Security in MPSoCs: A NoC Firewall and an Evaluation Framework

Miltos D. Grammatikakis, Kyprianos Papadimitriou, Polydoros Petrakis, Antonis Papagrigoriou, George Kornaros, Ioannis Christoforakis, Othon Tomoutzoglou, George Tsamis, and Marcello Coppola

***Abstract*—In multiprocessor system-on-chip (MPSoC), a CPU can access physical resources, such as on-chip memory or I/O devices. Along with normal requests, malevolent ones, generated by malicious processes running in one or more CPUs, could occur. A protection mechanism is therefore required to prevent injection of malicious instructions or data across the system. We propose a self-contained Network-on-Chip (NoC) firewall at the network interface (NI) layer which, by checking the physical address against a set of rules, rejects untrusted CPU requests to the on-chip memory, thus protecting all legitimate processes running in a multicore SoC. To sustain high performance, we implement the firewall in hardware, with rule-checking performed at segment-level based on deny rules. Furthermore, to evaluate its impact, we develop a novel framework on top of gem5 simulation environment, coupling ARM technology and an instance of a commercial point-to-point interconnect from STMicroelectronics (STNoC). Simulation tests include scenarios in which legitimate and malicious processes, running in different CPUs, request access to shared memory. Our results indicate that a firewall implementation at the NI can have a positive effect on network performance by reducing both end-to-end network delay and power consumption. We also show that our coarse-grain firewall can prevent saturation of the on-chip network and performs better than fine-grain alternatives that perform rule checking at page-level. Simulation results are accompanied with field measurements performed on a Zedboard platform running Linux, whereas the NoC Firewall is implemented as a reconfigurable, memory-mapped device on top of AMBA AXI4 interconnect fabric.**

***Index Terms*—Deny rules, firewall, MPSoC, network-on-chip, segment-level security, Spidergon STNoC.**

I. Introduction

NOWADAYS, there is an increasing interest in solutions for trusted computing mainly driven by the economic consequences when failing to ensure security in embedded applications [1]. MPSoCs and related on-chip communication architectures for connecting together SoC components have been widely studied during the last decade, e.g., NoC [2] or bus [3], while security aspects have only recently attracted interest. The lack of proper and efficient isolation of program code and data among trusted and untrusted applications constitutes a major challenge in shaping a secure architectural solution without jeopardizing performance.

Within this context, we explore the effect of security, and show that in the presence of malicious processes a security mechanism can reduce NoC power and end-to-end transmission delay. In [4], we presented a hardware-based NoC firewall performing efficient rule-checking of memory requests at segment-level, and a scalable modeling infrastructure relying on ARM and Spidergon STNoC augmented with security features. This paper extends that publication with the following contributions:

1) extensive experiments with multiple legitimate and malicious processes requesting access to memory;
2) a quantitative analysis of the effect of security on network delay and power consumption;
3) studying the extent to which a security mechanism can prevent NoC from entering a saturation phase;
4) performance comparison of our coarse-grain versus a fine-grain approach;
5) an implementation of the firewall on a field programmable gate array (Zynq-7020 FPGA) with GNU/Linux system driver, along with performance and area results.

The remaining paper is organized as follows. Section II overviews related work on hardware security. Section III describes the components and functionality of the proposed hardware NoC firewall. In Section IV, we define the threat model and type of traffic we generated to test our approach. Section V discusses the experimental framework. Section VI covers the different scenarios that examine the effect of the security mechanism when both legitimate and malicious requests occur. Section VII discusses our results for packet delivery time and NoC power consumption. In that section, we also compare the performance of our HW approach against a SW solution for reference purposes. In Section VIII, we contrast the performance of our coarse-grain approach against a fine-grain solution. Section IX examines an FPGA-based implementation under GNU/Linux in terms of performance and area costs. Finally, Section X summarizes this paper and outlines future work.

