapping the AP at ports that could have been under attack.

## 6.3 Security Analysis and Costs

This section explores attacks identified in the Threat Model, including Denial of Service (DoS), Spoofing, and Eavesdropping. It outlines the system's responses in such events (refer to fig. 6.1) and discusses the associated costs of implementing countermeasures.

The many-core system is modeled at the RTL (Register-Transfer Level) level using VHDL and SystemC hardware description languages, meaning that the many-core description is synthesizable for FPGAs or ASICs. Thus, the accuracy achieved in the experiments is at the clock cycle level, reflecting the actual system behavior. Applications run in the digital simulator for a few dozen milliseconds due to the complexity of the low-level simulation.

We adopt two mechanisms to execute attacks. A peripheral named "packet injector" is directly connected to the system without using the SNIP. The goal of using this peripheral is to inject controlled traffic into the system to emulate attacks. The second mechanism is an HT circuit (based on [Weber et al., 2020]) connected to routers. The HT trigger may be a malicious application or a given condition (e.g., time-triggered HT).

Simulations focus on five specific attack scenarios:

A. *DoS Flooding*: In this attack scenario, the "packet injector" transmits packets at a high throughput rate of 0.85 flits per clock cycle to an OSZ or SNIP. The objective is to saturate the NoC links and buffers to render the NoC unavailable for legitimate operations.

B. *Spoofing*: This attack is analogous to the DoS flooding attack but operates at a lower injection rate of 0.05 flits per clock cycle. Furthermore, the injected packets contain the correct keys to bypass the AP or SNIP, simulating a situation where the keys have been compromised.

C. *Eavesdropping*: a time-triggered HT infects a router inside the OSZ. The attack initiates after 5 milliseconds of simulation time has elapsed, duplicating packets traversing the infected router.

D. *Internal OSZ DoS blocking*: a time-triggered HT infects a router inside the OSZ. The HT blocks all router links for 1 millisecond at regular intervals of every 5 milliseconds throughout the simulation time.

E. *External OSZ DoS blocking*: similar to the previous attack, but in a router belonging to the GA.

DoS - Flooding

Figure 6.2 depicts the first attack scenario, in which malicious flows with forged packets target the SNIP or the AP of a given OSZ.

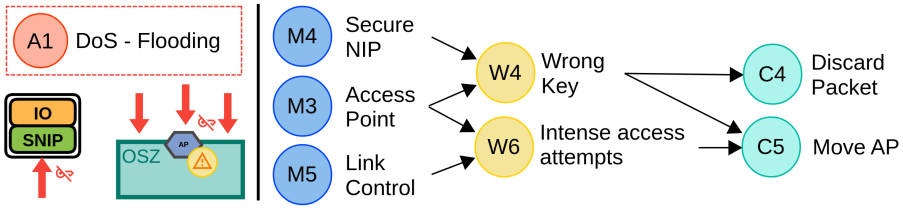



Figure 6.2 – DoS flooding attack scenario.

Packets that arrive in the SNIP without the correct keys are automatically discarded. Malicious packets arriving at the LCs and the AP of a given OSZ are also discarded. However, the **W6** warning signal is triggered if access attempts become too frequent. This alert informs the AP selection heuristic running in the MPE to avoid mapping an AP to this port due to the attack attempt and also initiates a Move AP countermeasure (**C5**).

If the malicious flow bypasses the hardware barrier (AP) and reaches a PE, the Master-slave protocol may identify unexpected data and subsequently discard the packet, as depicted by countermeasure **C4**.

The countermeasure **C4** in hardware instantly discards the packet. Conversely, in software (discard action performed by the PE), the process takes 378 clock cycles (*cc*), measured from the point of packet arrival interruption to the complete clearing of the DMNI slots where the packet was initially stored.

Spoofing

Figure 6.3 illustrates a Spoofing attack where the malicious flow has the correct authentication keys, enabling it to pass through the AP or even gain access to the SNIP.

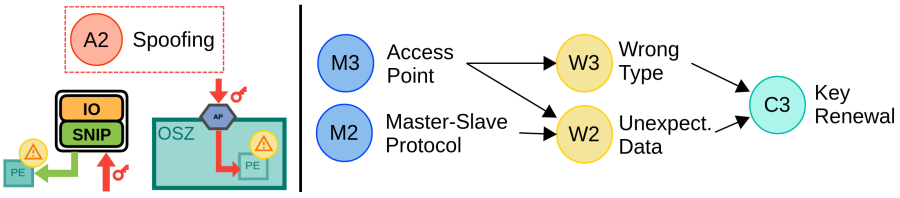
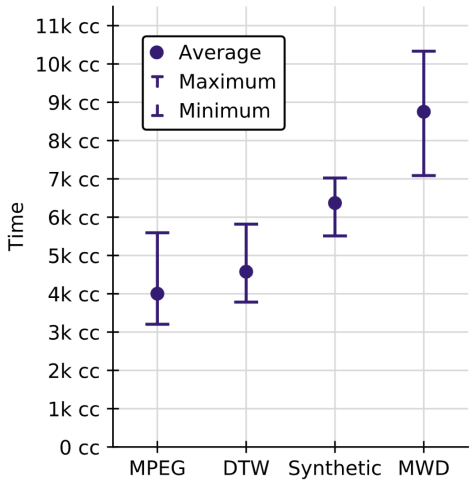



Figure 6.3 – Spoofing attack scenario.

When this malicious flow reaches the SNIP, the Master-slave protocol generates an answer packet to the address stored in the SNIP Application Table. Note that the answer does not go to the attacker but to the registered application. This answer packet reaches a PE in the OSZ, and the PE triggers an unexpected packet warning (**W2**), given that the application is not expecting answers from IO operations.

Another scenario occurs when the malicious packet reaches the AP with the correct keys but either at the incorrect moment or with the wrong type. As a countermeasure in both cases, a Key Renewal **C3** countermeasure is executed to refresh the keys and prevent further unauthorized access.

Figure 6.4 presents the Key Renewal cost, in clock cycles (*cc*), for four applications. The graph on (a) shows the average, maximum, and minimum time taken for each application to perform a key renewal. The values can be different, even for the same application, because key renewal costs have fixed and variable components. The fixed part refers to the time to process the key renewal request and decide on the new key renewal parameters. The variable component is related to synchronization costs, which includes the time taken to receive the key renewal acknowledgment from all tasks of the application plus the time needed for the tasks to finish their pending IO transactions before changing the keys.




(a)

| App | # Tasks | Comm. Vol. |
|---|---|---|
| MPEG | 5 | Low |
| DTW | 6 | Low |
| Synthetic | 6 | High |
| MWD | 12 | High |

(b)

Figure 6.4 – Key renewal overhead for different applications.

For the two applications with low IO communication volume (MPEG and DTW), the average time for key renewal is 4,000 and 4,500 clock cycles, respectively. Increasing the IO communication volume (Synthetic) and the number of tasks (MWD) directly impact the Key Renewal execution, reaching average values of 6,370 and 8,753 clock cycles, respectively. The fixed component of the cost corresponds to 600 cc (average values).

### Eavesdropping

Figure 6.5 presents an example of an Eavesdropping attack, where a malicious hardware in the data NoC duplicates a packet to send it outside the OSZ. However, this packet hits an LC or the AP at the OSZ border from inside, triggering an alert **W5** to indicate that a packet is on suspicious route. To mitigate this threat, a new path is calculated within the OSZ that avoids passing through the identified suspicious router.

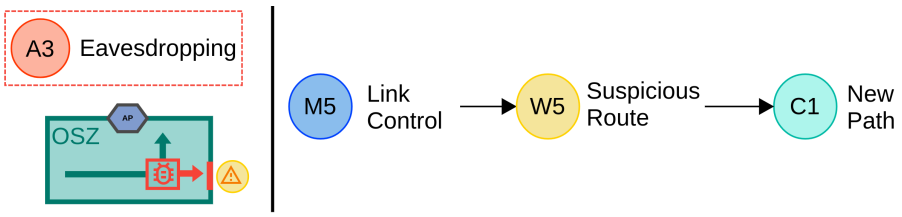



Figure 6.5 – Eavesdropping attack scenario.

The New Path countermeasure uses the Recovery protocol (Section 5.4) of the session manager to build a new path avoiding the previous, now suspicious, route. To achieve this, a *searchpath* message is broadcasted to all PEs in the OSZ, and a *backtrack* message is subsequently received with the correct sequence of hops. Figure 6.6 presents the time to build new paths ranging from 1 to 13 hops.

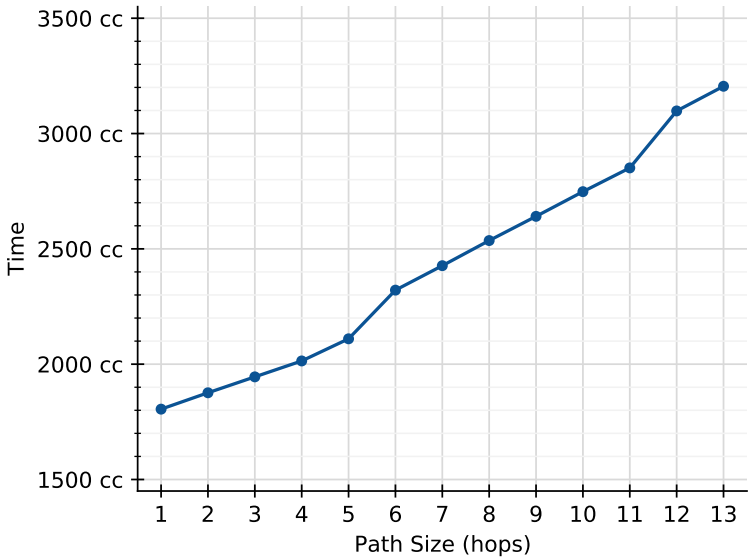



Figure 6.6 – Searchpath overhead for different path sizes.

The overhead curve starts at a path size equal to 1, consuming 1805 clock cycles. It then increments by 70 clock cycles per hop until the hop size reaches 6. From 7 hops, the overhead corresponds to 100 clock cycles per hop until the hop size equals 12. These periodic increases in the curve at every multiple of six are due to the Source Routing (SR). SR uses flits within the packet to store the direction for forwarding the packet. Each flit may store the directions for six hops. Consequently, the cost increases when a new word is required in the SR path.

### DoS-Blocking (OSZ)

Figure 6.7 presents an example of DoS blocking inside an OSZ. In this case, a malicious entity (e.g., HT infecting an NoC router) blocks any communication trying to pass through it. The Session protocol detects this behavior due to the control message emitted

alongside the data message. Due to the broadcast transmission, the control message arrives at the PE and not the data packet. This behavior raises the Missing Packet warning (W1).

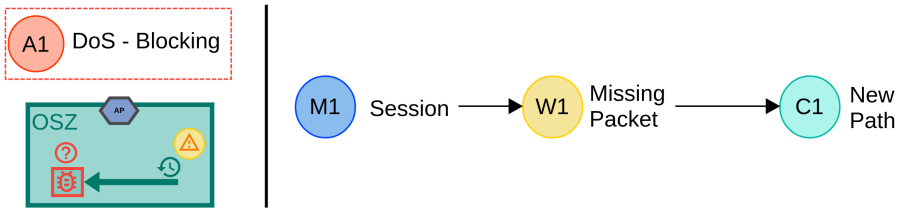



Figure 6.7 – DoS blocking scenario inside an OSZ.

To avoid this malicious entity, the system triggers a New Path C1 countermeasure to build a new route circumventing the suspicious router. The costs related to this countermeasure are the same presented on section 6.3.

