IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 3, 2020, accepted September 15, 2020, date of publication September 18, 2020,
date of current version October 8, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3025206

SDN-Based Secure Application Admission and Execution for Many-Cores

MARCELO RUARO[1], LUCIANO LORES CAIMI[2], (Member, IEEE), AND FERNANDO GEHM MORAES[1], (Senior Member, IEEE)
[1]School of Technology, Pontifical Catholic University of Rio Grande do Sul (PUCRS), 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 (CAPES), Brazil, under Grant Finance Code 001. The work of Fernando Gehm Moraes was supported in part by the Fundação de Amparo á pesquisa do Estado do Rio Grande do Sul (FAPERGS) under Grant 18/2551-0000501-0, and in part by the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) (Brazilian funding agencies) under Grant 302531/2016-5.

ABSTRACT General-purpose many-core system-on-chip (MCSoC) requires support to the execution of dynamic workloads, i.e., admission of new applications at runtime. Some applications may require QoS and security from the MCSoC, not tolerating that malicious tasks or hardware Trojans steal or corrupts their data. A robust method to provide security is to isolate the communication and computation. Most current works employ such isolation in continuous regions named secure zones (SZ). Motivated by the recent study of the Software-Defined Networking (SDN) paradigm for MCSoCs, this work proposes to use SDN-based management to implement the communication isolation at runtime. The computation isolation occurs by mapping only tasks of the same application at each core. The communication isolation is supported by the SDN paradigm, which establishes dedicating paths for secure applications. Results show that the SDN-based approach presents a negligible latency to admit and execute a secure application, with a reduced hardware cost and higher computational resources utilization compared to SZs.

INDEX TERMS Security, many-core, network-on-chip (NoC), secure zones, software-defined networking (SDN).

I. INTRODUCTION

Many-core Systems-on-Chip (MCSoCs) supporting dynamic workloads are ubiquitous in our lives as mobile smartphones, IoT and general embedded devices. The support of dynamic workloads requires adaptive techniques to admit new applications and fulfill their constraints at runtime.

Security is one of the constraints required by applications. An application can request the system to provide data integrity and protection against potential attacks that aim to steal or corrupt its data. The literature explores several techniques that enforce security addressing application admission and execution at the computation and communication levels. A method to protect the MCSoC design with prominence in the literature is the secure zone (SZ) [1]–[6]. The basis of this technique is the resource reservation to execute applications. For example, at the computational level, the application tasks do not share the same CPU with other applications. At the communication level, it is possible to protect flows by using dedicated routing algorithms, rerouting at runtime, cryptography, or firewalls.

Recent research showed the Software-Defined Networking (SDN) benefits for MCSoCs, targeting energy reduction and QoS [7]–[9]. The SDN paradigm reduces the Network-on-Chip (NoC) physical complexity by moving the router control logic from the hardware to the software level. The software managing the SDN is named *Controller*. The Controller keeps the NoC resources status and, based on its rules, defines paths at runtime for communicating tasks. The main benefit of SDN is its global knowledge of the NoC, which allows multi-objective communication path search and the possibility to the designer to update the search rules without the need to design a new NoC [7], [8].

Motivated by the SDN design flexibility, this work proposes a ''Secure SDN-based Application Admission and Execution'' (SSAE) framework to admit and execute applications securely. SSAE covers security at the computational and communication levels:

- *Computation*: a secure application is mapped in dedicated Processing Elements (PEs), i.e., its tasks are not

The associate editor coordinating the review of this manuscript and approving it for publication was Yanjiao Chen.

TABLE 1. Related works on security for MCSoCs. CMP = Computation support, CMM = Communication support.

Author	CMP	CMM	Method	Threat Model
Sepúlveda (2018) [10]	-	✓	PUF and MAC	Access control; Data integrity and authenticity
Rajesh (2015) [11]	-	✓	Firewall	Access control; Confidentiality
Kinsy (2017) [12]	-	✓	Firewall; Encryption	Memory access control; Data leakage and integrity; DoS
Oliveira (2018) [13]	-	✓	Firewall; Encryption	Data Confidentiality
Wassel (2014) [14]	-	✓	Temporal Network Partition	Timing SCA and DoS attacks mitigation
Boraten (2016) [15]	-	✓	Packet validation	Hardware Trojan; Data integrity
Reinbrecht (2017) [16]	-	✓	Routing Scheme	Timing Side-Channel Attack (SCA)
ARM (2008) [1]	✓	-	Secure Zones	Access control; Data integrity
Isakovic (2013) [2]	✓	✓	Secure Zones	Access control; Authentication; Data integrity
Fernandes (2016) [3]	-	✓	Secure Zones	Timing SCA and DoS attacks
Sepúlveda (2017) [4]	✓	✓	Secure Zones	Access control; Authentication; Confidentiality
Real (2018) [5]	✓	-	Secure Zones	Data integrity; Confidentiality inside the Cluster
Caimi (2019) [6]	✓	✓	Opaque Secure Zones	DoS, timing attack; Spoofing; Data integrity
This work	✓	✓	*SDN and Spatial Isolation*	*DoS, timing attack; Spoofing; Data integrity*

allowed to share a CPU with tasks belonging to other applications.

- *Communication*: SDN-based management defines exclusive communicating paths for secure flows using a Multiple-Physical NoC (MPN) [17], [18].

The *goal* of this work is the proposition and evaluation of SDN-based management to provide secure application admission and execution. The motivation for adopting SDN comes from its hypothetical advantages compared to SZs (evaluated in this work). The first one is the creation of dedicated connections enabling non-continuous regions, allowing a more flexible task mapping, and contributing to a better system utilization. The second one is to promote a lightweight physical design compared to approaches requiring dedicated hardware for SZs, encryption mechanisms or firewalls.

The original *contributions* of this work include:

- A framework to securely admit and execute applications into an MCSoC based on the SDN paradigm. The framework addresses a systemic protocol covering hardware and software components of the system.
- A comparison of the SDN-based secure management with a state-of-the-art SZ technique [6].

This work is organized as follows. Section II reviews related work on security and SDN for MCSoCs. Section III details the MCSoC architecture. Section IV describes the secure zone model, which is the reference model compared to our proposal. Section V presents the "Secure SDN-based Application Admission and Execution" (SSAE) framework, the main contribution of this work. Section VI evaluates SSAE in terms of computational complexity, area, and power, as well as compare it against the model presented in Section IV. Finally, Section VII concludes this article and point out directions for future work.

II. RELATED WORK

A. SECURITY FOR MCSoC

Table 1 presents a broad overview of works covering security for applications executing in MCSoCs. The Table shows the adoption of different methods, including Physical Unclonable Function (PUF) and Message Authentication Code (MAC) [10], firewalls [11]–[13], encryption [12], [13], temporal communication partition [14], packet validation [15], routing schemes [16], and Secure Zones (SZ).

Works [1]–[6] address SZ. Such proposals vary according to some characteristics, as *creation time* (design time [3] or runtime [4], [5]), *shape* (discontinuous [3], [4] or continuous [2], [6]), *resource sharing* (communication, computation, none), and *isolation* method (cryptography [4], specialized routing algorithms [3], rerouting [6], and spatial or temporal isolation [5]).

