Received October 26, 2021, accepted November 8, 2021, date of publication November 11, 2021, date of current version November 19, 2021.

*Digital Object Identifier 10.1109/ACCESS.2021.3127468*

# Detection and Countermeasures of Security Attacks and Faults on NoC-Based Many-Cores

**RAFAEL FOLLMANN FACCENDA[1], LUCIANO L. CAIMI[2], AND FERNANDO GEHM MORAES[1], (Senior Member, IEEE)**

[1]Pontifical Catholic University of Rio Grande do Sul (PUCRS), School of Technology, Porto Alegre 90619-900, Brazil
[2]Department of Computer Science, Federal University of Fronteira Sul (UFFS), Chapecó 89815-899, Brazil

Corresponding author: Fernando Gehm Moraes (fernando.moraes@pucrs.br)

This work was supported in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior –Brasil (CAPES)—Finance Code 001. The work of Fernando Gehm Moraes was supported by CNPq (309605/2020-2), Brazilian funding agency.

**ABSTRACT** The modularization and manufacture of many-cores system-on-chip that involve several vendors open up a vulnerability: the inclusion of Hardware Trojans (HT). In addition to that, the reduced feature size of transistors may accelerate aging effects, leading to faults. The literature presents techniques to tackle security and fault-tolerance, such as cryptography, authentication codes, error correction codes, creation at runtime of flow profiles to detect anomalous behavior. However, at the communication level (i.e., NoC), there is a gap in generic methods to detect attacks or faults. As detailed in the state-of-the-art session, approaches targeting the NoC protection against attacks add additional hardware in the NoC itself, which is prone to security attacks or faults. This work decouples the detection of attacks or faults by using data and control NoCs. The adoption of a control NoC enables the proposal of the Communication Session Protocol to monitor message exchange, detect abnormal behavior, and recover the communication from an eventual failure or attack. The execution time overhead varies according to the application communication model, from 3.5% to 33%. Such overhead is acceptable because once detected an abnormal communication behavior, the protocol changes the path between communicating task pairs and resumes the application execution.

**INDEX TERMS** Security countermeasures, fault-tolerance, hardware trojans, NoC-based many cores.

## I. INTRODUCTION

Many-core Systems on Chip (*MCSoCs*) are platforms designed to provide high-performance systems based on parallelism, meeting the current demand of embedded devices with power consumption and communication constraints [1]. Examples of modern architectures with a large number of processors interconnected by NoCs (Networks-on-Chip) includes the Mellanox family TILE-Gx72 (72 cores) [2], Intel Knights Landing [3], Oracle M8 (32 cores) [4], Kalray array (256 cores) [5], KiloCore chip (1,000 cores) [6], and Esperanto (1,100 RISC-V cores) [7].

The increasing number of different features and functionalities inside a single chip also increases the variety of third-party IPs (3PIPs). Such IPs come from different vendors due to competitive prices and time-to-market. The presence of 3PIPs raises the risk of having a *Hardware Trojan* (HT) insertion [8]. Assuming HTs infect the NoC, these can perform several types of attacks that threaten security principles [9]. 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.

Besides HT attacks, eventual faults may violate the system *dependability*. Dependability is the ability to deliver service that can justifiably be trusted [10]. Dependability includes the following attributes: availability, reliability, safety, integrity, and maintainability. The literature presents works focusing on communication dependability using secure cryptography key exchange [11] and cooperative communication [12] in environments as wireless network sensors, differently to our proposal, targeting *MCSoCs*.

Thus, in the context of *MCSoCs*, a unified method must be proposed 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.

The associate editor coordinating the review of this manuscript and approving it for publication was Mario Donato Marino.

The *motivation* of this work is the following question: "how to make the communication between processing elements safe and fault-tolerant"? The literature presents several techniques, such as cryptography [13], authentication codes [14], error correction codes [15], creation at runtime a flow profile to detect anomalous behavior [16]. Adopting these techniques makes it possible to detect violations related to security or faults in the NoC. However, they do not provide a generic method to detect the attacks or faults and evidence for the execution of countermeasures. For example, a PE may detect a tampered packet, then discard it and request a retransmission. This retransmission will probably use the same path, and the new packet will arrive with the same problem.

The *goal* of this work is to propose an original method to secure the communication between processing elements. This method relies on three pillars: (1) control NoC with broadcast transmission; (2) data network with support for XY and source routing; (3) an additional layer in the message exchange mechanism. The Processing Element (PE) operating system (OS) has exclusive access to the control NoC, thus preventing access to it by malicious applications. The message exchange mechanism involves at least two packets, requests for data, and data transmission. Our proposal adds to each packet an additional packet sent by the control NoC, with a unique identifier to each communicating pair, named *session key*, or $K_s$. This method enables the receiver of a packet to confirm its authenticity using $K_s$. Decoupling the attack or fault detection method between the data and control NoCs helps in anomalies detection and decision-making regarding countermeasures or fault tolerance. If $K_s$ does not match the stored key, a possible countermeasure is the request to resend the packet, avoiding the original path. As the data network supports source routing, it computes a new path, circumventing the affected region.

The *original* contributions of this work concerning the state-of-the-art include:

- a unified method to handle security threats and faults, being possible to extend it to cope with aging effects and quality-of-service;
- a protocol that mitigates attacks or circumvents faults without the need to locate the issue. A new path avoids the affected path by using a clever rerouting mechanism;
- a proposal that decouples the communication medium (NoC) from the infrastructure responsible for dealing with security and fault issues. Proposals in the literature add hardware modules in the hardware prone to attacks and faults, i.e., the NoC. Our work adopts a parallel and simple NoC to detect these issues. This contribution enables the adoption of 3PIP NoCs, keeping the design safe.

This work is organized as follows. Section II presents the baseline architecture and the threat model. Section III discusses the related work. Section IV describes the main original contribution, the Communication Session Protocol. The protocol: (*i*) monitors the message exchange; (*ii*) detects abnormal behavior (possibly caused by a failure or an attack); (*iii*) recovers the communication from an eventual failure or attack. Section V evaluates the proposed protocol, and Section VI concludes this work and point-out directions for future work.