DoS-Blocking in the Gray Area

Figure 6.8 shows a DoS blocking attack occurring during an IO transaction in the gray area. In this case, the Master-slave protocol detects the attack since the secure application never receives the answer from the packet sent to the IO device.

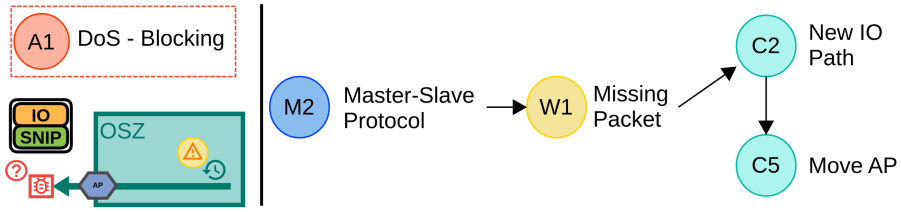



Figure 6.8 – DoS blocking scenario in the Gray Area.

The system initiates a countermeasure if an IO packet fails to receive an answer within a predetermined time threshold. The first countermeasure is to create a New IO Path (C2), corresponding to a new path to the IO device, traversing the gray area. Figure 6.9 shows the time to build a new IO path for different hop counts. This path construction is faster than the New Path (C1) because its entirely calculated by the kernel, while C1 also uses the control-NoC to find a new path inside the SZ.

The system can also trigger a Move AP (C5) countermeasure if the IO communication is still blocked even with the execution of C2. Figure 6.10 shows the Move AP process. The MPE decides a new position for the AP and notifies the chosen PE (PE0) that it must now configure this new AP (AP PE0). The chosen PE then sends `NEW_AP_INFO` to all $App_{sec}$ tasks, informing the new AP address, and starts a key renewal. The other PEs (PEn) receive this message, finish their pending IO transactions, update information about

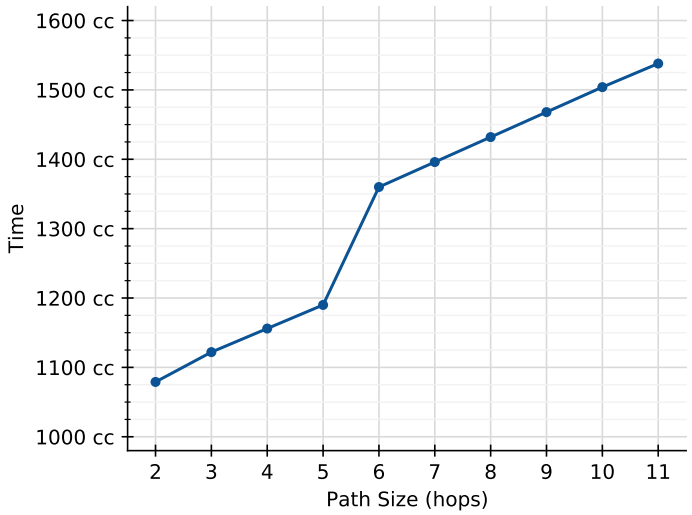



Figure 6.9 – Overhead for calculating a new path for peripherals.

the AP manager, and acknowledge the key renewal (`KEY_ACK`). Upon getting all the acknowledgments, the new AP manager sends `KEY_EVOLVE` and `REQ_SNIP_RENEW` to synchronize the keys and sets the registers of the new AP. When the previous AP manager (PE1) receives the `KEY_EVOLVE`, it closes the previous AP (AP PE1), clearing the memory-mapped registers (**CLEAR_PREV_AP**).

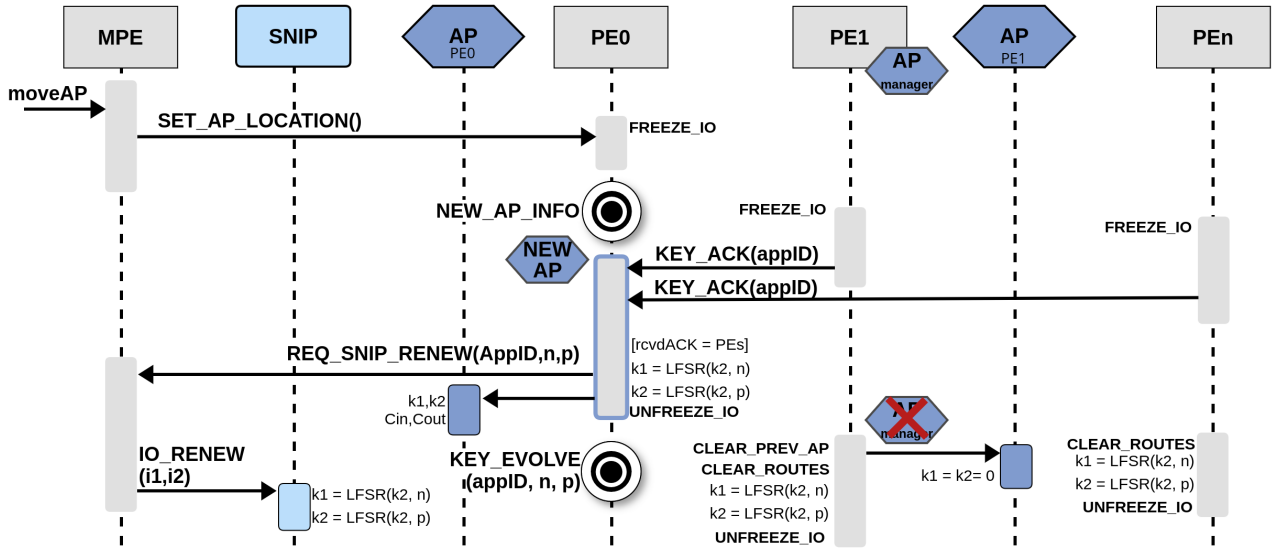



Figure 6.10 – Move AP sequence diagram.

Since the AP has changed, the current SR paths to IO are not valid because they consider the old position of the AP. For this reason, PEs and SNIPs need new IO paths. The MPE establishes a new SR path for the SNIP and sends through a `IO_CONFIG`. The PEs must recalculate route because the `KEY_EVOLVE` for new AP clears all IO paths on the PE (**CLEAR_ROUTES**).

Table 6.1 shows the costs of the Move AP countermeasure triggered on four applications: MPEG, DTW, Synthetic and MWD. The costs of Move AP can be analyzed as the cost to change the AP location and synchronize the new keys (summarized on Figure 6.10), plus the cost of the New IO Paths.

Table 6.1 – Average cost of Move AP countermeasure for different applications.

| App | Overhead (cc) | | |
|---|---|---|---|
| | Change AP | Path Config. | Total |
| MPEG | 5,754 | 3,132 | 8,887 |
| DTW | 5,634 | 3,118 | 8,753 |
| Synthetic | 7,467 | 3,182 | 10,047 |
| MWD | 11,457 | 3,433 | 14,890 |

The MPEG and DTW applications show similar overheads when changing the AP location as they have a small number of tasks (5 and 6, respectively). Conversely, the Synthetic application requires an additional 1,200 clock cycles due to its higher communication volume, while the MWD application requires around 5,800 additional clock cycles due to its larger task count (12). Both characteristics, IO communication volume and task number, affect the synchronization of the new AP location and key renewal. As all applications interact with two SNIPs, the costs of calculating a new IO path (column “Path Configuration”) remain similar, around 3,100 cc. This value refers to the cost of searching for the affected SNIPs, constructing the new IO paths (one for each SNIP - Figure 6.9), building the path, and sending the new path packets.

This countermeasure, which involves relocating the AP, is one of the most complex in terms of protocols and processing time. The worst-case scenario observed corresponds to 14,890 clock cycles, equivalent to 0.1489 ms@100 MHz. Besides its complexity, the proposed countermeasures add a negligible impact on applications’ performance due to its low computational overhead.

### 6.3.1 Final Remarks on Security Analysis and Costs

This section evaluated the framework against security threats such as Denial of Service, Spoofing, and Eavesdropping. The framework can help designers and developers to identify and mitigate security risks and threats, and to ensure the security and reliability of the system. The framework can also contribute to developing secure and trustworthy many-core systems for various applications.

Five attack scenarios were used to measure the costs of the countermeasures that the System Manager can apply to reinforce the system security. Table 6.2 illustrates the

costs of each countermeasure, considering the observed worst-case. The cost to create the OSZ (20.5K clock cycles in [Caimi et al., 2018b]) does not impact the application performance since it is executed before its execution starts. The table shows that the highest costs are related to renewing the keys and moving the AP location. The wost-case is 0.15 ms @ 100 MHz, demonstrating that the proposed countermeasures are lightweight, adding security to the applications.

Table 6.2 – Summary of the countermeasure costs, in clock cycles.

| Countermeasure | Worst-case (cc) |
|---|---|
| New OSZ path | 3,205 |
| New IO path | 1,538 |
| Key Renewal | 10,912 |
| Packet discarding | 378 |
| Move AP location | 14,890 |

The hardware to support the framework added an overhead of 48,8% in the communication infrastructure (NoC). This is an acceptable cost considering that the communication infrastructure is a part of the processing element (PE) and may lead to an actual area overhead at the PE level between 5% and 10% (Section 4.4).

These defense mechanisms outline a set of rules that strengthen the platform by reducing the vulnerabilities, consequently *preventing* and *mitigating* attacks. Even though some of these mechanisms apply an immediate countermeasure, such as an OSZ discarding a packet, system-level countermeasures can‘ *terminate* an attack, for example:

- **Task Migration**: Secure applications under attack, can be migrated to less susceptible regions of the MCSoC.
- **Abort Application**: if the MPE identifies that a given PE is the source of an attack, it identifies the tasks running in the PE, and thus the malicious application(s), sending a message to abort the potential malicious application(s).
- **Block PE**: if the previous countermeasure fails, the identified PE may contain, e.g., an HT, generating malicious traffic. Thus, the MPE sends a control message to activate the LC of the local port, isolating it from the rest of the system.
- **Change peripheral Port**: SNIPs may have a secondary channel connected to another router. When an attacker targets the SNIP, the primary channel can be swapped to the secondary one.

Collecting and recording information related to attacks is necessary to enable these systemic countermeasures. The next section evaluates the proposed framework with an intense attack campaign, showing that it correctly detects attacks, executes countermeasures and reports the occurences to the MPE.

## 6.4 Framework Evaluation

The attacks executed in the previous section were made with an individual simulation for each attack and with pre-defined static values. For example, the spoofing attack needed the AP key extracted from a previous simulation and directly inserted into the packet injector code. Consequently, only one spoofing attempt was possible due to the reactive key renewal. In the case of Hardware Trojans (HT), the router was selected randomly, and the blocking happened by forcing values in the simulation tool (Modelsim *force* command) at a predefined time chosen based on a previous simulation of the platform.

### 6.4.1 Attack Campaign

To perform complex attacks and distribute them into the system, we use three tools: dynamic HT emulation, configurable packet injector, and malicious application combined with packet injector.

#### Dynamic HT Emulation

This tool uses a script to read the traffic log of each router and count how many packets passed through them. Whenever this number reaches a threshold value, all router transmission signals (*tx*) are forced to logical '0' for 500 $\mu$s. After that, the counter starts again, from zero until it reaches the threshold again and repeats the process.

The router affected by this tool receives packets, its switch control module decides the output port and then directs the packet to the next router. If the *tx* signal is inactive due to the HT, the packet is dropped.