Table 1 shows that SZ is the only technique protecting at the same time computation (CMP column) and communication (CMM column) levels [2], [4], [6]. Other methods focus on the protection of flows traversing the NoC, not addressing security at the PE level. Some SZ approaches isolate PEs [1], [5], without protecting the NoC flows. Thus, our motivation is to study the effect of SDN-based management for secure communication and computation execution compared to a state-of-the-art SZ technique [6].

The last Table column presents the threat model. The proposal prevents attacks related to the communication (as DoS and timing attacks) due to the reservation of the links for the communication, which occurs through circuit switching. Therefore, it is impossible to attack the application's flows. Data confidentiality and integrity are guaranteed because processors do not share the computation with any other application, thus preventing spoofing and hijacking attacks. Note that the proposed method guarantees data confidentiality and integrity without using encryption. This feature provides two advantages over solutions that use cryptography, a smaller hardware implementation cost and no delay due to encryption and decryption of messages. Thus, the performance of applications presents better performance compared to methods requiring flow encryption.

The proposal and the SZ method from Caimi *et al.* [6] share the same thread model, but with different methods implemented at the hardware level.

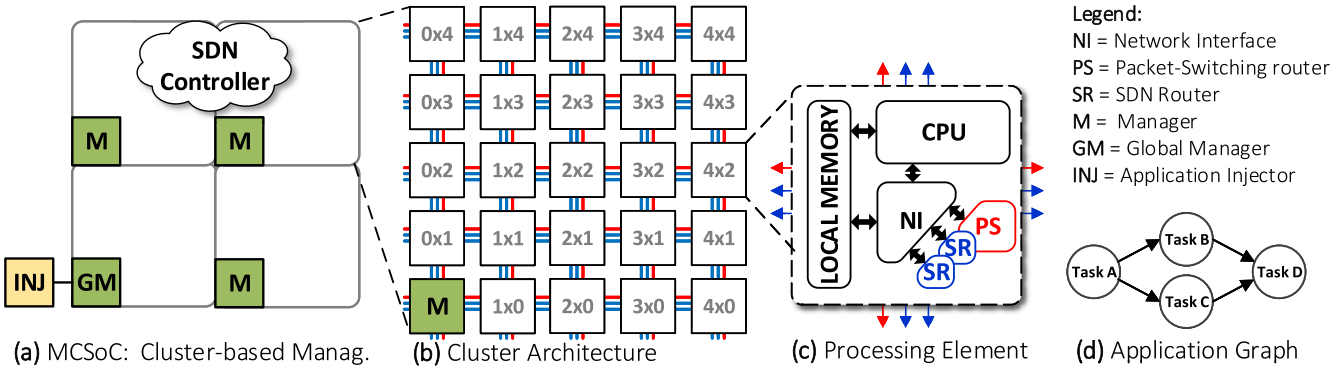

FIGURE 1. Many-core SoC (MCSoC) architecture.

B. SDN FOR MCSoC

SDN comes originally from computer networks [19]. Its main advantage is to unify the management of different device vendors, removing the control from the network device, and bring it to a software Controller. Recent works investigate SDN for MCSoCs using a centralized approach [8], [9], [20], while others propose distributed SDN management [7], [21], including cluster-based organizations [7].

Some SDN proposals target specific goals, as power management [21] and QoS [8], while other works are generic, focusing on the SDN paradigm support over the MCSoC architecture [9], [22]. Ellinidou *et al.* [20] address security, proposing a secure protocol to the Controller to configure SDN routers at runtime, using an architecture based on a Chiplet design, and not MCSoCs. There is a gap covering the adoption of *SDN to provide security for the user's application admission and execution*, which this work aims to fulfill.

III. MCSoC ARCHITECTURE

This Section presents the basics of the MCSoC architecture required for the understanding of our proposal.

A. HARDWARE

Figure 1(a) presents a global view of the MCSoC architecture, connected to an external device, *INJ* (application injector), responsible for deploying applications in the system. The MCSoC adopts cluster-based management [23], with one manager PE per cluster. A particular manager, GM (Global Manager), receives requests from *INJ*.

Figure 1(b) details a cluster, while Figure 1(c) the PE. A PE contains a CPU, memory (local memory or cache), a Network Interface (NI), and a set of routers that connect the PE to other PEs.

A Multiple-Physical NoC (MPN) [17], [18] interconnects PEs. The MPN contains disjoint subnets. One subnet is dedicated for Packet-Switching (PS). The other subnets (parameterizable number at design time) correspond to SDN routers (SR), implementing dedicated connections (CS – Circuit-Switching). SR routers do not have routing logic, being configured at runtime by the path definition rules of a Controller running at the management level (next Subsection).

The PS subnet carries management packets and best-effort applications' data flows. The SDN subnets transmit real-time and secure packets since their connections do not share data with other flows.

PS routers adopt wormhole packet switching, credit-based flow control, and input buffering (usually 8-flit depth). SR routers do not have control logic (routing and arbitration) since they are simple units that transmit data in streaming. A flit coming from a given input port is sent to one of the output ports, after one clock cycle, according to the configuration made by the Controller. The one clock cycle delay is due to the buffering process, avoiding long wires. Some CS designs assume it is possible to transmit a flit from a source to a target PE within a clock cycle [24], which is not physically feasible.

B. SYSTEM MANAGEMENT

System management is implemented at the software level. A given PE can run one of three classes of management software: slave, manager, global manager.

- **Slave PE (SP)**: runs a tiny OS (∼10KB) designed to support the user's task execution (multi-task), providing task scheduling, inter-task communication API, and interruption handling.
- **Manager PE (M)**: executes task mapping and controls at runtime constraints fulfillment by executing self-adaptive techniques, e.g. task migration and DVFS control.
- **Global Manager PE (GM)**: performs the same actions of an M PE and receives requests from the *INJ* peripheral, admitting new applications and selecting the cluster where the application will be mapped.

SPs run user's tasks, while manager PEs only execute management functions. As MCSoCs may have dozen to hundreds of PEs, we adopt hierarchical cluster-based management. A cluster has a set of SPs, managed by one M PE. Figure 1(a) shows an MCSoC instance with four clusters, and Figure 1(b) overviews one cluster. One M PE is set as GM at design time (typically the one closer to *INJ*).

Figure 1(a) contains the **SDN controller** (*Controller*), which may be a task running in a given PE (centralized SDN [8], [9], [20]) or several tasks distributed in the system

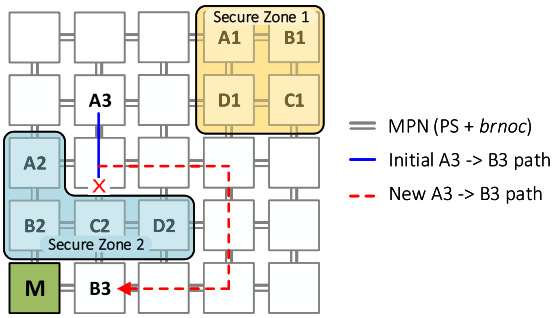

FIGURE 2. Example of two OSZs, and one non-secure flow (A3 ⇒ B3).