## II. SECURE ARCHITECTURE AND THREAT MODEL

Figure 1 presents the *MCSoC* reference architecture, a homogeneous NoC-based many-core. Each PE has a 32-bit RISC processor, a DMNI module (a network interface with DMA capabilities) [17], local dual-port memory, two routers, and wrappers. The reference architecture is derived from the public-available MEMPHIS [18] many-core.[1]

Two NoCs interconnect the PEs: *data* and *control* NoC. The data NoC transfers *data messages* exchanged by applications. The data NoC adopts duplicated physical channels, wormhole packet switching, simultaneous support for distributed XY, and source routing. The control NoC transfers the *control messages*, such as: *(i)* the set of PEs belonging to a secure zone; *(ii)* definition of a fault-free path to circumvent a secure zone. The control NoC has the following features: adoption of broadcast as the default transmission mode, bufferless router, each message has one flit. Its router has five internal blocks: two finite state machines, two arbiters, and an 8-slot CAM buffer. The control NoC router has a small area footprint, corresponding roughly to 20% of the data router [19].

Both NoCs contain *wrappers* in the flow control signals (*W* in Figure 1). When activated, the wrapper enables to discard all incoming and outcoming packets of a given port. This approach guarantees the creation of a logical barrier at the hardware level and thus the secure zones creation.

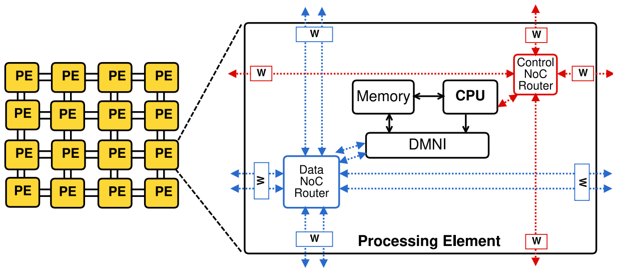


**FIGURE 1.** ***MCSoC*** **architecture (adapted from [20]).**

### A. SECURE ARCHITECTURE MODEL

Secure Zone (*SZ*) is a defense mechanism that spatially isolates a region of the many-core, reserving it for the execution of an application with security constraints ($App_{sec}$) [21]. *SZ*s isolate the $App_{sec}$ computation and communication from other applications running on the system.

A particular case of *SZ* is the Opaque Secure Zone (*OSZ*) [22]. *OSZ* is a defense mechanism executed at runtime. In summary, the method relies on finding a rectilinear region

[1]Available at: https://github.com/gaph-pucrs/Memphis.

on the system with PEs not executing user tasks to map $App_{sec}$. If there is no "free" region to map $App_{sec}$, the method migrates tasks to open space in the system. The *OSZ* activation occurs by setting wrappers ("W" in Figure 1) at the boundaries of the rectilinear region, blocking all incoming and outgoing traffic trying to cross the *OSZ*.

*OSZ* prevent attacks from outside sources, such as Denial-of-Service (DoS), timing attack, spoofing, man-in-the-middle [22], [23]. Even though the method is robust against external attacks, it still presents vulnerabilities when considering that data-NoC routers infected by HTs placed inside a Secure Zone.

The method we present to secure the communication is general and not coupled to the *OSZ* method. We chose the *OSZ* secure architecture because it prevents most of the attacks reported in the literature [22], [24].

### B. THREAT MODEL

Figure 2 illustrates an example of a DoS attack executed by an HT. Figure 2(a) presents a 3 × 3 system with two communicating tasks running on it: T1 and T2, and an HT (deactivated) in a router of the path. Figure 2(b) illustrates the HT activation. Thus, the HT blocks the communication after its activation, inducing the DoS attack (Figure 2(c)).

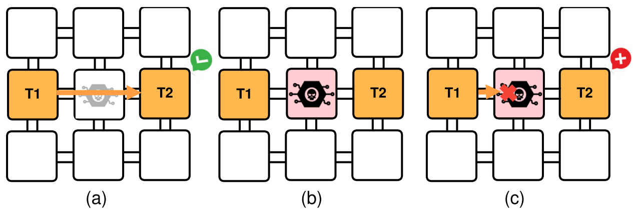


**FIGURE 2. Example of a Hardware Trojan affecting communication.**

According to [25], HTs can be *Always On* - meaning that they are always active, or *Triggered* - meaning they need to meet a condition to be activated, as the one used in the example above. HT triggers include time and physical condition (internal triggers) or user input and component output (external triggers).

Thus, even OSZ can be attacked internally by HTs, regardless the activation model. Assuming an NoC as a 3PIP, it can be infected by HTs, requiring countermeasures to avoid or mitigate the attacks. The *threat model* considered in this work includes the following DoS HT attacks [26]:

- 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 [27].
- 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.

The present work considers the CPU, memory, DMNI, control NoC, and the OS as reliable entities. Third-party entities like data NoC and applications are unreliable.

## III. RELATED WORK

This section presents works exploring the effects or countermeasures against attacks, including Hardware Trojans and DoS. Then, Section III-A presents a table summarizing the related work, comparing them to our proposal.

Charles and Mishra [28] propose a trust-aware routing to bypass malicious IPs during a message transmission through an NoC. In this work, the routers compute the *trust* they have in their neighbors. The trust values are computed following a trust model, adjusted at runtime. Whenever a packet does not reach the destination or the response does not come back to the source, the trust values of the routers are updated. Then, the packet is resent, and the routing now considers the new values of trust. However, this trust-aware routing uses the same NoC that is considered unsafe to update and configure the trust values, which might also be vulnerable to attacks. Our work also adopts routing to bypass malicious IPs, but using a control NoC to avoid paths in the data NoC that may be compromised by attacks.

Software-Defined Networking (SDN) is a paradigm that can be used as a countermeasure against attacks in the NoC. SDN removes the communication management from routers and moves it into a centralized controller responsible for configuring the routers. Ruaro *et al.* [29] adopt SDN to reduce the NoC energy consumption, provide QoS and fault tolerance. Moreover, the proposal adopts a secure method to configure the SDN routers using private keys. Experiments show that SDN can avoid attacks such as DoS, Flooding, and Spoofing. According to the Authors, the proposal is still vulnerable to HTs, lacking the discussion of methods that can be applied to prevent them.

Charles *et al.* [16] propose a detection and localization method of malicious IPs attacking the system with a distributed denial-of-service (DDoS) attack. The proposed framework is based on a communication model obtained at design time. At runtime, attacks are detected when performance boundaries are violated. In terms of localization of the IP causing the DoS attack, the authors use a congested graph containing information of all congested paths and routers of the system. Based on that, the attacker can be found most of the time. Similar to our threat model, this proposal focuses on DoS attacks, however, only in detection and localization, not exploring the recovery or defense mechanisms against such attacks on the system.