Any router can be selected to receive this HT. The user needs to configure the following parameters:

- *pe*: which routers to monitor and insert the faults. This parameter may receive the PEs coordinates in [x,y][1] or *random*;
- *infection_rate*: the percentage of routers to monitor and insert the faults in case of *random* in the *pe* above parameter;
- *time_step*: simulation step (in $\mu$s) to check if there are faults to input;
- *scan_time*: time between scans of the traffic file (in seconds);
- *trigger_value*: number of packets to block the router, the threshold value.

Figure 6.11 shows three example the dynamic HT configuration on a 5x5 MCSoC with the gray area being the first row and the first column: (a) with 5 PEs defined by the user; (b) random PEs at 25% infection rate, in this case 6 out of 25 routers are infected; (c) similar to previous, but with 50% infection rate: 12 out of 25 routers infected.

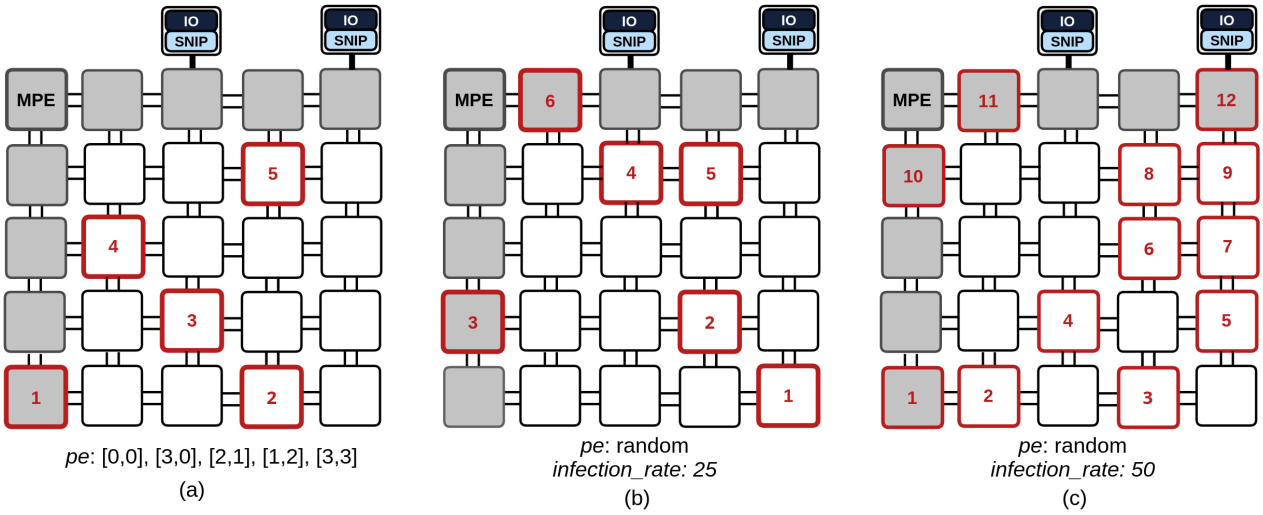



Figure 6.11 – Example of the dynamic HT configuration on a 5x5 MCSoC.

Configurable packet injector

The configurable packet injector is a malicious peripheral connected to any side of the MCSoC, without the SNIP. This injector can perform DoS attacks by sending packets in a high rate to the same target. The packets constructed the injector have the standard data packet size of the platform: 26 flits. From those flits, the user can configure the following fields:

- `HEADER_HI`: The first flit of the packet, holds the authentication values analyzed in the AP. To enter the OSZ, this value must be configured with the correct key.
- `HEADER_LO`: Second flit of the packet, holds the target address of the packet in XY coordinates.
- `SERVICE`: Defines the service of the packet. To perform attacks, usually has IO_ACK or IO_DELIVERY services.
- `F1_FLIT`: First authentication flit - *f1* from the authentication protocol.
- `F2_FLIT`: Second authentication flit - *f2*. Combined with f1 must have the correct value to be approved by the OS.

[1]The PEs are identified by its coordinate on the x-axis and y-axis as in a Cartesian plane with the origin in the bottom-left corner PE

- `SOURCE`: The address of the source of the packet by default, used by the SNIP to define the response packet's target.

Besides those specific fields of the packet, the user can also configure the simulation time to start and finish the transmission (*T_START* and *T_END*, respectively), and time between each packet (*PERIOD*).

Malicious task combined with packet injector

To perform spoofing attacks, it is essential to have the values of *k1* and *k2* to bypass the AP and the OS. So, we implemented applications with malicious tasks that leak *k1* and *k2* values every time a key renewal is performed. These values are sent to a malicious peripheral via the data-NoC inside a packet. The packet injector, therefore, can use this value on `HEADER_HI` to bypass the AP and in `F1_FLIT` and `F2_FLIT` to the packet be accepted in the OS.

One more vulnerability exposed by this scenario is the opening of the AP. Since the malicious task is inside the OSZ, one IO packet leaves the OSZ through an AP and increases the $C_{out}$ counter, leaving the AP opened to receive one packet.

Note that this behavior was done exclusively for this attack, as the tasks do not have access to *k1* and *k2*. To execute the spoofing attack without this mechanism, it would be necessary to obtain these keys by trial and error, which is impractical by simulation at the RTL level; or to have an HT capable of capturing the flits in the correct position of the packets, which is also not feasible due to the complexity of this task (snoop the packets, evaluate its contents, and transmit it to the malicious peripheral).

### 6.4.2 Experimental Setup

To show the effectiveness of the security mechanisms developed throughout this Thesis, we simulated the scenario depicted on Figure 6.12 with the following specifications:

- Size: 8x8;
- Gray Area: two rows (6 and 7) and two columns (3 and 4);
- Two peripherals connected on routers (2,7) and (5,7);
- Two malicious peripherals connected on routers (4,7) and (6,7);
- MPE running in the top-right PE, with the application injector connected directly to it;
- Six applications to run in secure mode. Four $App_{sec}$ communicate with IO devices (those with AP).

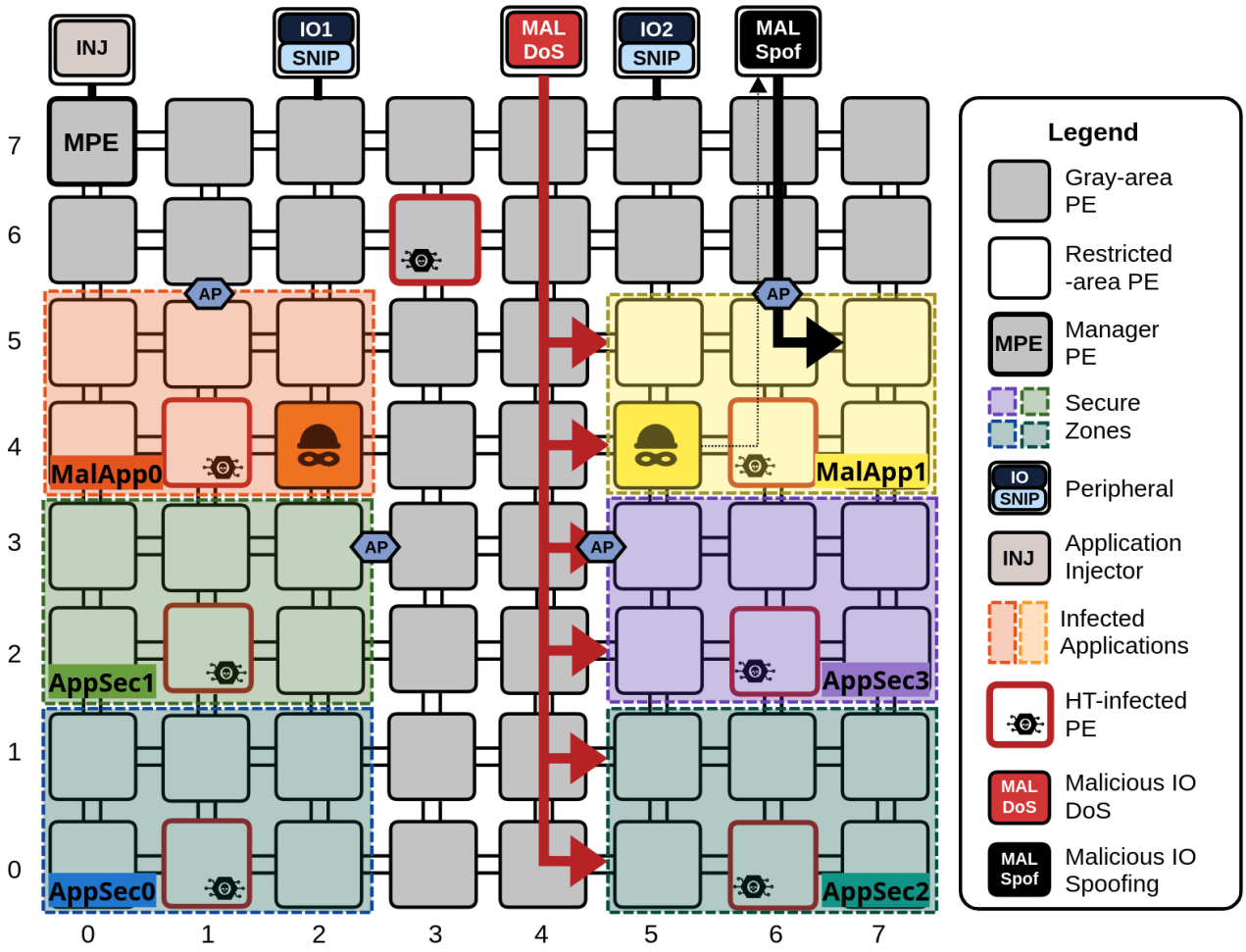



Figure 6.12 – Experimental setup to evaluate the security framework.

We performed the attack campaign to this scenario using the following threats:

- **HT** that drop packets.

  One inside each Secure Zone: (1,0), (1,2), (1,4), (6,0), (6,2), (6,4).

  One in the Gray-Area: (3,6).

- **MALDoS on (4,7)**: malicious IO device injecting packets at high rate to flood the NoC.

  Packets target PEs that are inside the three OSZ on the right, selected randomly.

  `PERIOD` of the injector is 2,000 clock cycles.

  The other injector parameters are not used.

- **MALSpoof on (6,7)**: malicious IO device that receives keys and tries to access the OSZ.

  Receives the keys from a malicious task running on PE (5,4).

  Tries to send packets to task (7,5).

  Packets contain the correct keyd on the first flit to enter the AP.

The `F1_FLIT` also contain the correct key.

The `F2_FLIT` is blank, which is detected as an attack by the OS.

- **MalApp0**: malicious application that tries to send packets to PEs outside the OSZ.

  On each iteration of the application, the task in PE (2,4) sends packets to PE (0,0)

- **MalApp1**: malicious application with a task that leaks key to IO device (MalSpoof).

  Every two key renewals, task in PE (5,4) send a packet leaking the keys to the IO in (6,7)

### 6.4.3 Results

This section presents two analyses using the scenario presented above, starting with the warnings reported to the MPE during the attack campaign and then discussing the execution time overhead.

Table 6.3 presents the number of warnings of each type registered throughout the scenario execution. In addition, the third column presents the detection mechanism, including the number of warnings emitted by each. The fourth column shows the countermeasure applied by the system.