(distributed SDN [7], [21]). The *Controller* is a task with OS privileges. It searches and configures dedicated paths according to requests made by an M PE. M PEs can request a path for communicating tasks when they have security or QoS constraints, or a fault is detected in a path already established. The Controller finds path based on its rules (e.g., the shortest-path that avoids faulty-routers and hot-spots regions), configure the path physically and reply to the M PE the path definition result. The M PE then configure the communicating task pair to use the new path. Further details about the Controller implementation are available at [7], [25], including a distributed Controller implementation.

C. USER'S APPLICATIONS

Communicating task graphs describes applications, like the one depicted in Figure 1(d). Tasks communicate through message passing. A message copies a memory region from the producer task, transmitting it to the receiver task's memory.

IV. SECURE ZONE MODEL

This section details the SZ model adopted for comparison purposes [6] – Opaque Secure Zones (*OSZ*). That proposal differentiates from other SZ methods because the secure region is opaque, i.e., flows belonging to other applications are forbidden to cross the OSZ. Such characteristic enables integrity and confidentiality to the application's data exchange, inhibits denial-of-service (DOS) and timing side-channel attacks (SCA) because it is not possible to disturb flows inside the OSZ. Figure 2 illustrates two OSZs, each one with a 4-task application mapped on it.

The OSZ approach assumes an MPN NoC with two PS subnets and a broadcast subnet. The broadcast subnet is a specialized NoC, named *brNoC* [26], used to send management commands using broadcast transmission. The M PEs are in charge of creating and closing OSZs at runtime, executing the following steps [6]:

- *shape selection*: reservation of a continuous rectilinear region, with PEs belonging to this region without executing any user task. If it is impossible to reserve the set of PEs required to run the application, it is possible to use task migration to create the OSZ region.
- *task allocation*: the M PE maps the application's tasks inside the OSZ, sending the tuples {*task_ID*, *address*} to the *INJ* peripheral, which transmits the object codes to the PEs. Each task object code has a MAC attached to it, verified during its reception. The MAC ensure integrity to the object code load.
- *wrapper activation*: the M PE actives wrappers at the OSZ boundary after task allocation, closing the OSZ to any packet trying to leave or enter the OSZ through the PS subnets. The M PE uses the *brNoC* for this step.
- *application initialization*: the M PE notifies the OSZ's PEs to start the application, by using the *brNoC*.
- *application ending*: at the end of the execution of all application's tasks, the M PE requests the wrappers' opening using the *brNoC*. Each slave PE clears it's memory content and deactivates the wrappers.

Packets hitting the OSZ boundary are retransmitted without crossing it. Consider the example presented in Figure 2, where task A3 transmits packets to B3. Once the OSZ closed, a packet transmitted by A3 hitting the OSZ is dropped, and the *brNoC* notifies A3 that the path is broken. The OS on which A3 is running requests to the *brNoC* to find a new path for B3. After finding a new path, all packets transmitted to B3 uses source routing, represented by the dashed red path between A3 and B3.

The OSZ advantage is to secure the applications' execution. However, we identify limitations related to OSZs:

- *Task mapping*. In systems with a small utilization, the mapper quickly finds continuous regions. With the system utilization increase, the availability of continuous regions reduces, preventing the application admission or requiring task migration to "open" space in the system. Also, opaque regions inside the system lead to fragmentation of non-secure applications, increasing the communication energy [27].
- *Communication of non-secure applications*. The PS NoC must support different routing mechanisms (as XY and source routing), and the shape selection step must avoid the creation of unreachable regions. The rerouting process impacts the performance of non-secure applications due to the adoption of longer paths.

V. SSAE FRAMEWORK

This Section presents the "Secure SDN-based Application Admission and Execution" (SSAE) framework. SSAE avoids the OSZs limitations previously mentioned:

- *Task mapping*. Not limited to continuous regions. Once defined the PEs to execute the secure application, the SDN Controller defines dedicated paths for the communicating tasks.
- *Communication of non-secure applications*. Not affected by the secure applications, once they use the PS subnet to communicate.

Figure 3 differentiates SSAE from OSZs (Figure 2). Tasks from application 2 communicate using SDN subnet 0 (blue). When task A3 transmits a packet to B3, it uses the PS subnet, not interfering in application 2 (blue tiles) traffic because

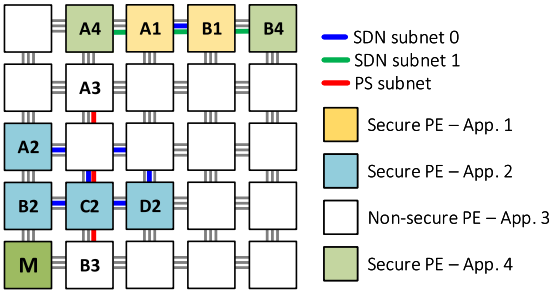

FIGURE 3. Example of three discontinuous SZs using SDN-based management, and one non-secure flow (A3 ⇒ B3).

subnets are disjoint. PEs of the top right corner shows a different scenario, two secure application: 1 (yellow tiles), and 4 (green tiles). Both applications are mapped close to each other. While tasks A1 and B1 use SDN subnet 0, tasks A4 and B4 use subnet 1. Both communications take place without interference, with minimal path length, and maximum throughput.

Figure 4 presents the SSAE protocol, detailing the interaction between the MCSoC components. Left labels, (a) to (e) in Figure 4, correspond to the next subsections.

A. INITIALIZATION

The initialization phase executes once, at system startup. In this phase, the GM authenticates the *INJ* to ensure that requests made by *INJ* come from reliable entities. Both *INJ* and GM execute a mutual authentication algorithm based on the Elliptic Curve Diffie–Hellman Key Exchange (ECDH) protocol. At the end of this phase, the *INJ* and the MCSoC share a common session key – K_e. This key is used to guarantee the integrity of the tasks' object code during the secure task loading (Section V-E).

The GM initializes the Controller with an *ML* list, which contains the set of M PE addresses that the Controller should accept path requisitions. The Controller accepts requests for SDN paths *iff* the M PE address is in *ML*, avoiding unauthorized path requests. The Controller rejects any packet whose address is not in *ML* [25].

B. APPLICATION ADMISSION

The *INJ* peripheral is responsible for requesting the deploy of new applications in the MCSoC. To deploy a new application, the *INJ* transmits a "new application request" message to the GM, with the application task graph. The GM authenticates the request with K_e, and if successful, it selects the cluster to receive the application using as criteria the cluster utilization, temperature or computing capacity (e.g., BIG.little).

The GM sends an "application request" message to the M PE of the selected cluster (GM can also be chosen). The application request contains the application task graph (for mapping), and the information if it is secure or not.