Zhang *et al.* [30] explore and evaluate the effects of two HT-based DoS-attacks: *blackhole* and *sinkhole*. The *blackhole* attack occurs when an infected router stops transmitting forward the packets that it receives, dropping the packet, or sending to unsafe destinations. The *sinkhole* attack works mostly on adaptive routing by requesting packets from neighbor routers, claiming to have buffer space. Then, when it receives the packet, it might also drop it or send it to another recipient. The authors explore several configurations of those attacks, varying parameters such as HT location, packet length, and the presence of a defense mechanism based on adaptive routing. In this case, HT-attacks are similar to our threat model. Nonetheless, the difference is that Zhang designed an NoC equipped with the defense mechanism is considered untrusted, making the countermeasure vulnerable, while we chose a control NoC to circumvent this issue since it is responsible for detecting and recovering from HT attacks.

Chaves *et al.* [31] propose detecting DoS (flooding) attacks in an NoC by monitoring packet collision, being able to locate the collision point and the directions of the malicious traffic. They propose to enhance the routers with a DoS monitor, which supervises the latency of the packets and reports it to the OS of the PE processor. When reaching the OS, the latency values are analyzed, triggering a DoS suspicion report in case of latencies out of the expected value, identifying the router where the collision happened. To detect the direction of the malicious traffic, the authors also proposed the store in the packet the inputs that competed to enter the sensitive traffic. Thus, when reaching the OS, it is possible to extract the direction from the malicious traffic. More recently, Chaves *et al.* in [32] explored protection against flooding DoS, proposing an extra monitor in the Network Interface, which monitors the length of the packets, reporting the presence of large malicious packets intended to cause congestion in the network. Similar to our proposal, both works present approaches for detecting DoS attacks, however, without detailing the recovery method, as our proposal does.

Hussain *et al.* [33] propose an *Energy Efficient Trojan Detection* design (EETD) to detect the presence of HTs. The proposal is based on two detection units. The first one, named End-to-end Trojan Detection Units (EDU), is placed at the PE and is always active. The EDUs are responsible for packet authentication and send an attack detection signal to a detection management unit. The second detection unit is named Localization Units (LUs), placed at each routers port and are power gated to save energy. The work can detect HTs with low performance overhead and energy consumption compared to state-of-the-art techniques. However, the difference of this work with ours is that this work does not present security countermeasures once an attack is detected.

Daoud and Rafla explore the effects of the *blackhole* attack [27] and a method to detect this attack [34]. They propose a protocol based on inter-router acknowledgment, making it possible to detect which router dropped a packet. Once again, the instrumentation is in the router, which is a structure considered vulnerable to attacks. Even though the authors also focus on detecting *blackhole* attack, they do not present a recovery procedure.

Raparti *et al.* [35] propose a security mechanism against a snooping attack caused by HTs. The HT could transmit a packet to the wrong destination (misrouting). The Snooping Invalidation Module aims to discard packets with invalid headers, thus, preventing sensitive information from reaching undesired targets. Furthermore, they present THANOS, a mechanism that observes the network and sends an alert in case of suspicious behavior. However, according to the authors, THANOS only mitigates the attacks. Thus, still requires a more robust defense mechanism that could completely recover the system against such attack.

Harttung *et al.* [36] and Moriam *et al.* [37] use cryptography and authentication to protect the system against HTs that can tamper or drop messages in the NoC. Three authentication approaches with symmetric cryptography are considered to detect modifications on the messages, which are discarded in case of tempering. Furthermore, the receiver can detect a packet loss via timeout and request a retransmission. This work is similar to ours since it detects the attack with a timeout mechanism. However, the recovery protocol occurs in the same network that tempered the received packet, thus also being exposed to HT attacks. Moreover, the recovery protocol is not detailed by the authors.

JYV *et al.* [38] consider four HT types: Flit Quantity Trojan (QT), Address Trojan (AT), and Head Hardware Trojan (HHT), and Tail Hardware Trojan (THT). The proposed solution to mitigate the action of the HTs is a Shuffle Encoder placed before the Input FIFO, obfuscating the information inside the packet and avoiding the triggering of the HT that could affect the packet. Simulation results in a $4 \times 4$ NoC show that the proposed method is efficient in thwarting the HT attacks. This work presents versions of HTs that are similar to the ones discussed in our Threat Model. However, the solution is based on instrumenting the router that is considered not trusted, affecting the reliability of the security mechanism.

Hazra *et al.* [39] also model four HT types and propose a detection mechanism. The detection mechanism is based on machine learning techniques such as Decision Tree (DT), Support Vector Machine (SVN), and K-Nearest Neighbor (KNN). The learning phase uses the total execution time, the power consumption, the total number of instructions executed, the total number of read misses and the total number of write misses. The Authors evaluate the accuracy of prediction for each different models of Trojans separately. The DT and SVM are the methods with better detection accuracy. Related to our proposal, this work focus on detecting an HT attack. However, as discussed above, there is a lack of works that focus on recovery besides the detection of attacks.

Sinha *et al.* [40] also use machine learning (ML) techniques to enhance system security. The Authors propose *Sniffer*, a tool to detect and locate flood DoS attacks. The authors train the ML module with network features such as

Buffer Waiting Time, Inter-Flit Interval, and Virtual Channel Occupancy. Each router has a module to observe the features mentioned above and a module for detecting abnormal traffic based on the ML technique. This work does not present the recovery process, besides instrumenting the same NoC to detect the attack.

### A. DISCUSSION

Table 1 classifies the works in terms of the adopted security mechanism; the instrumentation, meaning the unit/units responsible for the security mechanisms; and if there is a detection and recovery protocol. Daoud *et al.* [27], Charles *et al.* [16] and Hazra *et al.* [39] are examples of works that focus only on the detection of attacks, instead of implementing a countermeasure.