Table 6.3 – Warning reported on attack scenario

| Warning | Occurence | Detection | Action |
|---|---|---|---|
| Missing Packet (W1) | 95 | Session (75)<br>Master-Slave (20) | New Path (C1)<br>Resend Packet |
| Unexpected Data (W2) | 4 | Access Point (2)<br>Master-Slave (2) | Discard Packet (C4)<br>Key Renew (C3) |
| Wrong Key (W4) | 101 | Access Point (101) | Discard Packet (C4) |
| Suspicious Route (W5) | 19 | Link Control (19) | Discard Packet (C4) |
| Access Attempt (W6) | 248 | Link Control (248) | Discard Packet (C4) |
| **TOTAL** | **467** | | |

Missing Packet (W1) happened 95 times. The HTs inside the OSZ dropped 75 packets, which were recovered using the Session Manager. The HT in the GA dropped the other 20 packets, which were detected by the Master-Slave protocol of the IO communication and requested retransmission with a new path (C1).

The MPE registered four cases of Unexpected Data (W2): two from the AP and one from a PE. These four packets are from the Spoofing (MALSpoof) attack, in which the injector received the keys and tried to enter the OSZ. Four of the packets sent by this injector

had the correct keys, but two of them were blocked by the AP counters ($C_{in} = C_{out}$), and the other two passed through the AP and reached the PE that was not expecting IO data. Since, in these four cases, the attack packets had the correct keys, OSZ triggered a key renewal (C3).

Wrong Type (W3) does not appear in this scenario because the MALSpoof injector is configured to send all packets with the correct type. However, if the packets arrived at the AP with the wrong key, the AP would trigger the same countermeasures as the Unexpected Data (W2) explained above.

The forged packets from the MALSpoof injector that tried to enter the OSZ but had the wrong key triggered Wrong Key (W4) at the AP. The action is only packet discarding (C3) since the key does not match, meaning that the MALSpoof has not discovered them.

Suspicious Route (W5) are warnings from the Link Controls reporting that packets are trying to leave the OSZ. These are the packets sent by the malicious task on PE (2,4), from MalApp0.

Access Attempt (W6) shows 248 attempts to enter an OSZ. These are packets from the injector MALDoS that was sending packets to random PEs inside the three OSZ on the right. All of them were immediately discarded (C3).

> The attack campaign induced a total of 467 attacks, and our platform managed to recover the missing data and reinforced the protection of secure applications.

To analyze the impact of such attacks and their respective countermeasures in terms of execution time, Table 6.4 shows the execution time in terms of absolute values (second and third columns) and the relative overhead (fourth column). The baseline column corresponds to the execution without attacks. The malicious applications, when simulated in the baseline scenario, have their malicious behavior deactivated.

Table 6.4 – Time overhead for Framework scenario (in ms)

| Apps | Baseline | Attack Campaign | Overhead |
|---|---|---|---|
| AppSec0 | 9,74 | 11,23 | 15,26% |
| AppSec1 | 9,94 | 10,28 | 3,36% |
| AppSec2 | 9,78 | 11,21 | 14,68% |
| AppSec3 | 9,92 | 10,53 | 6,15% |
| MalApp0 | 9,93 | 10,05 | 1,15% |
| MalApp1 | 5,06 | 5,53 | 9,21% |
| **Scenario** | **13,69** | **14,30** | **4,47%** |

The highest overheads are on AppSec0 and AppSec2. As these applications do not communicate with IO devices, a communication-intensive task was probably mapped to the PE with an HT-infected router, and this caused more packets to be lost. Applications 1 and 3 presented a low overhead on the execution time: 3,36% and 6,15%, respectively. The reason might be the HT activation time that dropped different packets and affected these applications differently.

MalApp0 presents the lowest overhead of 1,15%, probably because the top AP was favorably mapped to avoid the HT in the GA. MalApp1 is the one that had their keys leaked by a task and attacked by MALSpoof, which triggered extra key renewals that affected the execution time, reaching 9,21% overhead.

The overall execution time for the scenario with attacks reveals that the implemented countermeasures, designed to protect the system against an extensive attack, resulted in an overhead of 4.47%. This overhead includes the countermeasures execution and the time required for reporting and recording the attacks. This data emphasizes the efficiency of the proposed countermeasures, demonstrating that the cost of securing applications in terms of execution time is remarkably low.

## 6.5 Final Remarks

The seven defense mechanisms included in the framework, complemented by monitoring and countermeasures, establish a comprehensive set of rules and procedures that protect the platform. They do so by diminishing vulnerabilities and facilitating recovery from attacks. While some mechanisms initiate immediate countermeasures, such as an OSZ discarding a packet, systemic countermeasures are absent. These systemic countermeasures could, for instance, identify and terminate an attack by aborting an application injecting malicious packets. Enhancing the recovery protocol of the Session proposal with information about the location of HTs could be beneficial. Furthermore, SeMAP might consider access attempts on OSZs to place the AP.

Consequently, this framework paves the way for developing system-level heuristics to increase security in many-core systems. Integrating monitoring data through a Security Manager, as illustrated in Table 6.3, would enable the detection of more complex attacks and facilitate a broader range of countermeasures than those currently proposed.

The attack campaign revealed limitations in the platform that restricted the scope of attacks. First, the NIs and SNIPs are not able to handle cropped packets. For instance, when an HT is activated, forcing the *tx* signal to 0 in the middle of packet transmission results in a cropped packet and the loss of the EoP signal. The NIs and SNIPs depend on receiving the EoP to complete packet reading, failing to function correctly after receiving a cropped

packet. A potential solution is to verify both the EoP and packet size at the end of reception. If there is a discrepancy, the packet should be discarded.

Another issue found during the attack campaign is the absence of recovery mechanisms for the Security Manager functions (MPE) and the application injector. The MPE transmits SNIP configuration packets via the data-NoC, and the application injector dispatches task source code to the PEs also through the data-NoC. If HTs intercept any of these packets, the MPE cannot currently detect and recover these messages. Implementing timeout mechanisms for resending packets without an acknowledgment message could address this issue.

# 7. CONCLUSION

In this Thesis, we introduced the following statement: *It is feasible to integrate a comprehensive suite of methods to secure application execution in MCSoCs, with low impact on execution time and area footprint, through a framework that monitors and detects suspicious behavior and applies countermeasures to reinforce security. The focus is to protect simultaneously the computation and communication resources of sensitive applications, including access to IO devices.*

Below, we evaluate the statement in parts, demonstrating that the strategic and specific objectives were achieved.

*"It is feasible to integrate a comprehensive suite of methods to secure application execution in MCSoCs..."* – demonstrated by integrating SeMAP, Session Manager, and Opaque Secure Zones into a framework that systemically manages security.

The IO transactions, treated by SeMAP, are protected using a lightweight authentication protocol based on LFSR and XOR gates instead of cryptography and complex key exchange mechanisms. Inner OSZ communication, treated by the Session Manager, is monitored using a control NoC disjointed from the data NoC to detect attacks from HT-infected routers and recover the communication. Note that cryptography is not mentioned in the set of defense mechanisms. The inter-OSZ does not require cryptography due to the spatial isolation of the application. The exposed flows between the AP to the SNIP may be encrypted with lightweight cryptography, but this is out of the scope of this Thesis.

*"..with low impact on execution time and area footprint..."* - refer to the costs of each proposal. Regarding execution time, the SeMAP showed 0.5% overhead to create and map an application into the system, compared to the baseline OSZ method. The authentication protocol presented 2.56% increase with frequent key renewal protocols. The Session Manager execution time overhead varies depending on the application profile, being low for pipeline applications (3.55%) and 33.81% in the worst case. The defense mechanisms can have impacts on system startup, to insert MAC on the task source code, or on application allocation, to decide the placement and activate the secure zones. Even though such impacts may delay the application start, they do not effect the total application execution time.

Regarding hardware overhead, Table 4.2 compared the baseline router area (17,973 $\mu m^2$) to the new communication infrastructure (control and secure router – 26,766 $\mu m^2$), the area overhead corresponded to 48.8%. Note that the control NoC is an important element of all the contributions presented in this Thesis. Considering that the communication infrastructure represents no more than 20% of the PE area, the increase in the PE area, in the worst case, stays between 5% and 10%.

*"..through a framework that monitors and detects suspicious behavior and applies countermeasures to reinforce security.'* - we presented systemic and integrated security mechanisms that simultaneously monitor, detect and mitigate a broad spectrum of threats in real time. The proposed framework monitors the system at several locations, allowing the detection of threats. Threat detection fires countermeasures and generates security warnings to a Security Manager.

*"The focus is to protect simultaneously the computation and communication resources of sensitive applications, including access to IO devices.'* - was confirmed by the last attack campaign performed on the 8x8 scenario. There, threats to computation are represented by Spoofing and DoS attacks that attempt to deliver forged packets to a protected application. Threats to communication are the Hardware Trojans, which dropped packets even inside the OSZs. HTs also threatened the IO access in the Gray area and the malicious peripherals trying to access PEs inside the OSZ. The Security Manager identified 467 attacks, and the platform managed to recover missing data and reinforced the protection of the applications, which all of them executed correctly, with only 4.47% execution time overhead (average value).

In conclusion, this Thesis fulfilled all the proposed objectives and significantly advanced MCSoC security by combining several defenses instead of approaching them individually. Furthermore, this Thesis opened several research opportunities by structuring a registry of warnings inside a Security Manager, which can be processed to make even more effective countermeasures, extend the system's protection, and include more attack types and sources in the threat model.

## 7.1 Future Work

As a guideline for future work, this Thesis has room for improvement as follows:

1. Systemic Countermeasures. The next step of this platform is to utilize the security reports generated by the framework to apply more complex countermeasures, such as migrating tasks away from a detected HT, terminating tasks that are trying to leak information, blocking routers that are impairing the packet transmission, or even use a secondary port for access to IO devices.

2. HT localization. The Session Manager does not rely on the HT locating to be effective. However, finding the infected routers can help with new path creation and OSZ allocation since both can exclude the infected router. A technique to locate HT can use packet probing and establishing a route of trust based on the arriving packets. Working in this direction would fix the issue on the **SEARCHPATH**, which is only able to find

one alternative route, enhancing the path construction algorithm to include information about the NoC to avoid specific routers or ports.

3. OSZ defragmentation. The utilization of Secure Zones brings fragmentation in the long term. Fragmentation refers to non-contiguous regions with PEs not executing tasks, negatively affecting resource utilization. As SeMAP enforces restrictions to the placement of OSZs, it is expected to have fragmentation in the restricted areas during extensive simulations, limiting the deployment of applications with security constraints. Fragmentation analysis on scenarios with dynamic workloads, varying the number of applications and tasks per application, is a research opportunity, followed by a defragmentation technique.

4. Protect the MPE transactions. During the final attack campaign, we observed a limitation regarding the attacks on MPE and AppInj packets. If an HT drops a packet coming from the MPE or from the AppInj, there are no mechanisms to detect the missing packet and, consequently, not be able to recover them. Creating an ack-timeout protocol that can identify the missing packet and ask for retransmission is a way to solve this limitation.

5. Incomplete Packets. Also during the final attack campaign, the DMNI and SNIP showed a limitation on receiving incomplete packets. These packets were cropped due to the HT activation in the middle of their transmission. Because of that, part of the packet would arrive at the destination and the targets were not able to process it. Adding an automatic discard mechanism in the DMNI that ignores packets which packet size and the end-of-packet signal do not match is a possible solution.

6. Formalization of security and threat concepts. To reinforce the theoretical basis of the concepts adopted in the security evaluations and the threat models, a further development of the concepts would help delimit the extent of the countermeasures and even categorize the level of threats.