C. SDN-BASED SECURE TASK MAPPING

When an M PE receives an "application request" message, it verifies if the application has security constraints. If it is a standard application (without security or QoS constraints), M executes task mapping by grouping tasks as close as possible to reduce the hop count. Otherwise, M performs the SDN-based Secure Task Mapping (SSTM) algorithm. SSTM has two constraints:

- spatial isolation at the computation level: secure tasks can only share a PE with tasks belonging to the same application;
- spatial isolation at the communication level: map tasks in regions where the SDN subnets utilization is low to increase the probability of all applications' tasks receiving a dedicated connection.

Before starting SSTM, the M PE requests to the Controller the MPN utilization. The Controller verifies if the M address is in *ML*, and replies with the set $U_{SDN} = \{SRU_{1,0}, SRU_{1,1}, SRU_{2,0}, SRU_{p,s}\}$, where *SRU* is the SR utilization at PE p and subnet s. Each $SRU_{p,s}$ is a five-bit vector $SRU_{p,s} = \{E_k, W_k, N_k, S_k, L_k\}$, where E_k, W_k, N_k, S_k, L_k corresponds to the status (used or free) of an SR input port (East, West, North, South, Local).

The next step corresponds to the SSTM algorithm execution, presented in Algorithm 1. The algorithm has as inputs:

i) the U_{SDN} set;
ii) the application task set:

$$A_t = \{(at_1, ctp_1), (at_2, ctp_2), \ldots, (at_n, ctp_n)\} \quad (1)$$

where: at_{ID} is a unique task identifier of the application to be mapped; and ctp_{ID} is the set of tasks with which task at_{ID} communicates, named *communicating task pairs*;

iii) the current cluster task mapping, MAP_{PE}, with a set of tuples $\{t_k, SP_{x,y}, init\}$, where t_k is a task identifier, $SP_{x,y}$ is the location where t_k is mapped, and *init* signalizes if t_k is an initial task[1] of a given application. Note that more than one task is allowed to be mapped in the same $SP_{x,y}$ (multi-task mapping).

The algorithm executes three main steps:

1) Task mapping - starts by selecting an initial PE, mapping the set A_t around this PE.
2) Bounding box computation (BB) - after mapping A_t, the algorithm computes a rectangular BB, including all mapped tasks.
3) BB utilization - SDN subnets utilization computation at each PE within the BB.

The selected mapping is the one with the smallest SDN subnets utilization. The SSTM output is the application mapping, A_{MAP}, with a set of tuples $\{at_k, SP_{x,y}, init\}$.

The presentation of Algorithm 1 contains four blocks. The first block (a) addresses variables initialization:

i) EX_{INIT}: list of initial PEs already addressed by the algorithm.
ii) $BB_{U\_RATE\_BEST}$: smallest BB utilization rate achieved during the algorithm execution.

[1]Initial task: the task of an application task graph with no dependencies to other tasks, i.e., it only sends data as "Task A" in Figure 1(d).

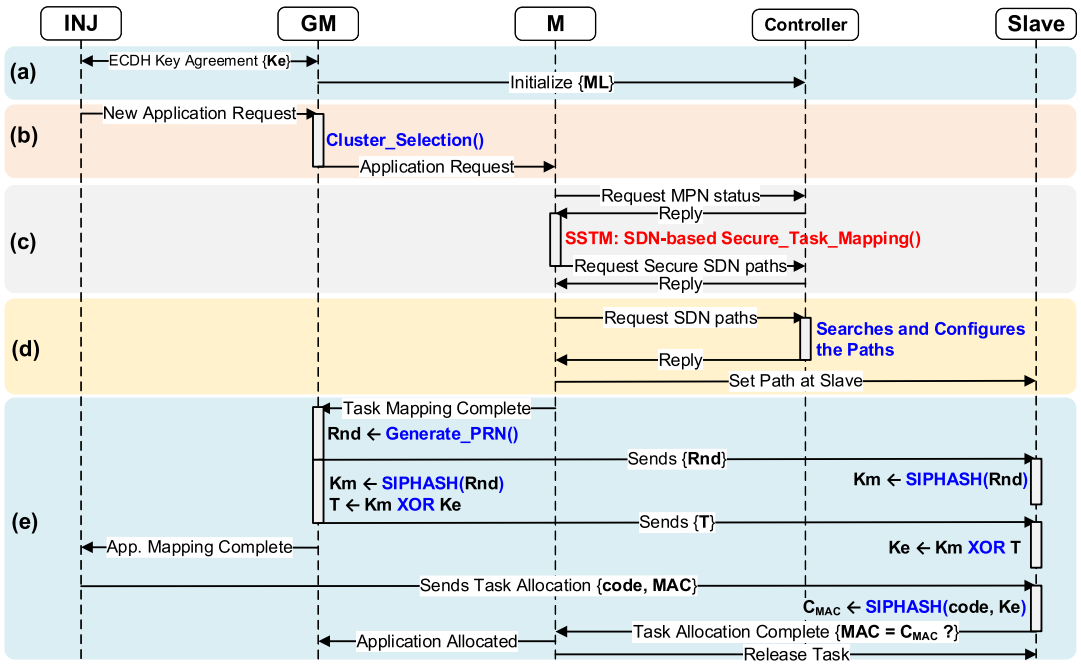

FIGURE 4. SSAE Protocol. (a) *INJ* authentication. (b) Application cluster selection. (c) SDN-based secure task mapping. (d) Secure SDN path configuration. (e) Secure application task admission.

Algorithm 1 SDN-Based Secure Task Mapping (SSTM)

```
Input: U_SDN, A_t, MAP_PE
Ouput: A_MAPP
1.  EX_INIT ← Ø
2.  BB_U_RATE_BEST ← ∞
3.  A_MAPP ← Ø                                          (a)
4.  N ← MAX_PROCESSORS
5.  Do
6.      PE_INIT ← select_initial_PE(EX_INIT, MAP_PE)
7.      If PE_INIT == NULL then
8.          Break   /*System is saturated*/             (b)
9.      End If
10.     EX_INIT ←EX_INIT U {PE_INIT} /*Adds PE_INIT to the
        EX_INIT set*/
11.     A_MAPP_TMP ← Ø
12.     For each task at_i of A_t do
13.         A_MAPP_TMP(at_i)
              ← diamond_mapping(PE_INIT, MAP_PE, at_i)
14.         If A_MAPP_TMP(at_i) == NULL then            (c)
15.             A_MAPP_TMP ← Ø
16.             Break   /*mapping not feasible*/
17.         End if
18.     End For
19.     If A_MAPP_TMP ≠ Ø
20.         A_BB ← compute_app_BB(A_MAPPTMP)
21.         BB_U ← compute_BB_SDN_utilization
            (A_BB, U_SDN)                               (d)
22.         BB_U_RATE ← BB_U/ get_PE_number(A_BB)
23.         If BB_U_RATE < BB_U_RATE_BESTthen
24.             BB_U_RATE_BEST ← BB_U_RATE
25.             A_MAPP ← A_MAPP_TMP
26.         End If
27.     End For
28.     N ← N − 1
29. While N > 0
30. Return A_MAPP
```

iii) A_{MAPP}: application mapping.

iv) N: number of attempts to execute the algorithm. In this example $N = MAX\_PROCESSORS$, which is the number of PEs in the cluster, being the worst-case in terms of iterations.