**TABLE 1. Related work summary.**

| Work | Method | Instrumentation | Detection | Recovery |
|---|---|---|---|---|
| Charles [28] (2020) | Adapt. Routing: Trust-aware Routing | Data NoC (Router) | - | - |
| Ruaro [29] (2020) | Adapt. Routing: SDN | SDN Subnet | - | - |
| Charles [16] (2020) | Detection and Localization | Data NoC (Router) | ✓ | - |
| Chaves [31] [32] (2019) (2021) | Latency and Collision Monitoring | Data NoC (Router) | ✓ | - |
| Zhang [30] (2018) | Detection: Traffic collision | Data NoC (Router) | ✓ | - |
| Hussain [33] (2018) | Authentication | Router | ✓ | - |
| Daoud [34] [27] (2019) | Adapt. Routing: Inter-router Ack. | Data NoC (Router) | ✓ | - |
| Raparti [35] (2019) | Packet Monitoring | Data NoC NI | ✓ | - |
| Harttung [36](2019) Moriam [37](2018) | Criptography and Authentication | Data NoC Operating System | ✓ | ✓ |
| Jyv [38] (2018) | Obfuscation | Data NoC (Router) | ✓ | - |
| Hazra [39] (2018) | Detection: Machine Learning | Cache | ✓ | - |
| Sinha [40] (2021) | Detection: Machine Learning | Router | ✓ | - |
| This Work | Adapt. Routing Authentication | Control NoC Operating System | ✓ | ✓ |

Adaptative routing is an approach used by several Authors when the goal is to protect the communication against DoS attacks and HTs. However, implementations are mostly inserted in routers assumed to be vulnerable against HTs, which compromises the reliability of the security mechanism. Moreover, even though most works focus on detecting HTs and suspicious behavior, only a few of them propose a recovery protocol.

Our proposal adopts a detection and recovery protocol that uses a control NoC to monitor packet transmission and find alternative paths to avoid faulty or malicious nodes of an NoC. The adoption of a control NoC enables the detection of suspicious behaviors on 3PIP NoCs. It is possible to insert an HT in the control NoC, but it can be easily detected. The detection is considered simple because the control NoC router has a low complexity design and a small silicon area footprint (20% of the data router [19]). For this reason, the control NoC is considered a reliable entity.

## IV. COMMUNICATION SESSION PROTOCOL

The Introduction Section mentioned the three pillars of the proposal: (1) control NoC; (2) data network with support for XY and source routing; (3) an additional layer in the message exchange mechanism.

The *control NoC* [19] is a lightweight network-on-chip, with all packets having one flit. When transmitting in broadcast, default transmission mode, packets reach all PEs of the system. Thus, this NoC can find a path from a source PE to a target PE if it exists, even in the presence of a fault or an HT in the path of the data NoC. This NoC may also use the unicast transmission to create a path between a source and a target PE, using a backtracking procedure. For security reasons, only the OS accesses the control NoC, avoiding its use by malicious applications.

The *data NoC* is a standard wormhole packet switching NoC without virtual channels. It has two particular architectural features. The first one is the adoption of two physical channels, acting as two disjoint NoCs. The flit size is half of the word size to minimize the area overhead, being the network interface responsible for serializing/deserializing the flits. The reason to adopt two physical NoC is to enable fully adaptive routing. The second feature is simultaneous support for XY (default routing algorithm) and source routing (SR). Source routing is required when, e.g., it is necessary to circumvent an *OSZ* or avoid a path with a faulty or infected router.

The following subsections detail the third pillar of the method, the additional layer in the message exchange mechanism. This layer is the *communication session protocol*, able to supervise message exchanges, detect suspicious behaviors and recover from attacks or failures.

### A. MONITORING MESSAGE EXCHANGE

The protocol starts by establishing a *session* (Definition 1). Figure 3(b) presents the sequence diagram with each step of the protocol, from its creation up to its end.

*Definition 1 (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 to the communicating pair.

*Definition 2 (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.

The consumer starts the protocol in the first `Message_Request` packet, transmitted using the data NoC. In parallel, through the control NoC, the consumer sends a `Start_Session` packet to the producer, with $K_s$ in its payload (Definition 2). When receiving this packet, the producer starts the session, by using $rnd$ and the task identifiers. When the producer delivers the requested message, it also transmits the `Session_Ack` packet confirming the successful session creation, using the control NoC. The `Session_Ack` also

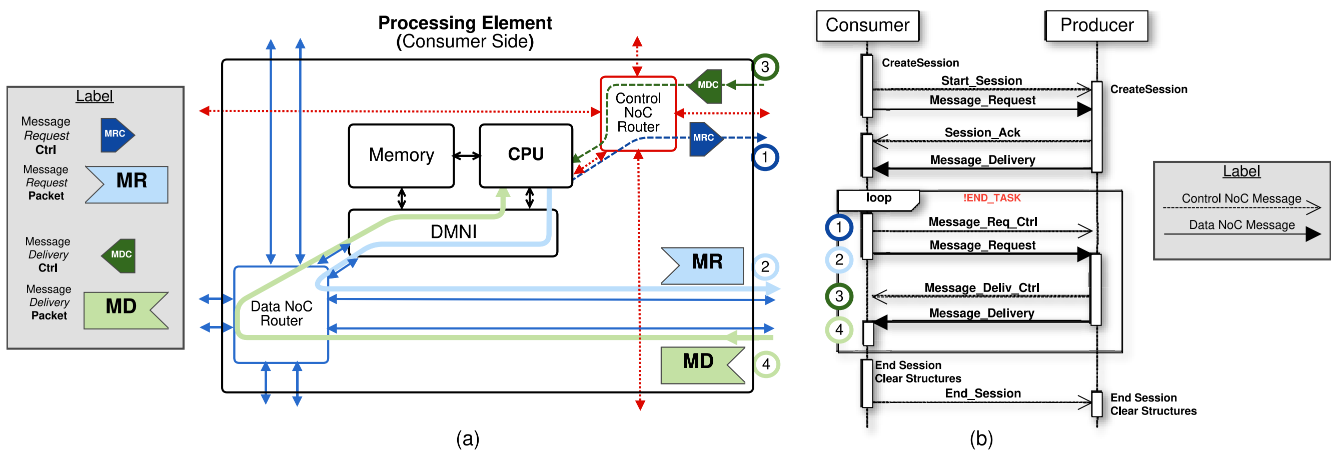


**FIGURE 3.** (a) Processing Element internal organization and the path taken from each step of the protocol. (b) Sequence Diagram of the Session Manager operation between the consumer and producer Tasks. The numbers 1-4 indicate each of the protocol steps spatially (a) and temporally (b).

contains $K_s$ to avoid tampered acknowledgments. At this point, both sides of the communication have $K_s$ and can exchange packets. Although not implemented in this work, additional security can be obtained by sending a cryptographic key at system startup, allowing to send the session key of each communicating pair encrypted, such as [23], [24].