# REFERENCES

[Aghaei et al., 2020] Aghaei, B., Reshadi, M., Masdari, M., Sajadi, S., Hosseinzadeh, M., and Darwesh, A. (2020). Network adapter architectures in network on chip: comprehensive literature review. *Cluster Computing*, 23(1):321–346. https://doi.org/10.1007/s10586-019-02924-2.

[Ahmed et al., 2021] Ahmed, M. M., Dhavlle, A., Mansoor, N., Dinakarrao, S. M. P., Basu, K., and Ganguly, A. (2021). What Can a Remote Access Hardware Trojan do to a Network-on-Chip? In *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. https://doi.org/10.1109/iscas51556.2021.9401297.

[Ali and Khan, 2021] Ali, U. and Khan, O. (2021). Connoc: A practical timing channel attack on network-on-chip hardware in a multicore processor. In *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 192–202. https://doi.org/10.1109/HOST49136.2021.9702280.

[ARM, 2013] ARM (2013). AMBA® AXITM and ACETM Protocol Specification. https://developer.arm.com/documentation/ihi0022/e/, January 2024.

[Avizienis et al., 2004] Avizienis, A., Laprie, J. C., Randell, B., and Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33. https://doi.org/10.1109/tdsc.2004.2.

[Azad et al., 2019] Azad, S. P., Jervan, G., Tempelmeier, M., and Sepúlveda, J. (2019). CAESAR-MPSoC: Dynamic and Efficient MPSoC Security Zones. In *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 477–482. https://doi.org/10.1109/isvlsi.2019.00092.

[Azad et al., 2018] Azad, S. P., Niazmand, B., Jervan, G., and Sepúlveda, J. (2018). Enabling Secure MPSoC Dynamic Operation through Protected Communication. In *IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*, pages 481–484. https://doi.org/10.1109/icecs.2018.8617940.

[Baron et al., 2013] Baron, S., Wangham, M. S., and Zeferino, C. A. (2013). Security mechanisms to improve the availability of a Network-on-Chip. In *IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*, pages 609–612. https://doi.org/10.1109/icecs.2013.6815488.

[Benhani et al., 2019] Benhani, E. M., López, C. M., and Bossuet, L. (2019). Secure Internal Communication of a Trustzone-Enabled HeterogeneousSoc Lightweight Encryption. In *International Conference on Field-Programmable Technology (FPT)*, pages 239–242. https://doi.org/10.1109/icfpt47387.2019.00037.

[Benini and Micheli, 2002] Benini, L. and Micheli, G. (2002). Networks on chips: a new SoC paradigm. *IEEE Computer*, 35:70–78. https://doi.org/10.1109/2.976921.

[Bisht and Das, 2022] Bisht, B. and Das, S. (2022). BHT-NoC: Blaming Hardware Trojans in NoC Routers. *IEEE Design & Test*, 39(6):39–47. https://doi.org/10.1109/MDAT.2022.3202998.

[Bohnenstiehl et al., 2016] Bohnenstiehl, B., Stillmaker, A., Pimentel, J., Andreas, T., Liu, B., Tran, A., Adeagbo, E., and Bass, B. (2016). A 5.8 pJ/Op 115 billion ops/sec, to 1.78 trillion ops/sec 32nm 1000-processor array. In *IEEE Symposium on VLSI Circuits (VLSIC)*, pages 1–2. https://doi.org/10.1109/vlsic.2016.7573511.

[Boraten and Kodi, 2016] Boraten, T. and Kodi, A. K. (2016). Mitigation of Denial of Service Attack with Hardware Trojans in NoC Architectures. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1091–1100. https://doi.org/10.1109/ipdps.2016.59.

[Caimi, 2019] Caimi, L. L. (2019). *Secure Admission and Execution of Applications in NoC-based Many-cores Systems*. PhD thesis, PPGCC-PUCRS. 121p.

[Caimi et al., 2021] Caimi, L. L., Faccenda, R., and Moraes, F. G. (2021). A Survey on Security Mechanisms for NoC-based Many-Core SoCs. *Journal of Integrated Circuits and Systems*, 16(2):1–15. https://doi.org/10.29292/jics.v16i2.485.

[Caimi et al., 2018a] Caimi, L. L., Fochi, V., and Moraes, F. G. (2018a). Secure Admission of Applications in Many-Cores. In *IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pages 761–764. https://doi.org/10.1109/icecs.2018.8618021.

[Caimi et al., 2018b] Caimi, L. L., Fochi, V., Wachter, E., and Moraes, F. G. (2018b). Runtime creation of continuous secure zones in many-core systems for secure applications. In *Latin American Symposium on Circuits and Systems (LASCAS)*, pages 1–4. https://doi.org/10.1109/lascas.2018.8399904.

[Caimi and Moraes, 2019] Caimi, L. L. and Moraes, F. (2019). Security in Many-Core SoCs Leveraged by Opaque Secure Zones. In *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 471–476. https://doi.org/10.1109/isvlsi.2019.00091.

[Carara et al., 2009] Carara, E. A., de Oliveira, R. P., Calazans, N. L. V., and Moraes, F. G. (2009). HeMPS - a framework for NoC-based MPSoC generation. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1345–1348. https://doi.org/10.1109/iscas.2009.5118013.

[Charles et al., 2022] Charles, S., Bindschaedler, V., and Mishra, P. (2022). Digital Watermarking for Detecting Malicious Intellectual Property Cores in NoC Architectures.

*IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 30(7):952–965. https://doi.org/10.1109/TVLSI.2022.3167606.

[Charles et al., 2020a] Charles, S., Logan, M., and Mishra, P. (2020a). Lightweight Anonymous Routing in NoC based SoCs. In *Design, Automation Test in Europe Conference (DATE)*, pages 334–337. https://doi.org/10.23919/date48585.2020.9116572.

[Charles et al., 2020b] Charles, S., Lyu, Y., and Mishra, P. (2020b). Real-Time Detection and Localization of Distributed DoS Attacks in NoC-Based SoCs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(12):4510–4523. https://doi.org/10.23919/date.2019.8715009.

[Charles and Mishra, 2020a] Charles, S. and Mishra, P. (2020a). Lightweight and Trust-Aware Routing in NoC-Based SoCs. In *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 160–167. https://doi.org/10.1109/isvlsi49217.2020.00038.

[Charles and Mishra, 2020b] Charles, S. and Mishra, P. (2020b). Securing Network-on-Chip Using Incremental Cryptography. In *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 168–175. https://doi.org/10.1109/isvlsi49217.2020.00039.

[Charles and Mishra, 2022] Charles, S. and Mishra, P. (2022). A Survey of Network-on-Chip Security Attacks and Countermeasures. *ACM Computing Surveys*, 54(5):101:1–101:36. https://doi.org/10.1145/3450964.

[Chaves et al., 2019] Chaves, C., Azad, S., Hollstein, T., and Sepúlveda, J. (2019). DoS attack detection and path collision localization in NoC-based MpsoC architectures. *Journal of Low Power Electronics and Applications*, 9. https://doi.org/10.3390/jlpea9010007.

[Comarú et al., 2023] Comarú, G., Faccenda, R. F., Caimi, L. L., and Moraes, F. G. (2023). Secure network interface for protecting IO communication in many-cores. In *Symposium on Integrated Circuits and Systems Design (SBCCI)*, pages 1–6. IEEE. https://doi.org/10.1109/SBCCI60457.2023.10261655.

[Costa et al., 2021] Costa, W. N., Lima, L. P., and de Lima Junior, O. A. (2021). Extracting Packet Dependence from NoC Simulation Traces Using Association Rule Mining. *Analog Integrated Circuits and Signal Processing*, 106(1):235–247. https://doi.org/10.1109/sbcci.2018.8533244.

[Daoud and Rafla, 2019a] Daoud, L. and Rafla, N. (2019a). Analysis of Black Hole Router Attack in Network-on-Chip. In *IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 69–72. https://doi.org/10.1109/mwscas.2019.8884979.

[Daoud and Rafla, 2019b] Daoud, L. and Rafla, N. (2019b). Detection and prevention protocol for black hole attack in Network-on-Chip. In *IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*, pages 22:1–22:2. https://doi.org/10.1145/3313231.3352374.

[Diffie and Hellman, 1976] Diffie, W. and Hellman, M. (1976). New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654. https://doi.org/10.1145/3549993.3550007.

[Dinechin et al., 2014] Dinechin, B. D. D., Amstel, D. V., and Lager, G. (2014). Time-critical computing on a single-chip massively parallel processor. In *Design, Automation Test in Europe Conference (DATE)*, pages 1–6. https://doi.org/10.7873/date.2014.110.

[Elkanishy et al., 2019] Elkanishy, A., Badawy, A. A., Furth, P. M., Boucheron, L. E., and Michael, C. P. (2019). Supervising Communication SoC for Secure Operation Using Machine Learning. In *IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 582–585. https://doi.org/10.1109/mwscas.2019.8885273.

[Ellinidou et al., 2019] Ellinidou, S., Sharma, G., Rigas, T., Vanspouwen, T., Markowitch, O., and Dricot, J. (2019). SSPSoC: A Secure SDN-Based Protocol over MPSoC. *Security and Communication Networks*, 2019:4869167:1–4869167:11. https://doi.org/10.1155/2019/4869167.

[Faccenda et al., 2021] Faccenda, R. F., Caimi, L. L., and Moraes, F. G. (2021). Detection and Countermeasures of Security Attacks and Faults on NoC-Based Many-Cores. *IEEE Access*, 9:153142–153152. https://doi.org/10.1109/access.2021.3127468.

[Fernandes et al., 2016] Fernandes, R., Marcon, C., Cataldo, R., Silveira, J., Sigl, G., and Sepúlveda, J. (2016). security aware routing approach for NoC-based MPSoCs. In *Symposium on Integrated Circuits and Systems Design (SBCCI)*, pages 1–6. https://doi.org/10.1109/SBCCI.2016.7724054.

[Fiorin et al., 2008] Fiorin, L., Lukovic, S., and Palermo, G. (2008). Implementation of a Reconfigurable Data protection Module for NoC-based MPSoCs. In *IEEE International Parallel and Distributed Processing Symposium*, pages 1–8. https://doi.org/10.1109/IPDPS.2008.4536514.

[Fiorin et al., 2013] Fiorin, L., Palermo, G., and Silvano, C. (2013). A configurable monitoring infrastructure for NoC-based architectures. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 22(11):2438–2442. https://doi.org/10.1109/TVLSI.2013.2290102.

[Fiorin et al., 2007] Fiorin, L., Silvano, C., and Sami, M. (2007). Security Aspects in Networks-on-Chips: Overview and Proposals for Secure Implementations. In *Euromicro Conference on Digital System Design Architectures, Methods and Tools (DSD)*, pages 539–542. https://doi.org/10.1109/dsd.2007.4341520.

[Forlin et al., 2019] Forlin, B., Reinbrecht, C., and Sepúlveda, J. (2019). Security Aspects of Real-Time MPSoCs: The Flaws and Opportunitiesof Preemptive NoCs. In *IEEE In-*

*ternational Conference on Very Large Scale Integration (VLSI-SoC)*, pages 209–233. https://doi.org/10.1007/978-3-030-53273-4_10.

[Ge et al., 2019] Ge, J., Gao, N., Tu, C., Xiang, J., and Liu, Z. (2019). AdapTimer: Hardware/Software Collaborative Timer Resistant to Flush-BasedCache Attacks on ARM-FPGA Embedded SoC. In *IEEE International Conference on Computer Design (ICCD)*, pages 585–593. https://doi.org/10.1109/iccd46524.2019.00085.