Next, block (*b*), the algorithm starts the do-while loop (lines 5-29) responsible for computing the application mapping – A_{MAP}. Line 6 invokes the function *select_initial_PE()* to select an initial PE – PE_{init}. The function walks over each free PE, i.e., a PE not running any task, looking for a candidate PE_{init}. The selected PE_{init} is the one with the highest Manhattan distance from another PE_{init} closer to it and not belonging to EX_{INIT} ($PE_{init} \notin EX_{INIT}$).

If there is no available PE_{init} (lines 7-9), the algorithm ends, returning A_{MAP} (it may be empty in a system with all SPs executing tasks). At line 10, the algorithm includes PE_{init} in the EX_{INIT} list to avoid its use in the next iterations.

Block (*c*) in Algorithm 1 maps the application's tasks. The function *diamond_mapping()* computes the mapping by assigning the initial task to PE_{init}, and the other tasks around it, according to the communication edges (*ctp* in Equation (1)) in the application task graph (the number of tasks per SP is a design parameter). This function returns the SP address for at_i. To meet the first constraint, spatial isolation at the computation level, tasks can be only assigned to SPs withs tasks of the application being mapped. If a given task cannot be mapped (lines 14-16), the mapping is not feasible since the *diamond_mapping()* uses a predetermined search radius to avoid applications' fragmentation [28].

The last block (*d*) addresses the second constraint, spatial isolation at the communication level. If the mapping succeeds, the algorithm calls the function *compute_app_BB()*

with the temporary mapping result (A_MAPP_TMP), returning in A_{BB} a data structure with the bounding box coordinates enclosing the application's tasks.

Line 21 computes the SDN utilization in A_{BB}, BB_U. This number is the summation of all used SDN subnets inside the BB. As different mappings may have a different number of SPs in BB, line 22 normalizes the utilization by dividing A_{BB} by the number of SPs inside the BB, resulting in BB_{U_RATE}.

Lines 23 to 26 selects the mapping with the smallest BB_{U_RATE}. The algorithm ends after executing "N" iterations, or when there is no available PE_{init}.

If A_{MAPP} returns empty at the end of the execution, it is not possible to admit the application due to the lack of computational resources.

D. SDN CONNECTIONS ESTABLISHMENT

As Figure 1(d) shows, an application contains a set of communicating task pairs (*ctp* in Equation (1)), with a sender *S* and a receiver *R* task. After finishing the task mapping, the M PE sends the list of all *ctps* to the Controller, requesting a dedicated SDN path to them. The Controller handles such a message by running its search path heuristic.

The exploitation of search path heuristics is out of this work scope. We adopt the Hadlock's algorithm due to its benefits [29], including a computational complexity of $O(X.Y)$, where X and Y are the MCSoC dimensions. Hadlock's algorithm always finds the shortest path between S and R, if it exists.

After finding a path, the Controller physically configures the SR routers in the path by sending a configuration message command through the PS subnet. The NI of the target PE receives the configuration message and configures the respective SR router, assigning a given input port to an output port. Ruaro *et al.* [25] describe the Secure SDN Framework for configuring the paths, detailing each step required to ensure the secure SR configuration.

This process finishes with the Controller sending to M the connection establishment status. If a path cannot be established, it is not possible to admit the application due to the lack of communication resources.

E. SECURE TASK LOADING

If the mapping and SDN connection succeeds, the protocol moves to the last phase. This phase securely loads the tasks into the selected SPs.

This step starts with GM transmitting K_e, without exposing it, to the SPs that will receive the tasks. The GM first sends a random number to these SPs, used to compute a temporary key, K_m, generated by the SIPHASH algorithm. This key encrypts K_e resulting in the T key. The GM transmits T to the SPs, which recover K_e.

The GM sends the A_{MAPP} result to *INJ*, which transmits the tasks' object code appended with a MAC, created using SIPHASH between the object code and K_e.

TABLE 2. Computational complexity evaluation for each phase of SSAE (cc: clock cycles).

SSAE Phase	Compl.	where n is ...	Avg. Overhead
(a) Initialization	$\mathcal{O}(\sqrt{n})$	Elliptic curve prime size	2.2×10^9 cc
(b) App. Admission	$\mathcal{O}(n)$	Cluster number	1,000 cc
(c) SSTM	$\mathcal{O}(n^3)$	Cluster size (PEs)	65,168 cc
(d) SDN Path	$\mathcal{O}(n^2)$	$Ctps$ number	48,006 cc
(e) Task Loading	$\mathcal{O}(n)$	App. tasks number	33,958 cc

When an SP receives the task allocation message, it stores the task's code into its local memory. The SP OS computes a MAC (C_{MAC}) by performing a SIPHASH using as input the task's code and K_e. The OS compares C_{MAC} with the received MAC. Note that even with the task loaded in the memory, the OS does not schedule it before receiving a release message.

All SPs notifies the status MAC comparison ("task allocation complete" message) to the cluster M. If all tasks were correctly received, M notifies all tasks to start their execution (release message). M also notifies the application admission result to the GM.

VI. EXPERIMENTAL RESULTS

The experimental setup adopts the *Memphis* MCSoC architecture [23]. The hardware is described in SystemC-RTL for fast simulation and VHDL for synthesis. The software is described in C language (mips-gcc cross-compiler, version 4.1.1, optimization O2).

The experiments aim to validate the SSAE framework and compare the performance and design cost with OSZ.

A. SSAE COMPLEXITY EVALUATION

Table 2 evaluates SSAE in terms of computational complexity required for each protocol phase, presented in Figure 4. The last column presents the *average latency* for each phase, measured from the experiments presented in the next Subsection.

Phase (a) – the initialization corresponds to the execution of the ECDH key agreement protocol to generate K_e between *INJ* and GM. As shown in the last column of Table 2, the initialization takes a higher amount of time than the other phases. The ECDH key agreement dominates such time. It is important to note that despite costly, this phase does not affect the application admission time, since it is only executed once, at system startup.

Phase (b) – the cost corresponds to the execution of the *Cluster_Selection()* algorithm. This algorithm has a small cost, being a function of the clusters' number. The cluster selection uses as cost function the cluster's utilization, evaluating the amount of running tasks at each cluster, selecting the one with the lower usage.