Then, after the creation of the Session, the messages exchange occurs as follows:

- The consumer sends two packets to the producer: **MRC** (`Message_Req_Ctrl`) through the control NoC and **MR** (`Message_Request`) through the data NoC. Numbers 1 and 2 in Figure 3 illustrate this step.
- Once the producer receives the **MRC**, **MR** and has data to send, it transmits two packets to the consumer: **MDC** (`Message_Deliv_ Ctrl`) through the control NoC, and **MD** (`Message_ Delivery`) through the data NoC. Numbers 3 and 4 in Figure 3 illustrate this step.

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 (control NoC) to all tasks that produce data to this task. The producers close the session on their side when they receive a `End_Session` message, clearing all the 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 this work avoids these two problems. Data transmission does not inject packets into the network but stores them in the OS, until a consumption request, through the reception of a **MR** packet. Thus, a packet injected into the network is consumed by the receiver, ensuring message ordering, and avoiding network congestion.

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.

The data packet (**MR** or **MD**) enables the OS to retrieve part of $K_s$: $\{ID_p, ID_c\}$ – steps 2 and 4 in Figure 3. The control packet (**MRC** or **MDC**) contains the *rnd* value – steps 1 and 3 in Figure 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.

## B. DETECTING SUSPICIOUS BEHAVIOR

Three situations may signalize to the OS a suspicious behavior: (*i*) a mismatch when comparing $K_s$; (*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.

Figure 4(a) illustrates a correct packet reception. The straight yellow arrow corresponds to data packets, while the dashed purple arrow refers to control packets.

According to Section II-B, the proposed method may deal with:

- Packet Loss: the receiver PE knows that a data packet should arrive due to the reception of a control packet. The present work uses a 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 receive a control packet, it is discarded.
- 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 wrong sequence number or a latency higher than a threshold, which is also detected by the timeout mechanism.

Figure 4(b) illustrates a packet loss or permanent port blocking due to a fault in the router. The current work uses $K_s$ to detect the above threats. A sequence number in the data packet payload enables detecting intermittent packet blocking.

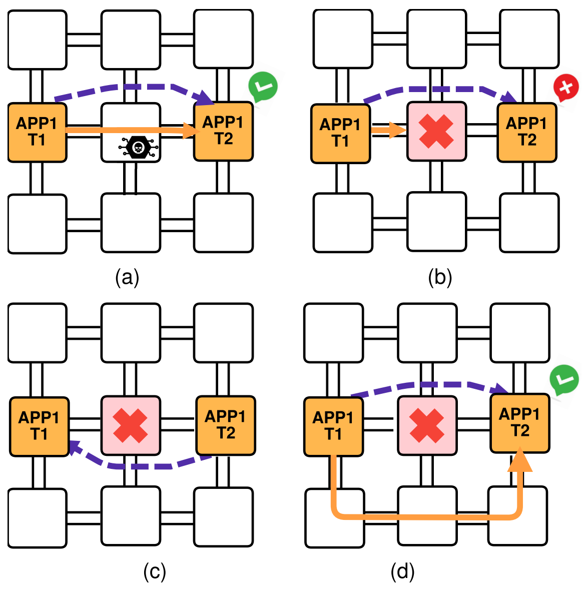


FIGURE 4. **Representation of a message recovery process using the control NoC. (a) Successful message transmission. (b) Data transmission interrupted in the data NoC due to a fault or HT. (c) Request for packet retransmission using the control NoC. (d) Successful retransmission using source routing. Dashed arrows: packets transmitted in broadcast using the control NoC. Straight arrows: packets transmitted through the data NoC.**

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 sequence number (currently embedded in the payload of the data packet), a Message Authentication Code (MAC), or a timestamp. The sequence number enables, e.g., the detection of packets dropped by an HT or a faulty link. A MAC enables the detection of corrupted packets. The timestamp allows for detecting anomalous variations on the latency, which may imply a timing attack.

### C. RECOVERING FROM ATTACKS OR FAILURES

After detecting an attack or fault, the receiver starts the recovery process. Here, the control NoC also plays a major role. Instead of wasting resources to detect the HT location [16] or the faulty link [31], the proposal adopts a rerouting mechanism. To find a new path, the control NoC avoids the output port used by the producer task and the input port used by the consumer task. These two rules ensure a new path without using the previous routers due to the restrictions imposed on the broadcast transmission.

Figure 4(c) shows the consumer PE notifying the producer PE a missing packet detected by the timeout (could be dropped or received in an incorrect order). The use of the control NoC and the broadcast transmission ensures the reception of this packet, avoiding the affected router(s). The producer PE injects a `Path_Search` message in the control NoC to the consumer PE. If a path exists, this packet arrives at the consumer PE, and the consumer PE starts a backtracking process to the producer PE, with the new path, as shown in Figure 4(d).

Note that the focus of this work is not the detection of the HT(s) or faulty router(s) location. The method uses a path search approach that avoids the previous path. With the new path, the producer PE resends the lost packet (as explained previously, the producer stores the message in a local buffer) using source routing (SR). All subsequent packets use this path up to the detection of a new event. Note that once defined the 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 (e.g., Figure 4).

## V. RESULTS

This section evaluates the Communication Session Protocol in terms of application execution time overhead (Section V-A), overhead on the session handling routines (Section V-B), and analysis of the recovery protocol impact on real benchmarks (Section V-C).

The many-core, described in Section II, is modeled at the RTL level (NoCs and DMNI in VHDL, and the processor and memory in SystemC) [20]. The OS and applications are described in C language. Such low-level simulation generates clock-cycle accurate results.

Each application is a $CTG(T, E)$ (Communicating task Graph), a directed and connected graph. Each vertex $t_i \in T$ represents a task, and each edge $e_{ij} \in E$ represents the communication from $t_i$ to $t_j$. Tasks communicate with each other using a message-passing protocol, similar to MPI (Message Passing Interface). Results are gathered simulating seven benchmarks: DTW, MPEG, MWD, MPEG4, Dijkstra, VOPD, and AES in a $4 \times 4$ $MCSoC$. These benchmarks are used in the many-core research community [41], [42]. Figure 5 presents two application graphs corresponding to the MPEG and AES applications.