[Gondal et al., 2020] Gondal, H., Fayyaz, S., Aftab, A., Nokhaiz, S., Arshad, M., and Saleem, W. (2020). A method to detect and avoid hardware trojan for network-on-chip architecture based on error correction code and junction router (ECCJR). *International Journal of Advanced Computer Science and Applications*, 11(4):581–586. https://doi.org/10.14569/ijacsa.2020.0110476.

[Halder et al., 2023] Halder, D., Merugu, M., and Ray, S. (2023). Obnocs: Protecting network-on-chip fabrics against reverse-engineering attacks. *ACM Transactions on Embedded Computing Systems*, 22(5s). https://doi.org/10.1145/3609107.

[Harttung et al., 2019] Harttung, J., Franz, E., Moriam, S., and Walther, P. (2019). Lightweight Authenticated Encryption for Network-on-Chip Communications. In *ACM Great Lakes Symposium on VLSI*, pages 33–38. https://doi.org/10.1145/3299874.3317990.

[Hemani et al., 2000] Hemani, A., Jantsch, A., Kumar, S., Postula, A., Öberg, J., Millberg, M., and Lindqvist, D. (2000). Network on chip: An architecture for billion transistor era. In *Nordic Circuits and Systems Conference (NORCHIP)*, pages 166–173.

[Ho et al., 2019] Ho, W., Pammu, A. A., Ne, K. Z. L., Chong, K., and Gwee, B. (2019). Reconfigurable Routing Paths As Noise Generators Using NoC Platformfor Hardware Security Applications. In *IEEE International System-on-Chip Conference (SoCC)*, pages 86–91. https://doi.org/10.1109/socc46988.2019.1570557958.

[Hussain and Guo, 2019] Hussain, M. and Guo, H. (2019). A Bandwidth-Aware Authentication Scheme for Packet-Integrity Attack Detection on Trojan Infected NoC. In *IEEE/IFIP International Conference on VLSI and System-on-Chip, VLSI-SoC*, pages 201–206. https://doi.org/10.1109/vlsi-soc.2018.8645051.

[Hussain et al., 2018] Hussain, M., Malekpour, A., Guo, H., and Parameswaran, S. (2018). EETD: An Energy Efficient Design for Runtime Hardware Trojan Detection in Untrusted Network-on-Chip. In *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 345–350. https://doi.org/10.1109/isvlsi.2018.00070.

[Indrusiak et al., 2019] Indrusiak, L. S., Harbin, J., Reinbrecht, C., and Sepúlveda, J. (2019). Side-channel protected MPSoC through secure real-time networks-on-chip. *Microprocessors and Microsystems*, 68:34–46. https://doi.org/10.1016/j.micpro.2019.04.004.

[JYV et al., 2018] JYV, M. K., Swain, A. K., K, S. K., Sahoo, S. R., and Mahapatra, K. (2018). Run Time Mitigation of Performance Degradation Hardware Trojan Attacks in Network on Chip. In *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 738–743. https://doi.org/10.1109/isvlsi.2018.00139.

[Kashi et al., 2021] Kashi, S., Patooghy, A., Rahmati, D., and Fazeli, M. (2021). An Energy Efficient Synthesis Flow for Application Specific SoC Design. *Integration, the VLSI Journal*, 81:331–341. https://doi.org/10.1016/j.vlsi.2021.08.005.

[Kenarangi and Partin-Vaisband, 2019] Kenarangi, F. and Partin-Vaisband, I. (2019). Security Network On-Chip for Mitigating Side-Channel Attacks. In *ACM/IEEE International Workshop on System Level Interconnect Prediction (SLIP)*, pages 1–6. https://doi.org/10.1109/slip.2019.8771328.

[Kornaros et al., 2018] Kornaros, G., Tomoutzoglou, O., and Coppola, M. (2018). Hardware-Assisted Security in Electronic Control Units: Secure Automotive Communications by Utilizing One-Time-Programmable Network on Chip and Firewalls. *IEEE Micro*, 38(5):63–74. https://doi.org/10.1109/mm.2018.053631143.

[Kulkarni et al., 2016] Kulkarni, A., Pino, Y., and Mohsenin, T. (2016). SVM-based real-time hardware Trojan detection for many-core platform. In *International Symposium on Quality Electronic Design (ISQED)*, pages 362–367. https://doi.org/10.1109/isqed.2016.7479228.

[Kulkarni et al., 2021] Kulkarni, V. J., Rajan, M., Gupta, R., Jose, J., and Nandi, S. (2021). Packet header attack by hardware trojan in noc based TCMP and its impact analysis. In *IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*, pages 21–28. https://doi.org/10.1145/3479876.3481597.

[Kumar et al., 2021] Kumar, J. M., Swain, A. K., Mahapatra, K., et al. (2021). Fortified-noc: A robust approach for trojan-resilient network-on-chips to fortify multicore-based consumer electronics. *IEEE Transactions on Consumer Electronics*, 68(1):57–68. https://doi.org/10.1109/TCE.2021.3129155.

[Kumar et al., 2018] Kumar, S., Seth, S., Sahoo, S. R., Mahapatra, A., Swain, A. K., and Mahapatra, K. (2018). PUF-Based Secure Test Wrapper for SoC Testing. In *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 672–677. https://doi.org/10.1109/isvlsi.2018.00127.

[Li et al., 2016] Li, H., Liu, Q., and Zhang, J. (2016). A survey of hardware Trojan threat and defense. *Integration, the VLSI Journal*, 55:426–437. https://www.sciencedirect.com/science/article/pii/S0167926016000067.

[Linder and Harden, 1991] Linder, D. H. and Harden, J. C. (1991). An Adaptive and Fault Tolerant Wormhole Routing Strategy for k-ary n-cubes. *IEEE Transactions on Computer*, 40(1):2–12. https://doi.org/10.1109/12.67315.

[Manju et al., 2020] Manju, R., Das, A., Jose, J., and Mishra, P. (2020). SECTAR: Secure NoC using Trojan Aware Routing. In *IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*, pages 1–8. https://doi.org/10.1109/nocs50636.2020.9241711.

[Meng et al., 2023] Meng, X., Raj, K., Ray, S., and Basu, K. (2023). SeVNoC: Security Validation of System-on-Chip Designs With NoC Fabrics. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 42(2):672–682. https://doi.org/10.1109/TCAD.2022.3179307.

[Moraes et al., 2004] Moraes, F. G., Calazans, N., Mello, A., Möller, L., and Ost, L. (2004). HERMES: an infrastructure for low area overhead packet-switching networks on chip. *Integration, the VLSI Journal*, 38(1):69 – 93. https://doi.org/10.1016/j.vlsi.2004.03.003.

[Mountford et al., 2023] Mountford, T., Dhavlle, A., Tevebaugh, A., Mansoor, N., Dinakarrao, S. M. P., and Ganguly, A. (2023). Address Obfuscation to Protect against Hardware Trojans in Network-on-Chips. *Journal of Low Power Electronics and Applications*, 13(3):50. https://doi.org/10.3390/jlpea13030050.

[Oliveira et al., 2018] Oliveira, B., Reusch, R., Medina, H., and Moraes, F. G. (2018). Evaluating the Cost to Cipher the NoC Communication. In *Latin American Symposium on Circuits and Systems (LASCAS)*, pages 1–4. https://doi.org/10.1109/lascas.2018.8399914.

[Oracle, 2017] Oracle (2017). Oracle's SPARC T8 and SPARC M8 Server Architecture. Technical report, Oracle Corporation. 44p.

[Patooghy et al., 2023] Patooghy, A., Hasanzadeh, M., Sarihi, A., Abdelrehim, M., and Badawy, A. A. (2023). Securing Network-on-chips Against Fault-injection and Cryptoanalysis Attacks via Stochastic Anonymous Routing. *ACM Journal on Emerging Technologies in Computing Systems*, 19(3):22:1–22:21. https://doi.org/10.1145/3592798.

[Peckham, 2020] Peckham, O. (2020). Esperanto Unveils ML Chip with Nearly 1,100 RISC-V Cores. https://www.hpcwire.com/2020/12/08/esperanto-unveils-ml-chip-with-nearly-1100-risc-v-cores, March 2022.

[Philomina, 2021] Philomina, J. (2021). A Study on the Effect of Hardware Trojans in the Performance of Network on Chip Architectures. In *International Conference on Smart Computing and Communications (ICSCC)*, pages 314–318. https://doi.org/10.1109/icscc51209.2021.9528249.

[Popovici et al., 2010] Popovici, K., Rousseau, F., Jerraya, A. A., and Wolf, M. (2010). *Embedded Software Design and Programming of Multiprocessor System-on-Chip: Simulink and System C Case Studies*. Springer Publishing Company. 290p.

[Ramachandran, 2002] Ramachandran, J. (2002). *Designing Security Architecture Solutions*. John Wiley & Sons, Inc., 483p. 483p.

[Raparti and Pasricha, 2019] Raparti, V. Y. and Pasricha, S. (2019). Lightweight Mitigation of Hardware Trojan Attacks in NoC-based Manycore Computing. In *ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. https://doi.org/10.1145/3316781.3317851.

[Rauber and Rünger, 2013] Rauber, T. and Rünger, G. (2013). *Parallel Programming for Multicore and Cluster Systems*. Springer. 463p.

[Ravikumar et al., 2019] Ravikumar, C., Swamy, S., and Uma, B. (2019). A hierarchical approach to self-test, fault-tolerance and routing security in a Network-on-Chip. In *IEEE International Test Conference India (ITC India)*, pages 1–6. https://doi.org/10.1109/itcindia46717.2019.8979997.

[Real et al., 2018] Real, M. M., Wehner, P., Lapotre, V., Göhringer, D., and Gogniat, G. (2018). Application Deployment Strategies for Spatial Isolation on Many-Core Accelerators. *ACM Transaction on Embedded Computing Systems*, 17(2):55:1–55:31. https://doi.org/10.1145/3168383.

[Reinbrecht et al., 2020] Reinbrecht, C., Aljuffri, A., Hamdioui, S., Taouil, M., Forlin, B., and Sepúlveda, J. (2020). Guard-NoC: A Protection Against Side-Channel Attacks for MPSoCs. In *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 536–541. https://doi.org/10.1109/isvlsi49217.2020.000-1.

[Reinbrecht et al., 2019] Reinbrecht, C., Forlin, B., and Sepúlveda, J. (2019). Cache timing attacks on NoC-based MPSoCs. *Microprocessors and Microsystems*, 66:1–9. https://doi.org/10.1016/j.micpro.2019.01.007.

[Rout et al., 2020] Rout, S. S., Singh, A., Patil, S. B., Sinha, M., and Deb, S. (2020). Security Threats in Channel Access Mechanism of Wireless NoC and Efficient Countermeasures. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. https://doi.org/10.1109/iscas45731.2020.9180581.

[Rovinski et al., 2019] Rovinski, A. et al. (2019). Evaluating Celerity: A 16-nm 695 Giga-RISC-V Instructions/s Manycore Processor With Synthesizable PLL. *IEEE Solid-State Circuits Letters*, 2(12):289–292. https://doi.org/10.1109/lssc.2019.2953847.

[Ruaro et al., 2018] Ruaro, M., Caimi, L. L., Fochi, V., and Moraes, F. G. (2018). A Framework for Heterogeneous Many-core SoCs Generation. In *Latin American Symposium on Circuits and Systems (LASCAS)*, pages 89–92. https://doi.org/10.1109/lascas.2019.8667590.