Phase (c) – the SSTM algorithm cost is a function of the following parameters:

- p: cluster size;

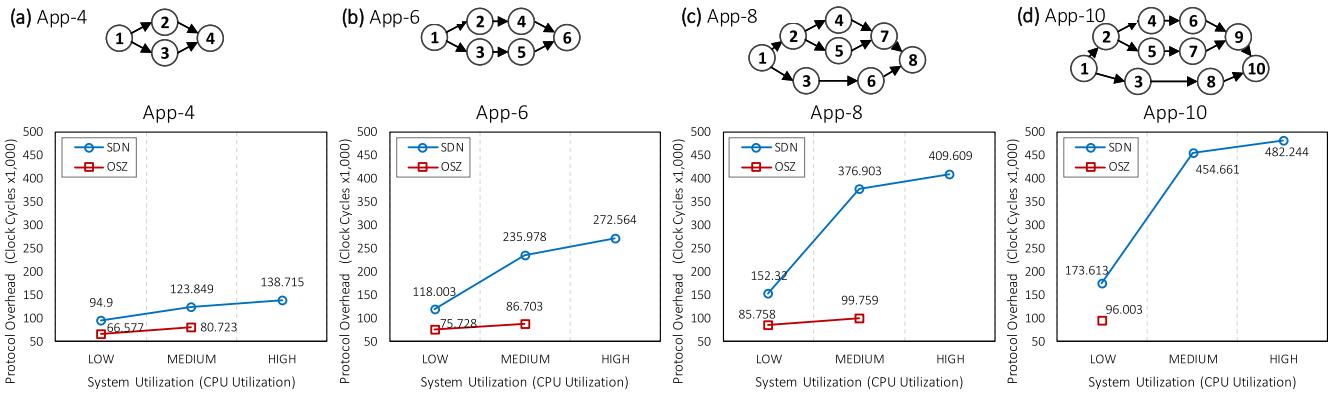

FIGURE 5. Performance comparision of SDN and OSZ.

- a: number of application number running in the cluster;
- b: bounding box size.

The algorithm executes functions with the following complexities:

- select_initial PE: $\theta(p.a)$
- diamond_mapping: $\theta(p.log(p))$
- compute_app_BB: $\theta(b)$
- compute_BB_SDN_utilization: $\theta(b)$

Therefore, considering the loops of SSTM, it presents the complexity $\theta(p.[p.a + a.p.log(p) + b + b])$, which leads to a worst-case complexity of $\mathcal{O}(p^3)$.

Phase (d) – SDN path establishment, has the complexity according to the Hadlock's algorithm $\mathcal{O}(n)$, where n is the number of PEs in the cluster. However, note that the Hadlock's algorithm is executed for each communicating task pair (*ctp*).

As the number of *ctp*s can be higher than the cluster size (in a pessimist case, with an application having several *ctp*s), the complexity is $\theta(p.n)$, where p is the PE number and n is the *ctp* number. Therefore, the worst-case complexity results in a quadratic function of $\mathcal{O}(n^2)$, with n being the number of *ctps*.

Phase (e) – Task's loading, has a linear complexity $\mathcal{O}(n)$, with n being the application tasks' number.

B. PERFORMANCE COMPARISON

The performance comparison adopts two MCSoC implementations: secure zone model – **OSZ** (Section IV), SSAE framework – **SSAE** (Section V). The comparison considers two performance figures:

- **PO**: protocol latency overhead (in clock cycles – *cc*) is the time that the target secure application (T_A) has to wait to start its execution. PO is the interval from the moment *INJ* requests T_A to GM, until GM reply to *INJ* the end of the T_A's allocation - phases (b) to (e) in Figure 4 and Table 2.
- **SU**: System CPU utilization in terms of simultaneously running tasks. PU is given according to Equation 2:

$$SU = \frac{R}{(S \times M)} \tag{2}$$

where R is the number of running tasks, S is the number of slave PEs of the cluster, and M is the maximum number of tasks allowed to execute at each slave PE.

The experimental setup consists of four T_As, with its communicating graphs depicted in Figure 5(a–d), varying the tasks' number for each application (4, 6, 8, 10). The tasks' number directly influences the PO.

We adopted a 5 × 5 cluster, with each PE supporting the execution of one task, corresponding to a cluster with 24 SPs ($S = 24$ and $M = 1$ in Equation (2)).

Experiments consider mapping T_A in an MCSoC with only secure applications mapped on it, with a preloaded SU. The preloaded SU are: *Low*, 5 tasks pre-mapped corresponding to a SU=20%; *Medium*, 12 tasks pre-mapped corresponding to a SU=50%; *High*, the number of available PEs equals to the number of T_A's tasks. Such initial SU pushes each approach to increasingly saturated scenarios, where the probability of finding resources to a secure application execution tends to decreases.

Figure 5 presents four graphs. At each graph, the x-axis corresponds to System CPU utilization, **SU**, which may be low, medium and high. The y-axis corresponds to the protocol latency, **PO**, for each SU. This figure presents 24 experiments, varying the T_A size and the preloaded SU for SSAE and OSZ. The missing points in the plots correspond to scenarios where the approach failed to admit T_A.

The PO evaluation shows that OSZ is faster (−49% on average) than SSAE to admit and execute a secure application. Both approaches execute task mapping (software job). The OSZ advantage is the smallest search area than SSAE. The OSZ mapping search area is the secure region area, while SSAE searches in the entire cluster. Also, the SSAE approach requires an SDN path establishment phase (software job), for each *ctp*, with a quadratic complexity (Table 2). Thus, the higher PO cost for the SSAE approach

is expected due to the higher number of jobs executed in software.

The SU evaluation shows that OSZ fails to admit applications with a *high* preloaded SU. The probability for OSZ to find a continuous shape in such scenarios reduces due to the fragmentation of the previous applications' mapping. A solution to solve this issue is to use task migration to create a continuous region. However, task migration only is feasible if the migrated tasks are non-secure and not have real-time constraints (i.e., best-effort tasks). Additionally, task migration contributes to increasing PO since it imposes an overhead around 200,000 *cc* to each migration, according to the average state-of-the-art techniques [30].

The application tasks' number also affects SU. Figure 5(d) addresses an application with ten tasks. In this experiment, OSZ fails to admit the secure application even for the *medium* preloaded SU. An application with a larger size has more tasks to be mapped, increasing the size required by the secure zone and reducing the chances to find it.

The advantage of OSZ is the shorter time to admit the application. Consider a system running at 500 MHz, with a PO = 500.000 *cc* (SDN worst-case in Figure 5). The resulting latency to admit the application, in this case, is 1 ms. As this overhead occurs once, at the application admission, it is in practice imperceptible to the end-user. Thus, the main advantage of SDN is to explore better the available system resources (higher SU), avoiding the restrictions of continuous regions.

After evaluating the latency to admit applications, the following results evaluate the applications' execution time. The evaluation considers two benchmarks: Dynamic Time Warping (DTW) – a pattern recognition application, and MPEG4 – an audio and video decoder. Figure 6(a) and Figure 6(b) show the communicating task graph of DTW and MPEG4 benchmarks, respectively. The graph in Figure 6(c) evaluates the execution time (x-axis) versus the iteration latency (y-axis). The DTW recognizes 50 patterns (50 iterations), and the MPEG4 decodes 100 frames (100 iterations). Each curve in the graph corresponds to one simulation, with the *INJ* peripheral requesting the application admission at 1 ms.