II. Background and Motivation

NoC firewall mechanisms provide domain protection by controlling memory access rights through hardware

Manuscript received July 10, 2014; revised November 18, 2014 and March 14, 2015; accepted May 5, 2015. This work was supported by EU FP7-ICT: TRESCCA under Grant 318036 and DREAMS under Grant 610640. This paper was recommended by Associate Editor O. Sinanoglu.

M. D. Grammatikakis, K. Papadimitriou, P. Petrakis, A. Papagrigoriou, G. Kornaros, I. Christoforakis, O. Tomoutzoglou, and G. Tsamis are with the Technological Educational Institute of Crete, Heraklion 71004, Greece (e-mail: mdgramma@cs.teicrete.gr).

M. Coppola is with STMicroelectronics, Grenoble, France.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2015.2448684

multicompartment isolation [5]–[8], therefore ensuring a well-defined end-user service agreement and billing model. The main purpose of memory protection is to prevent a process from accessing shared memory that has not been allocated to it. This prevents a malicious process (or a bug within a process) from affecting other processes, or the OS itself, issuing a segmentation fault or storage violation exception to the offending process which generally causes abnormal termination (killing the process).

Memory protection in multiprocessor operating systems includes high-level security techniques, such as address space layout randomization and executable space protection. In most previous memory protection schemes, fine-grain page-level security is proposed. Thus, decisions are made on whether to accept or reject a specific request based on rules hidden within a page-level memory descriptor or via an independent memory unit. Unlike fine-grain page-level protection, we consider coarse-grain, segment-level protection based on the physical address of the NoC transaction request. Our approach not only supports isolation, but also attempts to harmonize the system security infrastructure by implementing the NoC firewall module as a new distributed platform service at the network interface (NI) layer. These low-level HW/SW platform services enable advanced features in modern MPSoC applications.

Unlike the hardware memory protection module of Porquet *et al*. [7] and Wiggings *et al*. [8] considered *page-level* security, the proposed NoC firewall relies on segment-level protection. Since rules are configured and used directly at the NI, the proposed scheme has relatively less area overhead and latency. Our NoC firewall is integrated in the NI, similar to [5] and [6]. However, while [5], [6] use dedicated virtual channels to pass specialized rule-checking information processor and thread identifiers, we consider generic segment-level rules based on the NoC transaction's physical address bits (excluding offset). In addition, while [5], [6] compare different initiator and target implementations of the memory protection module in terms of area and power cost, we concentrate only on the initiator side, aiming at examining the processor requests rather than the memory accesses.

III. Hardware NoC Firewall

We propose a lightweight, nonintrusive security module composed of a hardware IP (and corresponding driver components) providing isolation in integrated NoC-based MPSoC solutions. The module is placed between the network-on-chip and a system component, e.g., CPU, memory controller or hardware accelerator, ideally at the NI. It acts as a firewall, adapting concepts from enterprise network firewalls to the on-chip communication architecture of MPSoC. Typically, there are multiple NoC firewalls distributed in the system, i.e., typically one for each initiator IP. Within each firewall component different access rules can be configured in order to control access to the protected memory regions.

Executable system-level specifications have been developed for a coarse-grain firewall solution implemented at the NI layer of a network-on-chip. We developed and validated both a bit-accurate, cycle-accurate SystemC virtual prototype and an equivalent gem5 model, which we time-annotated from register transfer level (RTL) simulation. Below, we examine the architecture and behavioral-level characteristics of the firewall. It operates at the NI layer by the following.

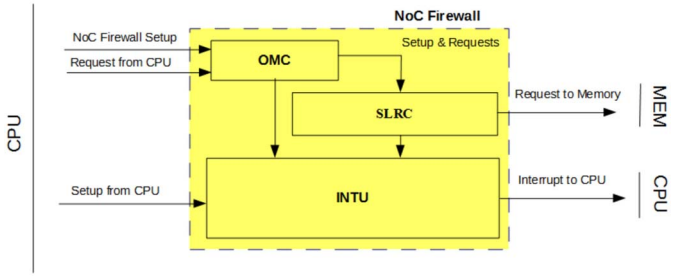


Fig. 1. Top-level NoC firewall module.