[Ruaro et al., 2020] Ruaro, M., Caimi, L. L., and Moraes, F. G. (2020). A Systemic and Secure SDN Framework for NoC-Based Many-Cores. *IEEE Access*, 8:105997–106008. https://doi.org/10.1109/access.2020.3000457.

[Sepúlveda et al., 2018] Sepúlveda, J., Willgerodt, F., and Pehl, M. (2018). SEPUFSoC: Using PUFs for Memory Integrity and Authentication in Multi-Processors System-on-Chip. In *Great Lakes Symposium on VLSI (GLSVLSI)*, pages 39–44. https://doi.org/10.1145/3194554.3194562.

[Shakya et al., 2017] Shakya, B., He, T., Salmani, H., Forte, D., Bhunia, S., and Tehranipoor, M. (2017). Benchmarking of hardware trojans and maliciously affected circuits. *Journal of Hardware and Systems Security*, 1(1):85–102. https://doi.org/10.1007/s41635-017-0001-6.

[Sharma et al., 2021] Sharma, G., Bousdras, G., Ellinidou, S., Markowitch, O., Dricot, J.-M., and Milojevic, D. (2021). Exploring the security landscape: Noc-based mpsoc to cloud-of-chips. *Microprocessors and Microsystems*, 84:103963. https://doi.org/10.1016/j.micpro.2021.103963.

[Sharma et al., 2019] Sharma, G., Kuchta, V., Sahu, R. A., Ellinidou, S., Bala, S., Markowitch, O., and Dricot, J. (2019). A Twofold Group Key Agreement Protocol for NoC based MPSoCs. *Transactions on Emerging Telecommunications Technologies*, 30(6):1–18. https://doi.org/10.1109/pst.2018.8514117.

[Siddiqui et al., 2019] Siddiqui, A. S., Shirley, G., Joseph, S. R., Gui, Y., Plusquellic, J., van Dijk, M., and Saqib, F. (2019). Multilayer Camouflaged Secure Boot for SoCs. In *International Workshop on Microprocessor/SoC Test, Security and Verification (MTV)*, pages 56–61. https://doi.org/10.1109/mtv48867.2019.00019.

[Sodani et al., 2016] Sodani, A., Gramunt, R., Corbal, J., Kim, H. S., Vinod, K., Chinthamani, S., Hutsell, S., Agarwal, R., and Liu, Y. C. (2016). Knights Landing: Second-Generation Intel Xeon Phi Product. *IEEE Micro*, 36(2):34–46. https://doi.org/10.1109/mm.2016.25.

[Sudusinghe et al., 2022] Sudusinghe, C., Charles, S., Ahangama, S., and Mishra, P. (2022). Eavesdropping Attack Detection Using Machine Learning in Network-on-Chip Architectures. *IEEE Design & Test*, 39(6):28–38. https://doi.org/10.1109/MDAT.2022.3202995.

[Tecnhlogies, 2018] Tecnhlogies, M. (2018). TILE-Gx72 Processor Overview. http://www.mellanox.com/page/products_dyn?product_, March 2022.

[Tibaldi et al., 2021] Tibaldi, M., Pilato, C., and Ferrandi, F. (2021). Automatic Generation of Heterogeneous SoC Architectures With Secure Communications. *IEEE Embedded Systems Letters*, 13(2):61–64. https://doi.org/10.1109/les.2020.3003974.

[Tran et al., 2021] Tran, T.-K., Dang, T.-P., Bui, T.-T., and Huynh, H.-T. (2021). A reliable approach to secure iot systems using cryptosystems based on soc fpga platforms. In

*International Symposium on Electrical and Electronics Engineering (ISEE)*, pages 53–58. https://doi.org/10.1109/ISEE51682.2021.9418709.

[Wachter et al., 2017] Wachter, E., Caimi, L. L., Fochi, V., Munhoz, D., and Moraes, F. G. (2017). BrNoC: A broadcast NoC for control messages in many-core systems. *Microelectronics Journal*, 68:69 – 77. https://doi.org/10.1016/j.mejo.2017.08.010.

[Weber et al., 2020] Weber, I., Marchezan, G., Caimi, L., Marcon, C., and Moraes, F. G. (2020). Open-Source NoC-Based Many-Core for Evaluating Hardware Trojan Detection Methods. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. http://dx.doi.org/10.1109/ISCAS45731.2020.9180578.

[Wu et al., 2021] Wu, Q., Wang, X., and Chen, J. (2021). Defending against thermal covert channel attacks by task migration in many-core system. In *IEEE International Conference on Circuits and Systems (ICCS)*, pages 111–120. https://doi.org/10.1109/ICCS52645.2021.9697251.

[Xiao et al., 2020] Xiao, Y., Xin, J., and Shen, Y. (2020). CNN Based Electromagnetic Side Channel Attacks on SoC. *IOP Conference Series: Materials Science and Engineering*, 782(3):1–7. https://doi.org/10.1088/1757-899x/782/3/032055.

[Yao et al., 2023] Yao, J., Zhang, Y., Hua, Y., Li, Y., Yang, J., and Chen, X. (2023). Spotlight: An impairing packet transmission attack targeting specific node in noc-based TCMP. In *IEEE European Test Symposium (ETS)*, pages 1–4. https://doi.org/10.1109/ETS56758.2023.10174197.

[Zhang et al., 2018] Zhang, L., Wang, X., Jiang, Y., Yang, M., Mak, T. S. T., and Singh, A. K. (2018). Effectiveness of HT-assisted sinkhole and blackhole denial of service attacks targeting mesh networks-on-chip. *Journal of Systems Architecture*, 89:84–94. https://doi.org/10.1016/j.sysarc.2018.07.005.

[Zhang et al., 2022] Zhang, Y., Li, Y., Chen, X., Yang, J., Hua, Y., and Yao, J. (2022). Puf-based secure test wrapper design for network-on-chip. In *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 181–184. https://doi.org/10.1109/HOST54066.2022.9840115.

[Zhao et al., 2020] Zhao, Y., Wang, X., Jiang, Y., Wang, L., Yang, M., Singh, A. K., and Mak, T. S. T. (2020). On hardware-trojan-assisted power budgeting system attack targeting many core systems. *Journal of Systems Architecture*, 109:1–11. https://doi.org/10.1109/socc.2018.8618565.

# APPENDIX A – LIST OF PUBLICATIONS

## Journal Publications

A Comprehensive Framework for Systemic Security Management in NoC-based Many-cores
FACCENDA, Rafael; COMARÚ, Gustavo; CAIMI, Luciano; MORAES, Fernando Gehm
IEEE Access, vol. 11, pp 131836 - 131847, November 2023
https://doi.org/10.1109/ACCESS.2023.3336565

SeMAP - A Method to Secure the Communication in NoC-based Many Cores
FACCENDA, Rafael; COMARÚ, Gustavo; CAIMI, Luciano; Moraes, Fernando Gehm
IEEE Design & Test, vol. 40(5), pp 42-51, October 2023
https://dx.doi.org/10.1109/MDAT.2023.3277813

Detection and Countermeasures of Security Attacks and Faults on NoC-based Many-Cores
FACCENDA, Rafael; CAIMI, Luciano; MORAES, Fernando Gehm
IEEE Access, vol. 9, pp. 153142-153152, November 2021
https://doi.org/10.1109/ACCESS.2021.3127468

A Survey on Security Mechanisms for NoC-based Many-Core SoCs
CAIMI, Luciano; FACCENDA, Rafael; MORAES, Fernando Gehm
Journal of Integrated Circuits and Systems, vol. 16(2), pp. 1-15. November 2021
https://doi.org/10.29292/jics.v16i2.485

## Conference Publications

Lightweight Authentication for Secure IO Communication in NoC-based Many-cores
FACCENDA, Rafael; COMARÚ, Gustavo; CAIMI, Luciano; MORAES, Fernando Gehm
In: ISCAS, 2023
http://dx.doi.org/10.1109/ISCAS46773.2023.10181962

Secure Network Interface for Protecting IO Communication in Many-cores
COMARÚ, Gustavo; FACCENDA, Rafael; CAIMI, Luciano; MORAES, Fernando Gehm
In: SBCCI, 2023
http://dx.doi.org/10.1109/SBCCI60457.2023.10261655

Secure Communication with Peripherals in NoC-based Many-cores
FACCENDA, Rafael Follmann; COMARÚ, Gustavo; CAIMI, Luciano; MORAES, Fernando Gehm
In: SBCCI, 2022
http://dx.doi.org/10.1109/SBCCI55532.2022.9893244

# APPENDIX B – CTG OF THE APPLICATIONS

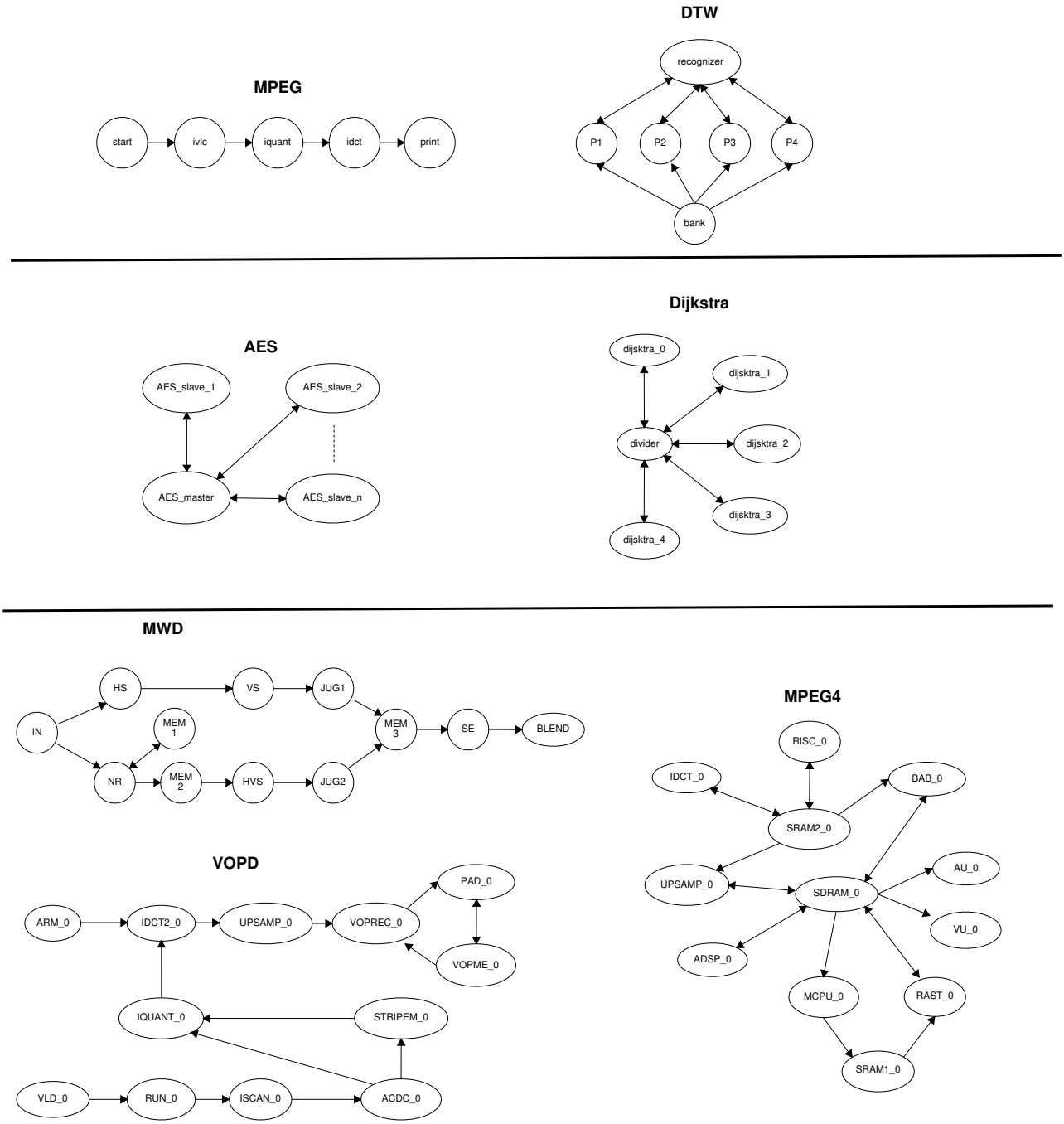



Figure B.1 – CTG of the used benchmarks.