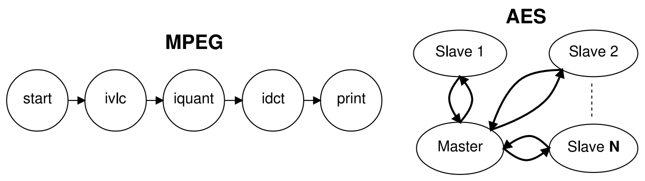


FIGURE 5. **CTG (Communicating Task Graphs) for MPEG and AES benchmarks. Note the for MPEG the inner tasks synchronize two sessions, while the Master task in the AES benchmark synchronizes 6 sessions.**

The benchmarks have distinct communication models, such as pipeline and master-slave, to avoid biased results. 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 benchmark is parameterizable in the number of slave tasks (4 or 8) and the number of 16-byte blocks to encrypt (8 up to 512). For example, AES_4_8 requires two iterations to encrypt or decrypt a 128-byte message (8 blocks of 16 bytes), while AES_8_8 requires only one iteration for the same workload.

### A. OVERHEAD OF THE COMMUNICATION PROTOCOL

Table 2 compares the applications' execution time between the baseline system and the one with the Communication Session protocol.

**TABLE 2.** Applications execution time with and without the protocol.

| Application | | Original (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 tasks acting as master tasks (*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 is used to illustrate the 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 (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 27% on application execution time is an acceptable cost considering the security and fault tolerance benefits added by the communication session protocol.

### B. OVERHEAD OF SESSION HANDLING ROUTINES

The Communication Session protocol adds additional computation in the OS to handle the protocol packets. There are two new algorithms: the handling of the **`MRC`** and **`MDC`** packets. In addition, the routines to handle **`MR`** and **`MD`** packets were modified to interface with the control NoC.

To analyze the impact of the protocol in the OS, Table 3 shows the time (in clock cycles) taken by the OS to handle the messages in the Session protocol, compared to the baseline implementation, considering 128 iterations of the application. The values refer to the AES 4_128 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 3.** 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 when the REQUEST arrives.

Table 3 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 2. 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.

### C. RECOVERY COST

After detecting a suspicious behavior, the recovery process starts, as detailed in Section IV-C. 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 6 illustrates the MPEG and HT mapping. This Figure shows the `MESSAGE_DELIVERY` and `MESSAGE_REQUEST` packets, according to the XY algorithm. The HT activation interrupts flows 3, 5, and 8.

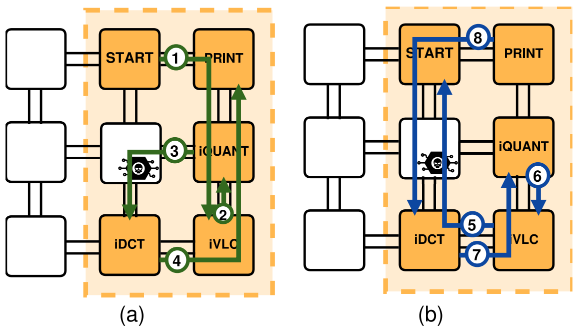


**FIGURE 6.** Task mapping of the MPEG application with the inter-task communication on a 3 × 3 Mesh NoC with a 2 × 3 OSZ (yellow area). (a) Message Delivery packets. (b) Message Request packets.

Each application executes ten iterations, with the HT configured to block all ports of the infected router at 3 ms. Figure 7 shows the time taken for each iteration for both applications. Each graph has three curves:

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

Figure 7(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 7(b) illustrates the AES application. This application also stalls at iteration 6. Due to the

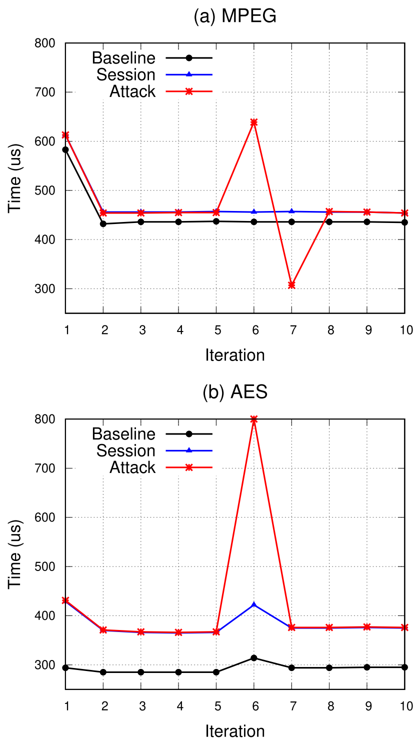


**FIGURE 7.** Impact of the Recovery Protocol on applications (a) MPEG (b) AES. X-axis: iteration number, y-axis: iteration latency.

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 7, the recovery process overhead happens once. After the recovery process, the HT is still active, but the applications are not affected by it. 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 4.** 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%) |

Table 4 presents the execution time for each scenario. The overheads using the Communication Session Protocol are according to the ones presented in Table 2, varying according to 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 when the number of executed iterations increases.

This work is an option for the ones that seek HT location and isolation. The method can detect anomalous behaviors and create a new path that avoids the original path through rerouting.

## VI. CONCLUSION

This work presented an original method to detect security attacks and faults in the communication architecture - NoC. 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 simple control NoC that uses broadcast transmission to find alternative paths to the paths used before detecting the attack or fault. The proposed method does not need to locate the source of the problem, which is the goal of most works found in the literature. The cost of our proposal is the increase in the execution time. On the other hand, the Communication Session Protocol operates efficiently to recover from attacks, such as Packet Loss, Packet Misrouting, and Port Blocking, caused by HTs or failures in NoC links.

This work does not exclude methods that locate HTs or faulty routers. Our proposed method combined with these 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.

Another future work direction is to make the synchronization of packet reception more flexible in master-slave applications. Currently, the master task must wait for packets from all the slave tasks sequentially, which impacts the execution time. This flexibility would help to reduce the overhead observed in master-slave applications.

## REFERENCES

[1] K. Popovici, F. Rousseau, A. A. Jerraya, and M. Wolf, *Embedded Software Design and Programming of Multiprocessor System-on-Chip: Simulink and System C Case Studies*. New York, NY, USA: Springer, 2010, p. 290. [Online]. Available: https://link.springer.com/book/10.1007/978-1-4419-5567-8

[2] M. Technologies. (Nov. 2018). *TILE-Gx72 Processor Overview*. [Online]. Available: http://www.mellanox.com

[3] A. Sodani, R. Gramunt, J. Corbal, H. S. Kim, K. Vinod, S. Chinthamani, S. Hutsell, R. Agarwal, and Y. C. Liu, "Knights landing: Second-generation Intel xeon phi product," *IEEE Micro*, vol. 36, no. 2, pp. 34–46, Mar./Apr. 2016, doi: 10.1109/MM.2016.25.