1) Statically or dynamically configuring rules for system or user processes that access segments of physical memory.
2) Filtering the underlying transactions and ensuring that security rules are obeyed at the NI of each initiator by providing fast and efficient access to rules; this implies that a compromised process cannot access data owned by a secure process, thus successfully subverting distributed denial-of-service (DDoS), eavesdropping, malware attacks on data and hardware/software vulnerabilities, including corrupt accesses from memory-mapped devices. These security threats are usually classified as medium probability events, but have very high to extremely high (DDoS) impact. The associated security risk is usually identified by the product of frequency and magnitude of loss (often seen as monetary value). It can be visualized nicely using heat maps [9].

Fig. 1 shows the top-level architecture.

1) The operating mode controller (OMC) accepts, decodes and dispatches NoC firewall commands.
2) The segment-level rule-checking (SLRC) module processes incoming memory accesses and configuration commands; notice that within the SLRC, a number of memory structures are responsible for implementing deny rules at segment-level (we assume an allow-by-default policy). SLRC also includes monitors (timers and event counters) to record NoC activity, and report performance and/or security issues to the interrupt unit (INTU).
3) The INTU accepts (in parallel) interrupt requests from the OMC block (e.g., invalid commands) and the SLRC block (resulting from rule-checking). It reports interrupt contexts to the CPU.

As shown in Fig. 2, a coarse-grain rule-checking approach is implemented in the SLRC unit. This policy controls accesses to the memory at segment-level, whereas segment size is variable. SLRC segments are implemented using register sets that define the (start, end) physical address range of each segment. Parallel search within these segments is implemented with comparators which compare the physical address of the incoming access after extracting lower order bits that correspond to the offset L; the value of L is 12 bits for the ARM v7 architecture.

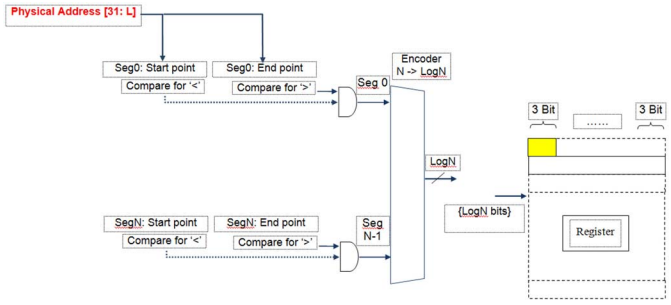


Fig. 2. Range search data structures using register-based parallel search.

This process involves checking whether an incoming physical address is in any preconfigured segment range. Since we have implemented a policy based on deny rules, if there is no match in the SLRC segment structure, then normal access is allowed to the physical address. Otherwise, if an incoming physical address falls within a particular segment address range, then the rule table needs to be accessed to examine the specific rule. In our models, each segment rule is three-bits long, specifying access control based on any combination of read, write, and execute privileges.

It is implementation-dependent (based on silicon cost) whether a context associative memory (CAM) structure is faster and/or less complex than the register-based comparators shown in Fig. 2. Based on RTL simulation and synthesis results, it appears that if the number of supported segments is small (below 128), then the proposed solution solely based on parallel comparisons using registers is adequate in terms of performance. This option is much more flexible, since it allows simpler firewall configuration. Other solutions, such as a parallel range search approach based on two ternary CAMs to encode segments using the longest common prefix concept provide nice asymptotics, but complicate significantly the network interface [10].

IV. Threat Model

We focus on NoC-based multicore SoC architectures in which a process invoked in any CPU can be potentially malicious. A representative work listing different type of threats is described in [11]. In our case, we are not concerned about physical attacks against the SoC, such as fault injection, side-channel analysis, and chip rewriting. Instead, the kind of attacks we seek to prevent are as follows.