Table 3 presents the applications' start time and the applications' execution time. Considering Table 3 and Figure 6, we observe:

- Protocol latency overhead (**PO**). Vertical bars show the start time of both applications. As expected, SSAE has a higher PO compared to OSZ, endorsing the experiment presented in Figure 5. The higher PO seen in the DTW-SDN comes from the larger number of connections to set up (16 connections) than the MPEG4-SDN (7 connections).
- Execution time (**ET**). The SSAE execution time tends to be smaller than OSZ due to the reduced iteration latency. The iteration latency reduction comes from the communication using CS, where packets spent just one clock cycle per router. SSAE is 1.92% and 2.55% faster than OSZ for DTW and MPEG4, respectively.
- Total execution time (**TET** = PO + ET). The PO only affects the starting of applications' execution. With SSAE having a smaller iteration latency than OSZ, its TET tends to be smaller which helps to amortize the PO delay, as observed in the MPEG4 benchmark. For the DTW benchmark, the OSZ is faster than SDN due to the smaller number of iterations.

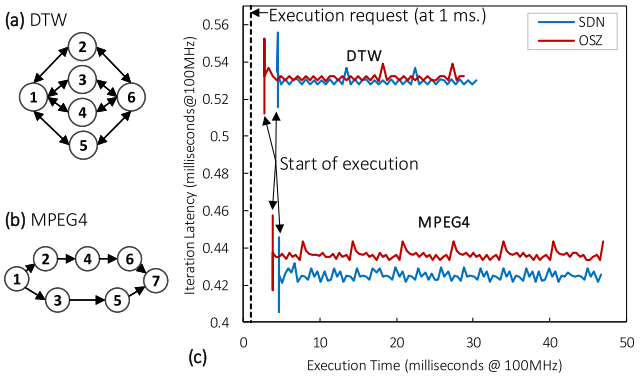

FIGURE 6. (a) DTW communicating task graph. (b) MPEG4 communicating task graph. (c) Performance comparison of MPEG4 and DTW running at SSAE (SDN) and OSZ systems.

TABLE 3. Evaluation of the start and execution time for OSZ and SDN approaches.

benchmark	n# iterations	start time (ms)		execution time (ms)	
		OSZ	SDN	OSZ	SDN
DTW	50	2.3	4.0	26.1	25.6
MPEG4	100	2.8	3.6	43.2	42.1

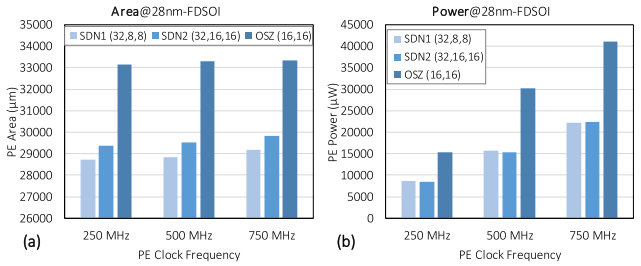

FIGURE 7. Area and power comparison.

Summarizing, even with a larger PO, the SSAE proposal does not impact the applications' performance.

C. DESIGN COST

The next experiment evaluates the physical design cost for SSAE and OSZ. Our goal is to show the area and estimated power (provided after the physical synthesis) required for each PE design in both approaches. We use the Cadence Genus tool, with a 28nm FD-SOI technology, varying the frequency constraint: 250MHz, 500MHz, and 750MHz. Figure 7 shows the achieved results for area (a) and power (b).

The OSZ PE has two 16-bit PS subnets and a dedicated NoC for rerouting packets blocked due to a secure zone [26]. The SDN PE has three subnets, one 32-bit PS and two SDNs. The SDN subnets are modeled in two versions: **SDN1**: 8-bit flits; and **SDN2**: 16-bit flits.

The power required by OSZ was, on average, 85.4% and 87.2% higher than SDN1 and SDN2, respectively. The main reason for explaining such difference comes from the straightforward design of SDN routers, which have a 1-flit buffer depth and no routing and arbitration logic. The OSZ approach uses PS routers, with 8-flit depth buffers, requiring arbitration and routing at each hop in the path.

The area required by OSZ was, on average, 15% and 12.4% higher than SDN1 and SDN2, respectively. It is recommended to increase the number of SDN subnets in larger systems, to increase the path diversity and ensure that paths are found in congested scenarios [31]. Thus, the SDN area may be larger than the OSZ increasing the number of SDN subnets.

Such results highlight an SDN advantage over OSZ: the reduced power consumption. Once a path established, communication occurs through circuit switching, resulting in a smaller switching activity.

VII. CONCLUSION AND FUTURE WORK

This work proposed a framework to admit and execute secure application into a many-core based on SDN management (SSAE). The main novelty of SSAE is to provide security to applications by dynamically establishing circuit switching (CS) for its communicating task pairs using the SDN management paradigm. This feature offers communication integrity, leading to data transmission without the overhead of encryption, arbitration, and routing required in PS NoCs.

The adoption of a session key, K_e, to the SIPHASH MAC algorithm prevents hijacking and spoofing attacks when transferring the task's object code. Processors verify the MAC attached to the object-code before starting the execution, preventing malicious code insertion, and guarantee that only external authenticated entities deploy secure applications in the MCSoC.

Attacks to the availability, like DoS attacks, are prevented due to the resources' isolation. At the NoC level, CS's adoption avoids attacks that explore congestion or starvation at the NoC links or routers. Avoidance of attacks over computation resources results from the mapping decision that guarantees that processors execute tasks belonging to the same application. Timing attacks are prevented since no time inferences can be taken from packets in CS channels since no external packet can share links used by CS.

Comparing SSAE to a state-of-the-art secure zone approach (OSZ), SSAE induces a higher application admission latency due to the SDN execution for finding paths between communicating tasks. Nonetheless, such latency is negligible at the end-user perspective since it remains below 1 ms (@800MHz) according to the presented experiments.

On the other side, SSAE revealed two important findings. The first one is the better system utilization, specifically for congested scenarios, where OSZ cannot find continuous regions to map tasks due to the system fragmentation. The second one is the lower power consumption resulted from the reduced switching activity when using CS.

Future work includes developing attack campaigns aiming to evaluate scenarios with threats in both computational and communication levels.

REFERENCES

[1] ARM. (Nov. 2008). *ARM Security Technology Building a Secure System Using TrustZone Technology*. [Online]. Available: http://infocenter.arm.com

[2] H. Isakovic and A. Wasicek, "Secure channels in an integrated MPSoC architecture," in *Proc. IECON-39th Annu. Conf. IEEE Ind. Electron. Soc.*, Nov. 2013, pp. 4488–4493.

[3] R. Fernandes, C. Marcon, R. Cataldo, J. Silveira, G. Sigl, and J. Sepulveda, "A security aware routing approach for NoC-based MPSoCs," in *Proc. 29th Symp. Integr. Circuits Syst. Design (SBCCI)*, Aug. 2016, pp. 1–6.

[4] J. Sepulveda, D. Flórez, V. Immler, G. Gogniat, and G. Sigl, "Efficient security zones implementation through hierarchical group key management at NoC-based MPSoCs," *Microprocessors Microsyst.*, vol. 50, pp. 164–174, May 2017.