# APPENDIX C – SEARCH STRING

```
(( TITLE ( "MPSoC" ) AND TITLE ( "security" ) )
OR ( TITLE ( "MPSoC" ) AND TITLE ( "secure" ) )
OR ( TITLE ( "MP-SoC" ) AND TITLE ( "secure" ) )
OR ( TITLE ( "MPSoC" ) AND TITLE ( "firewall" ) )
OR ( TITLE ( "MPSoC" ) AND ( "demilitarized" ) )
OR ( TITLE ( "MPSoC" ) AND ( "dos" ) )
OR ( TITLE ( "MPSoC" ) AND ( "cryptography" ) )
OR ( TITLE ( "Many-Core" ) AND TITLE ( "secure" ) )
OR ( TITLE ( "Many-Core" ) AND TITLE ( "attack" ) )
OR ( TITLE ( "Many-Core" ) AND TITLE ( "trojan" ) )
OR ( TITLE ( "Many-Core" ) AND ( "cryptography" ) )
OR ( TITLE ( "NoC" ) AND TITLE ( "security" ) )
OR ( TITLE ( "NoCs" ) AND TITLE ( "security" ) )
OR ( TITLE ( "network-on-chip" ) AND TITLE ( "security" ) )
OR ( TITLE ( "NoC" ) AND TITLE ( "trojan" ) )
OR ( TITLE ( "network-on-chip" ) AND TITLE ( "trojan" ) )
OR ( TITLE ( "NoC" ) AND TITLE ( "attack" ) )
OR ( TITLE ( "network-on-chip" ) AND TITLE ( "attack" ) )
OR ( TITLE ( "NoC" ) AND ( "cryptography" ) )
OR ( TITLE ( "network-on-chip" ) AND ( "cryptography" ) )
OR ( TITLE ( "SoC" ) AND TITLE ( "firewall" ) )
OR ( TITLE ( "SoC" ) AND TITLE ( "attack" ) )
OR ( TITLE ( "SoC" ) AND TITLE ( "secure" ) )
OR ( TITLE ( "SoC" ) AND TITLE ( "dos" ) )
OR ( TITLE ( "zone" ) AND TITLE ( "secure" ) ))
AND (SUBJAREA(COMP) OR SUBJAREA(ENGI) OR SUBJAREA(MATH))
AND (LANGUAGE(English))
AND (DOCTYPE(cp) OR DOCTYPE(ar) OR DOCTYPE(bk) OR DOCTYPE(ch) OR
DOCTYPE(ip))
AND (LIMIT-TO (PUBYEAR, 2020) OR LIMIT-TO (PUBYEAR, 2019) OR LIMIT-TO
(PUBYEAR, 2018))
```

# APPENDIX D – OSZ API DETAIL

Figure D.1 presents a sequence diagram, covering the $App_{sec}$ lifetime, from its admission into the system, up to its finishing. Yellow circles in the Figure (●) are functions of the kernel related to the task allocation. Blue circles (●) are the kernel functions moved to the OSZ API. The communication among **Injector**, **Manager**, and **PEs** is represented by arrows, being white arrows flows transferred through the data-NoC while the black arrows transferred through the control-NoC. Dotted black arrows illustrate a broadcast control message that reaches all PEs of the $App_{sec}$.

The OSZ protocol follows these main steps:

- The protocol starts with a request, from the **Injector**, for a new $App_{sec}$ to be executed. The **Manager** reads and registers the resources required by $App_{sec}$ (functions 1 and 2), creates and maps the OSZ (3 to 7).
- The **Manager** executes the task mapping (8), and sends the mapping result to the **Injector** (**APP_ALLOC_MAP**) and the **TASK_RELEASE** to the PEs, notifying the task that will be executed at each PE. The **Injector** sends the task object code to each PE. Once the allocation process finishes, the **PEs** send the **TASK_ALLOCATED** control message to the **Manager**.
- With all tasks mapped, the manager executes the OSZ Setting (9 to 11), and release the $App_{sec}$ to execute (**START_APP_SERVICE** control message).
- When the application finishes, the **Manager** executes the OSZ Unsetting (11 to 12), and releases the resources reserved for $App_{sec}$ (13).

The OSZ API functions are detailed below.

3 get_static_SZ()
Defines the OSZ when it is statically defined at design time. This is used mostly for debug purposes, to choose the location of an OSZ.

4 create_shapes()
Based on the number of $App_{sec}$ tasks and the number of tasks the PEs execute, it creates a set of rectangular shapes that can receive $App_{sec}$. Many shapes are created and sorted, being the first the closest to a square without CPU sharing (i.e., 1 task per PE).

5 shape_location()
This function sweeps the platform with the shape set to find a free region that can receive $App_{sec}$. Initially, it goes from the largest shapes to the smaller ones, without

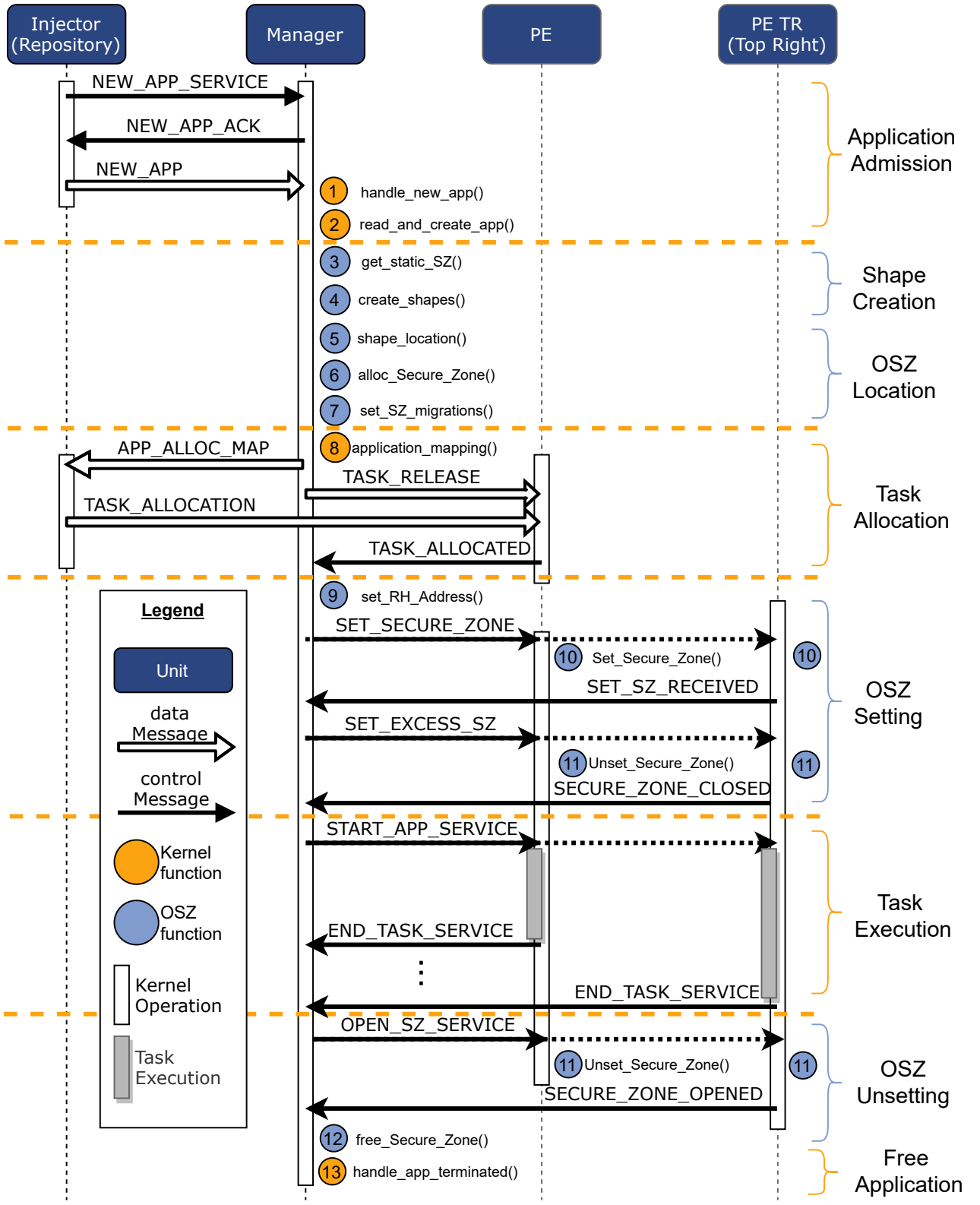



Figure D.1 – Sequence diagram of the OSZ method.

CPU sharing. Next, the shapes that assume two tasks per PE are used. The process continues until the maximum amount of tasks the PEs can execute is reached.

6. alloc_Secure_Zone()
   This function registers the selected shape position and size in an internal structure of the manager.

7. set_SZ_migrations()
   If the shape_location() function returns null, it is necessary to free some PEs to map $App_{sec}$. In this case, the shape_location() sorts the shapes in the reverse order, starting with the maximum allowed CPU sharing and the smaller shapes first. The goal is to select the region that minimizes the number of task migrations. Once selected the shape location and size, the manager fire the task(s) migration(s) and re-executes the alloc_Secure_Zone() function. Note that if the system has most of its resources used, it may be unfeasible to admit $App_{sec}$.

9. set_RH_address()
   Stores the address of the PE which is in the upper right corner of the OSZ (**PE TR**). This is the most distant PE from the **Manager** (mapped at PE (0,0)), being the PE that will notify that the OSZ has been closed.

10. Set_Secure_Zone()
    When the **Manager** sends the **SET_SECURE_ZONE** control message, the PEs that will execute $App_{sec}$ tasks verify if they are at the borders of the OSZ. If true, the kernel computes which the wrappers to set. The **PE TR** notifies the **Manager** that the OSZ was closed, through the **SET_SZ_RECEIVED** control message.

11. Unset_Secure_Zone()
    This function has two roles. The first one is to remove PEs not used in the OSZ. The mapping function verifies the number of required PEs to execute $App_{sec}$, leaving the leftmost PEs, from the bottom to the top, free. Thus, this function changes the wrapper status of the PEs not used in the OSZ, modifying the OSZ shape.

12. free_Secure_Zone()
    This function, executed by the he manager, clears the structure with the information related to the OSZ of the $App_{sec}$ that finished.


Pontifícia Universidade Católica do Rio Grande do Sul
Pró-Reitoria de Pesquisa e Pós-Graduação
Av. Ipiranga, 6681 – Prédio 1 – Térreo
Porto Alegre – RS – Brasil
Fone: (51) 3320-3513
E-mail: propesq@pucrs.br
Site: www.pucrs.br