1) Protection of sensitive software against logical adversaries, such as rogue processes, including viruses, trojans, or even corrupt hardware or software stacks of memory-mapped devices; these threats relate to data leakage, confidentiality, integrity, and availability issues. A typical scenario involves isolation of important trusted services, such as billing, authentication, and telecommunication, from an open OS (e.g., Android) executing uncertified third-party applications.
2) Denial-of-service or DDoS attacks, where malicious code injected by attackers prevents legitimate users from using a service, e.g., by saturating the NoC through massive unauthorized accesses to a memory controller. A link between DDoS and memory protection can be established with a network-based scenario by considering protection of the Linux kernel socket interface (e.g., when a server process forks to accept new network connections), or a distributed cache thrashing scenario which drains memory resources.

Later in Section VI, we present the traffic scenarios we deployed to study the behavior of the on-chip-network under different workloads, either legitimate, malicious, or mixed ones. To do this we generated real workload of different sizes as described in Section V. Although we have used real network traffic, measuring application execution time (e.g., by running PARSEC benchmarks on an MPSoC executing Linux) is beyond the scope of this paper. In fact, use of PARSEC benchmarks in full-system simulation is currently limited in gem5, since it is impossible to consider a realistic full-system scenario with an ARM multiprocessor architecture, Ruby memory system and Garnet interconnect [12]. Therefore, our focus is not on processor, but network performance, e.g., examining the time it takes to transfer a packet across the network when incorporating security at the NI of each CPU. More specifically, by using realistic traffic on the STNOC, we can control the amount of packets injected per time unit and examine variations in network delay and power consumption prior to network saturation.

V. Experimental Framework

To evaluate the effect of security (NoC firewall) on network latency and power consumption in shared memory MPSoCs, we created a framework connecting together CPUs and shared memory with a network-on-chip. Within this framework we measure delivery times starting from the time a packet reaches the NI queue, and power consumption at network-layer. We have also enhanced this framework by integrating the functionality of the NoC firewall described previously at the NI. This allows us to evaluate the effect of firewall activation/deactivation on the data delivery time and power consumption at the network-layer and below (network-, link-, and physical-layer of the MPSoC). A co-simulation approach (cycle-approximate for ARM Cortex-A9) allows us to consider realistic memory accesses. Although an extension to consider bare-metal applications is imminent, this is beyond the scope of this paper.

A. Building-Up the Framework

Our framework combines ARM v7 CPU technology and STNoC point-to-point interconnect. STNoC is a ring-based topology comprising three main building ingredients; NI that functions as access point to the interconnect, simple nonprogrammable router, and physical link. STNoC also supports programmable network services, such as interleaving, NI reprogramming, and quality-of-service (QoS), and can be customized according to given specifications to efficiently interconnect MPSoC components, i.e., CPUs, memories, and peripherals [13]. Fig. 3 shows an example topology for attaching different components to the STNoC. From the network point of view, each component is a node, while information (instructions and data) is transmitted across nodes in the form

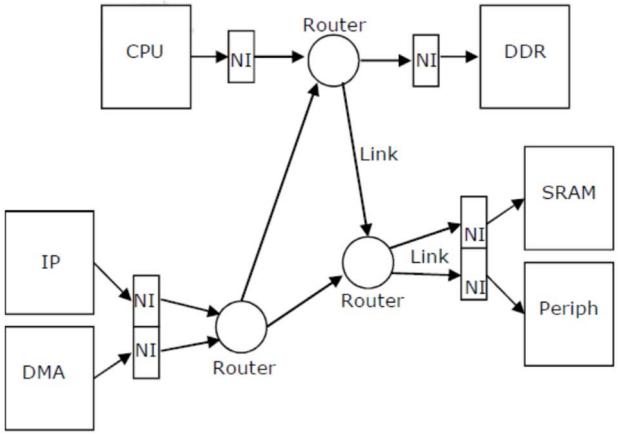


Fig. 3. Example of STNoC topology interconnecting the MPSoC components. STNoC consists of three basic building ingredients: NI, router, and link.

of packets. Prior to transmission, a packet is first broken down in smaller units called flits (flow control units) that are released into the network. A flit carries the maximum amount of data that can traverse the link per transaction. As a result, for a given packet size the amount of flits comprising each packet depends on the link-width. During the NoC configuration process, a designer uses tools (called INoC and MetaNoC) to customize the topology and tune different parameters, such as link width, packet size, number of flits in a packet, and QoS. For example, for an STNoC with 16 Bytes link-width, if the selected size of the packet is 64 Bytes, this is split up in four flits.