[4] Oracle, "Oracle's SPARC T8 and SPARC M8 server architecture," Oracle Corp., Santa Clara, CA, USA, White Paper, 2017. [Online]. Available: https://www.oracle.com/a/ocom/docs/sparc-t8-m8-server-architecture.pdf

[5] B. D. D. Dinechin, D. V. Amstel, M. Poulhies, and G. Lager, "Time-critical computing on a single-chip massively parallel processor," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2014, pp. 1–6, doi: 10.7873/DATE.2014.110.

[6] B. Bohnenstiehl, A. Stillmaker, J. Pimentel, T. Andreas, B. Liu, A. Tran, E. Adeagbo, and B. Baas, "A 5.8 pJ/Op 115 billion ops/sec, to 1.78 trillion ops/sec 32nm 1000-processor array," in *Proc. IEEE Symp. VLSI Circuits (VLSI-Circuits)*, Jun. 2016, pp. 1–2, doi: 10.1109/VLSIC.2016.7573511.

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

[8] H. Li, Q. Liu, and J. Zhang, "A survey of hardware Trojan threat and defense," *Integration*, vol. 55, pp. 426–437, Sep. 2016, doi: 10.1016/j.vlsi.2016.01.004.

[9] J. Ramachandran, *Designing Security Architecture Solutions*. Hoboken, NJ, USA: Wiley, 2002, p. 483.

[10] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Trans. Depend. Sec. Comput.*, vol. 1, no. 1, pp. 11–33, Jan./Mar. 2004, doi: 10.1109/TDSC.2004.2.

[11] G. Mehmood, M. S. Khan, A. Waheed, M. Zareei, M. Fayaz, T. Sadad, N. Kama, and A. Azmi, "An efficient and secure session key management scheme in wireless sensor network," *Complexity*, vol. 2021, pp. 1–10, Jun. 2021, doi: 10.1155/2021/6577492.

[12] G. Mehmood, M. Z. Khan, S. Abbas, M. Faisal, and H. U. Rahman, "An energy-efficient and cooperative fault-tolerant communication approach for wireless body area network," *IEEE Access*, vol. 8, pp. 69134–69147, 2020, doi: 10.1109/ACCESS.2020.2986268.

[13] S. Charles and P. Mishra, "Securing network-on-chip using incremental cryptography," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2020, pp. 168–175, doi: 10.1109/ISVLSI49217.2020.00039.

[14] G. Sharma, V. Kuchta, R. Anand Sahu, S. Ellinidou, S. Bala, O. Markowitch, and J. Dricot, "A twofold group key agreement protocol for NoC-based MPSoCs," *Trans. Emerg. Telecommun. Technol.*, vol. 30, no. 6, pp. 1–18, Jun. 2019, doi: 10.1002/ett.3633.

[15] H. A. H. Gondal, S. Fayyaz, A. Aftab, S. Nokhaiz, M. Bilal, and W. Saleem, "A method to detect and avoid hardware Trojan for network-on-chip architecture based on error correction code and junction router (ECCJR)," *Int. J. Adv. Comput. Sci. Appl.*, vol. 11, no. 4, pp. 581–586, 2020, doi: 10.14569/IJACSA.2020.0110476.

[16] S. Charles, Y. Lyu, and P. Mishra, "Real-time detection and localization of distributed DoS attacks in NoC-based SoCs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 12, pp. 4510–4523, Dec. 2020, doi: 10.1109/TCAD.2020.2972524.

[17] M. Ruaro, F. B. Lazzarotto, C. A. Marcon, and F. G. Moraes, "DMNI: A specialized network interface for NoC-based MPSoCs," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2016, pp. 1202–1205, doi: 10.1109/ISCAS.2016.7527462.

[18] M. Ruaro, L. L. Caimi, V. Fochi, and F. G. Moraes, "Memphis: A framework for heterogeneous many-core SoCs generation and validation," *Design Autom. Embedded Syst.*, vol. 23, nos. 3–4, pp. 103–122, Dec. 2019, doi: 10.1007/s10617-019-09223-4.

[19] E. Wachter, L. L. Caimi, V. Fochi, D. Munhoz, and F. G. Moraes, "BrNoC : A broadcast NoC for control messages in many-core systems," *Microelectron. J.*, vol. 68, pp. 69–77, Oct. 2017, doi: 10.1016/j.mejo.2017.08.010.

[20] L. L. Caimi, V. Fochi, E. Wächter, D. Munhoz, and F. G. Moraes, "Activation of secure zones in many-core systems with dynamic rerouting," in *Proc. ISCAS*, 2017, pp. 1–4, doi: 10.1109/ISCAS.2017.8050256.

[21] L. L. Caimi, V. Fochi, E. Wachter, and F. G. Moraes, "Runtime creation of continuous secure zones in many-core systems for secure applications," in *Proc. IEEE 9th Latin Amer. Symp. Circuits Syst. (LASCAS)*, Feb. 2018, pp. 1–4, doi: 10.1109/LASCAS.2018.8399904.

[22] L. L. Caimi and F. G. Moraes, "Security in many-core SoCs leveraged by opaque secure zones," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2019, pp. 471–476, doi: 10.1109/ISVLSI.2019.00091.

[23] L. L. Caimi, V. Fochi, and F. G. Moraes, "Secure admission of applications in many-cores," in *Proc. 25th IEEE Int. Conf. Electron., Circuits Syst. (ICECS)*, Dec. 2018, pp. 761–764, doi: 10.1109/ICECS.2018.8618021.

[24] M. Ruaro, L. L. Caimi, and F. G. Moraes, "SDN-based secure application admission and execution for many-cores," *IEEE Access*, vol. 8, pp. 177296–177306, 2020, doi: 10.1109/ACCESS.2020.3025206.

[25] B. Shakya, T. He, H. Salmani, D. Forte, S. Bhunia, and M. Tehranipoor, "Benchmarking of hardware Trojans and maliciously affected circuits," *J. Hardw. Syst. Secur.*, vol. 1, no. 1, pp. 85–102, Mar. 2017, doi: 10.1007/s41635-017-0001-6.

[26] J. Philomina, "A study on the effect of hardware Trojans in the performance of network on chip architectures," in *Proc. 8th Int. Conf. Smart Comput. Commun. (ICSCC)*, Jul. 2021, pp. 314–318, doi: 10.1109/ICSCC51209.2021.9528249.