[5] M. M. Real, P. Wehner, V. Lapotre, D. Göhringer, and G. Gogniat, "Application deployment strategies for spatial isolation on many-core accelerators," *ACM Trans. Embedded Comput. Syst.*, vol. 17, no. 2, pp. 1–31, Apr. 2018.

[6] 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.

[7] M. Ruaro, N. Velloso, A. Jantsch, and F. G. Moraes, "Distributed SDN architecture for NoC-based many-core SoCs," in *Proc. 13th IEEE/ACM Int. Symp. Netw.-Chip*, 2019, p. 8.

[8] A. Kostrzewa, S. Tobuschat, and R. Ernst, "Self-aware network-on-chip control in real-time systems," *IEEE Des. Test*, vol. 35, no. 5, pp. 19–27, Oct. 2018.

[9] K. Berestizshevsky, G. Even, Y. Fais, and J. Ostrometzky, "SDNoC: Software defined network on a chip," *Microprocessors Microsyst.*, vol. 50, pp. 138–153, May 2017.

[10] J. Sepulveda, F. Willgerodt, and M. Pehl, "SEPUFSoC: Using PUFs for memory integrity and authentication in multi-processors system-on-chip," in *Proc. Great Lakes Symp. VLSI (GLSVLSI)*, 2018, pp. 39–44.

[11] J. Rajesh, D. M. Ancajas, K. Chakraborty, and S. Roy, "Runtime detection of a bandwidth denial attack from a rogue network-on-chip," in *Proc. NOCS*, 2015, pp. 8:1–8:8.

[12] M. A. Kinsy, S. Khadka, M. Isakov, and A. Farrukh, "Hermes: Secure heterogeneous multicore architecture design," in *Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust (HOST)*, May 2017, pp. 14–20.

[13] B. Oliveira, R. Reusch, H. Medina, and F. Moraes, "Evaluating the cost to cipher the NoC communication," in *Proc. IEEE 9th Latin Amer. Symp. Circuits Syst. (LASCAS)*, Feb. 2018, pp. 1–4.

[14] H. M. G. Wassel, Y. Gao, J. K. Oberg, T. Huffmire, R. Kastner, F. T. Chong, and T. Sherwood, "Networks on chip with provable security properties," *IEEE Micro*, vol. 34, no. 3, pp. 57–68, May 2014.

[15] T. Boraten and A. K. Kodi, "Packet security with path sensitization for NoCs," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2016, pp. 1136–1139.

[16] C. Reinbrecht, A. Susin, L. Bossuet, G. Sigl, and J. Sepúlveda, "Timing attack on NoC-based systems: Prime+probe attack and NoC-based protection," *Microprocessors Microsyst.*, vol. 52, pp. 556–565, Jul. 2017.

[17] Y. J. Yoon, N. Concer, M. Petracca, and L. P. Carloni, "Virtual channels and multiple physical networks: Two alternatives to improve NoC performance," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 32, no. 12, pp. 1906–1919, Dec. 2013.

[18] S. Liu, A. Jantsch, and Z. Lu, "MultiCS: Circuit switched NoC with multiple sub-networks and sub-channels," *J. Syst. Archit.*, vol. 61, no. 9, pp. 423–434, Oct. 2015.
[19] F. Bannour, S. Souihi, and A. Mellouk, "Distributed SDN control: Survey, taxonomy, and challenges," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 1, pp. 333–354, 1st Quart., 2018.
[20] S. Ellinidou, G. Sharma, T. Rigas, T. Vanspouwen, O. Markowitch, and J.-M. Dricot, "SSPSoC: A secure SDN-based protocol over MPSoC," *Secur. Commun. Netw.*, vol. 2019, pp. 1–11, Mar. 2019.
[21] A. Scionti, S. Mazumdar, and A. Portero, "Towards a scalable software defined network-on-chip for next generation cloud," *Sensors*, vol. 18, no. 7, pp. 1–24, 2018.
[22] I.-D. Salvador, S.-A. Remberto, M. Brox, and M. A. Ortiz, "Software defined network controller: A neat solution administration for reconfigurable multi-core NoC," in *Proc. Int. Conf. ReConFigurable Comput. FPGAs (ReConFig)*, Dec. 2017, pp. 1–4.
[23] M. Ruaro, L. Caimi, V. Fochi, and F. Moraes, "Memphis: A framework for heterogeneous many-core SoCs generation and validation," *Des. Autom. Embedded Syst.*, vol. 23, no. 3, pp. 103–122, 2019.
[24] T. Krishna, C.-H.-O. Chen, W.-C. Kwon, and L.-S. Peh, "Smart: Single-cycle multihop traversals over a shared network on chip," *IEEE Micro*, vol. 34, no. 3, pp. 43–56, May 2014.
[25] 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.
[26] 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.
[27] T. Maqsood, N. Tziritas, T. Loukopoulos, A. S. Madani, U. S. Khan, C.-Z. Xu, and Y. A. Zomaya, "Energy and communication aware task mapping for MPSoCs," *J. Parallel Distrib. Comput.*, vol. 121, pp. 71–89, Nov. 2018.
[28] C. A. Bonney, "Fault tolerant task mapping in many-core systems," Ph.D. dissertation, Dept. Electron. Eng., Univ. York, York, U.K., 2016.
[29] F. Hadlock, "A shortest path algorithm for grid graphs," *Networks*, vol. 7, no. 4, pp. 323–334, 1977.
[30] M. Ruaro and F. G. Moraes, "Demystifying the cost of task migration in distributed memory many-core systems," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2017, pp. 1–4.
[31] M. Ruaro, H. M. Medina, and F. G. Moraes, "SDN-based circuit-switching for many-cores," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2017, pp. 385–390.

MARCELO RUARO was born in Três de Maio, Brazil, in 1988. He received the M.Sc. and Ph.D. degrees in computer science from the Pontifical Catholic University of Rio Grande do Sul (PUCRS), Porto Alegre, Brazil, in 2014 and 2018, respectively. He is currently a Postdoctoral Researcher with PUCRS. He has eight years of research experience in the field of NoC and many-cores SoC architectures and two years of experience in the embedded system industry. His primary research interests include software-defined networking and security for many-core systems.

LUCIANO LORES CAIMI (Member, IEEE) received the M.Sc. degree in electrical engineering from the Federal University of Santa Catarina (UFSC), Florianópolis, Brazil, in 1998, and the Ph.D. degree in computer science from the Pontifical Catholic University of Rio Grande do Sul (PUCRS), Porto Alegre, Brazil, in 2019. He is currently an Adjunct Professor with the Federal University of Fronteira Sul (UFFS). His main research interests include multiprocessor systems on chip (MPSoC) and security for embedded systems.

FERNANDO GEHM MORAES (Senior Member, IEEE) received the degree in electrical engineering and the M.Sc. degree from the 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 the Pontifical Catholic University of Rio Grande do Sul (PUCRS), since 2002. He has authored or coauthored 38 peer-refereed journal articles in the field of VLSI design. His primary research interests include microelectronics, FPGAs, reconfigurable architectures, NoCs, and MPSoCs.