To conduct experiments we modeled the above system in the gem5 simulator [14], a platform widely used in computer architecture research, encompassing processor and system-level macro-architecture. It is highly-configurable and includes support for multiple and diverse processor models including ARM, with detailed memory systems and interconnect models. Within gem5 environment, we created an instance of an STNoC network, which we time-annotated from cycle-accurate design specifications of the STMicroelectronics STNoC backbone, including the STNoC router, the NI and a synchronous link. We also implemented a gem5 model of the proposed NoC firewall and integrated it in the NIs of STNoC. Then, we time-annotated the NoC firewall model based on preliminary results from a cycle-accurate design implemented in RTL (VHDL).

In both models, time annotation is based on gem5 scheduler functions that can schedule, deschedule and reschedule events on the simulation timeline. For example, by extending a class called consumer, it is possible to use a function that schedules an event to happen after a specific amount of simulation cycles relative to the current simulation time; upon arrival at this time point a wakeup method is called. Using these classes and corresponding member functions, we are able to schedule dependent and independent events [15].

B. Gem5 STNoC Model and Timing Analysis

Based on available specifications, we have modeled STNoC backbone technology as accurately as possible by making several adjustments to the 5-stages Garnet fixed pipeline model [14], [16], [17]. Several STNoC configuration parameters have been modelled, including link width, packet flits (header and body), virtual circuits (high and low priority), buffer size and number of credits per virtual circuit (VC) and router and NI port. We have also implemented STNoC QoS policies, i.e., fair bandwidth allocation for rate control. These policies apply to flits travelling on the same VC. In fact, our gem5 STNoC router configuration supports three levels of arbitration based on info available in the header flit of the STNoC packet:

1) current faction bit used as a an epoch, i.e., separating messages injected to a router;
2) packet priority, round robin or least recently used (LRU);
3) round robin or LRU as third level of arbitration (this is only used when packet priority is the second level).

In relation to the STNoC router model, we have encapsulated garnet switch allocation (doing port scheduling) within the VC allocator, while also reducing the pipeline depth to match STNoC specifications. In fact, for the STNoC router instance we currently support, the router delay has been reduced to only two cycles, i.e., one cycle for input buffering and route computation stages (IB+RC), and one more cycle for the remaining four stages: 1) virtual channel allocation; 2) switch allocation; 3) switch traversal; and 4) link traversal (VA+SA+ST+LT). Notice that if a flit travels on a high priority VC and the STNoC router behaves "without packet lock," flit interleaving between different VCs is possible and the high priority flit may preempt a low priority one.

Although in our current setup we assume an STNoC topology (normal spidergon) of degree 4, many different topologies with a maximum degree 5 can be modeled by modifying python configuration files. The current implementation of gem5 STNoC router model uses the internal Garnet routing tables. This allows not only to support the only commercially-used STNoC routing strategy, i.e., source-based scheme, but also other deterministic, randomized or adaptive policies for design space exploration.

In our gem5 STNoC model, router-to-router, NI-to-router and router-to-NI LT takes no cycles, similar to the actual STNoC synchronous link implementation. NI latency takes one cycle, which corresponds to STNoC flit registering. The NoC clock frequency can also be configured appropriately (default value is 10^9 ticks per second).

For a cycle-accurate STNoC model, it is necessary to schedule predefined events at appropriate time instances. The gem5 scheduler provides a mechanism to schedule, deschedule and reschedule events on the simulation timeline. Table I explains the major gem5 scheduling events in the gem5 model of the STNoC backbone. Event interactions are modified from the Garnet fixed pipeline model architecture accordingly.

The gem5 STNoC router operates in two pipeline cycles as follows.