[27] L. Daoud and N. Rafla, "Analysis of black hole router attack in network-on-chip," in *Proc. IEEE 62nd Int. Midwest Symp. Circuits Syst. (MWSCAS)*, Aug. 2019, pp. 69–72, doi: 10.1109/MWSCAS.2019.8884979.

[28] S. Charles and P. Mishra, "Lightweight and trust-aware routing in NoC-based SoCs," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2020, pp. 160–167, doi: 10.1109/ISVLSI49217.2020.00038.

[29] M. Ruaro, L. L. Caimi, and F. G. Moraes, "A systemic and secure SDN framework for NoC-based many-cores," *IEEE Access*, vol. 8, pp. 105997–106008, 2020, doi: 10.1109/ACCESS.2020.3000457.

[30] L. Zhang, X. Wang, Y. Jiang, M. Yang, T. Mak, and A. K. Singh, "Effectiveness of HT-assisted sinkhole and blackhole denial of service attacks targeting mesh networks-on-chip," *J. Syst. Archit.*, vol. 89, pp. 84–94, Sep. 2018, doi: 10.1016/j.sysarc.2018.07.005.

[31] C. G. Chaves, S. P. Azad, T. Hollstein, and J. Sep.úlveda, "DoS attack detection and path collision localization in NoC-based MpsoC architectures," *J. Low Power Electron. Appl.*, vol. 9, no. 1, pp. 1–20, 2019, doi: 10.3390/jlpea9010007.

[32] C. G. Chaves, J. Sepúlveda, and T. Hollstein, "Lightweight monitoring scheme for flooding DoS attack detection in multi-tenant MPSoCs," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2021, pp. 1–5, doi: 10.1109/ISCAS51556.2021.9401153.

[33] M. Hussain, A. Malekpour, H. Guo, and S. Parameswaran, "EETD: An energy efficient design for runtime hardware Trojan detection in untrusted network-on-chip," in *Proc. IEEE Comput. SoC. Annu. Symp. VLSI (ISVLSI)*, Jul. 2018, pp. 345–350, doi: 10.1109/ISVLSI.2018.00070.

[34] L. Daoud and N. Rafla, "Detection and prevention protocol for black hole attack in network-on-chip," in *Proc. 13rd IEEE/ACM Int. Symp. Netw. Chip*, Oct. 2019, p. 22, doi: 10.1145/3313231.3352374.

[35] V. Y. Raparti and S. Pasricha, "Lightweight mitigation of hardware Trojan attacks in NoC-based manycore computing," in *Proc. 56th Annu. Design Autom. Conf.*, Jun. 2019, pp. 1–6, doi: 10.1145/3316781.3317851.

[36] J. Harttung, E. Franz, S. Moriam, and P. Walther, "Lightweight authenticated encryption for network-on-chip communications," in *Proc. Great Lakes Symp. VLSI*, May 2019, pp. 33–38, doi: 10.1145/3299874.3317990.

[37] S. Moriam, E. Franz, P. Walther, A. Kumar, T. Strufe, and G. Fettweis, "Protecting communication in many-core systems against active attackers," in *Proc. Great Lakes Symp. VLSI*, May 2018, pp. 45–50, doi: 10.1145/3194554.3194582.

[38] M. K. J. Y. V., A. K. Swain, S. Kumar, S. R. Sahoo, and K. Mahapatra, "Run time mitigation of performance degradation hardware Trojan attacks in network on chip," in *Proc. IEEE Comput. SoC. Annu. Symp. VLSI (ISVLSI)*, Jul. 2018, pp. 738–743, doi: 10.1109/ISVLSI.2018.00139.

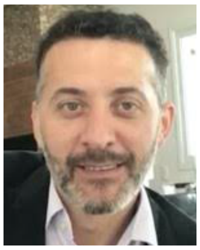
[39] S. Hazra, J. S. Sattenapalli, A. Roy, and M. Dalui, "Evaluation and detection of hardware Trojan for real-time many-core systems," in *Proc. 8th Int. Symp. Embedded Comput. Syst. Design (ISED)*, Dec. 2018, pp. 31–36, doi: 10.1109/ISED.2018.8703990.

[40] M. Sinha, S. Gupta, S. S. Rout, and S. Deb, "Sniffer: A machine learning approach for DoS attack localization in NoC-based SoCs," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 11, no. 2, pp. 278–291, Jun. 2021, doi: 10.1109/JETCAS.2021.3083289.

[41] W. N. Costa, L. P. Lima, and O. A. de Lima Junior, "Extracting method of packet dependence from NoC simulation traces using association rule mining," *Anal. Integr. Circuits Signal Process.*, vol. 106, no. 1, pp. 235–247, Jan. 2021, doi: 10.1007/s10470-020-01645-6.

[42] S. Kashi, A. Patooghy, D. Rahmati, and M. Fazeli, "An energy efficient synthesis flow for application specific SoC design," *Integration*, vol. 81, pp. 331–341, Nov. 2021, doi: 10.1016/j.vlsi.2021.08.005.

**RAFAEL FOLLMANN FACCENDA** received the B.Sc. degree in computer engineering and the M.Sc. degree from the Universidade Federal de Santa Maria (UFSM), Santa Maria, Brazil, in 2018 and 2020, respectively. He is currently pursuing the Ph.D. degree in computer science with PUCRS, Brazil. His research interests include many-cores, NoCs, embedded systems, and security for NoC-base many-cores.



**LUCIANO L. CAIMI** received the M.Sc. degree in electrical engineering from the Federal University of Santa Catarina (UFSC), Florianopolis, Brazil, in 1998, and the Ph.D. degree in computer science from PUCRS University, Porto Alegre, Brazil, in 2019. He is currently an Adjunct Professor with the Federal University of Fronteira Sul (UFFS). His research interests include multiprocessor systems on chip (MPSoCs) and security for embedded systems.

**FERNANDO GEHM MORAES** (Senior Member, IEEE) received the B.Sc. degree in electrical engineering and the M.Sc. degree from Universidade Federal do Rio Grande do Sul (UFRGS), Porto Alegre, Brazil, in 1987 and 1990, respectively, and the Ph.D. degree from the Laboratoire d'Informatique, Robotique et Microélectronique de Montpellier, France, in 1994. He has been a Full Professor with PUCRS, since 2002. He has authored and coauthored 47 peer-refereed journal articles in the field of VLSI design. His research interests include microelectronics, FPGAs, reconfigurable architectures, NoCs, and MPSoCs.