1) In the input buffer and route computation pipeline stage (called IB+RC), a head flit, upon arriving at an input port, is decoded and buffered according to its input VC. At the same cycle, a request is sent to the route

TABLE I
EVENT SCHEDULING IN gem5 STNoC NETWORK MODEL

STNoC module	Scheduled events
NI	Wake up output link consumer (downstream router IB+RC)
NI	Self-schedule at next cycle (protocol buffer)
NI	Schedule credit link consumer at next cycle (upstream router)
IB+RC	Wake up VA+SA+ST+LT for VC arbitration
VA+SA+ST+LT	Schedule credit link consumer (upstream router or NI)
VA+SA+ST+LT	Self-schedule at next cycle
VA+SA+ST+LT	Wake up output link consumer (downstream router IB+RC or NI)

computation unit, and the output port for this packet is calculated.

2) Then, for the head flit, the router arbitrates for a VC (VA stage) and proceeds to switch allocation (SA stage) where it arbitrates for the output ports. Upon winning both, the flit moves to switch traversal (ST stage). In ST stage, the flit traverses the crossbar if credits currently exist. ST is followed by LT, whereas the head flit can travel to the next node. Body and tail flits follow a similar pipeline without going through RC and VA stages; instead they inherit the VC allocated by the head flit. The tail flit upon leaving the router, "frees" the VC reserved by the packet. Notice that in parallel to flit processing, credit-based information must be updated at the upstream router based on relevant info collected from downstream routers.

We have considered the timing behavior of the gem5 STNoC model experimentally by designing and simulating static traffic scenarios representing corner-cases. Adherence to cycle-accuracy for all STNoC instances is extremely hard due to the vast number of STNoC configuration parameters and complexity of the gem5 STNoC model (tens of thousands lines of code). The goal of these static scenarios is to calibrate and improve cycle-accuracy of our gem5 models.

As an example, consider router behavior when three STNoC packets of five flits each, enter from three different ports of a single router and exit from two different ports (or a single port). STNoC packets travel on two VCs (VC0 and VC1), which are assigned a static priority, the higher being for VC0. Fig. 4 shows three five-flit packets routed to two output ports (port 0 and port 2), however, in this case two packets are routed on high priority VC0, (flits shown in red color) and one packet is routed on low priority VC1 (flits shown in blue color). The priority of VC0 has a big impact on results, since two packets (ten packet flits) simultaneously flow out of ports 0 and 2 in packet lock fashion (without interleaving). Thus, during time interval 111 to 115, both output ports are fully utilized. Thereafter, only packet 3 remains, so its flits (2_0 to 2_4) exit from port 0 at time interval 116 to 120.

If we make the hypothesis that one of high priority packets is malware, and is thus dropped at the NI, we can observe that the tail of the low priority packet (blue flits) will depart much earlier at time 115, thus saving five cycles. This case, with one malware packet and two normal packets (i.e., 33% malicious requests) results in 100% reduction in total delay. However, considering the five extra cycles required at the NoC firewall (NI) to detect and drop a malware packet, this benefit diminishes to zero.

From the above description we obtain that when malware packets are detected and dropped, increased benefits in total packet delay can result only when network congestion is high, i.e., multiple packets head to the same port and the amount of STNoC flits in a packet is high. We have examined gem5 STNoC model cycle-accuracy by considering its behavior with static traffic patterns.

Next, we will consider dynamic traffic and examine the effect of the NoC firewall to flit latency and delay assuming that a number of malicious accesses occur.

C. Traffic Generation, Rules, and Setting Attributes

To study the system response under different conditions, we developed a method that generates legitimate and malicious requests within the gem5 simulator. We consider two types of processes requesting memory access. The first type corresponds to legitimate (also called safe) processes, S_i, which perform write requests to a certain memory range. The second type corresponds to malicious processes, M_j, which perform remote write requests to random addresses located in the same memory range that is requested by the legitimate processes. Therefore, all memory requests target the range specified by the S_i memory descriptors, and consequently malicious requests interfere with legitimate communications.

For a realistic testbench, we aligned the rule-checking mechanisms to actual segments in the pagemap of the S_i Linux process address space. We accomplished this by cross-compiling and executing the S_i Linux process on top of ARM Fast Models (ARM-FM) simulator [18]. ARM-FM is a functionally accurate environment in which ARM v7 CPU models are implemented as instruction set simulators. Thus, we can model an ARM Cortex-A9 architecture by loading a Linux image and running full system simulation (i.e., executing Linux). The functional behavior of the model is equivalent to real hardware, although timing accuracy is sacrificed for the sake of efficient simulation (and HW/SW development/validation).

In our scenarios, we assume the S_i Linux process allocates an array of 4096 integers, and then continuously writes data to array elements in an infinite loop. This enables examining Linux page tables via ARM-FM and obtaining the memory descriptors that correspond to specific (virtual) array accesses

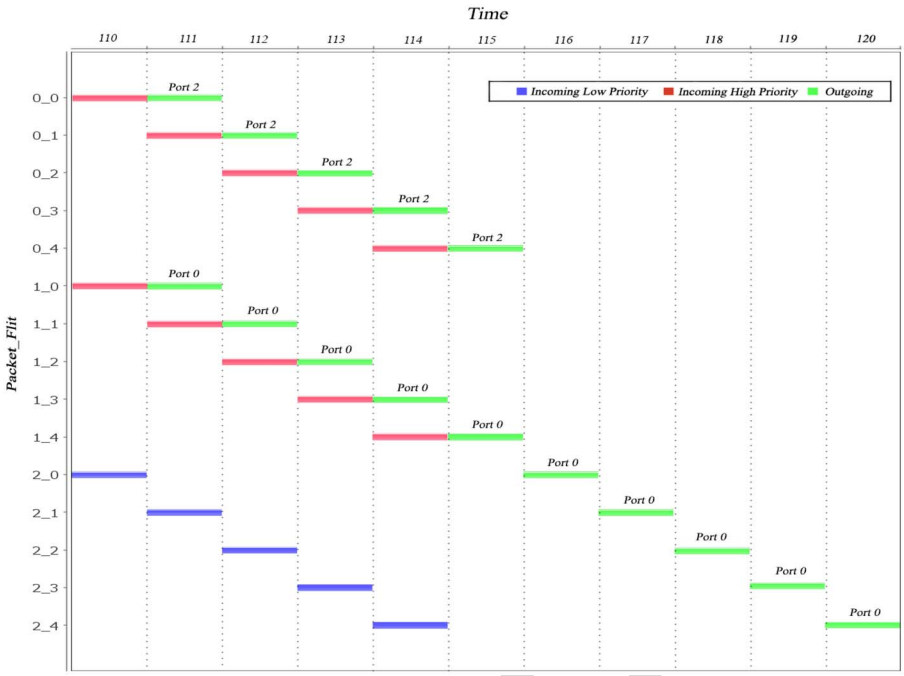


Fig. 4. Scenario with three packets routed to two identical output ports with high or low VC priorities.

at the moment they occur. Thus, in relation to the S_i process, we modified a pagemap-analysis suite of programs and then cross-compiled them for ARM v7 architecture using the common gnu-eabi toolchain [19]. This tool-suite allows us to collect and analyze the Linux pagemaps corresponding to the S_i Linux process (assembly instructions) while it executes on top of ARM-FM. The memory descriptors of the S_i process from Linux pagemaps analysis correspond to the heap range from $0x11000$ to $0x36000$; notice that five descriptors correspond to 5×4K units of byte addressable memory which is enough to store 4K 32-bit integers (the first descriptor essentially contains malloc-specific data).

The use of a 4 KB segment configuration is due to limited support from the OS. Linux process memory allocation often results in noncontiguous descriptors. By analyzing the page map of common PARSEC benchmarks (swaptions and splash), we have noticed that some of the allocated address space corresponds to memory descriptors that are noncontiguous. Hence, in general, it is necessary to configure rules on small segments of granularity (4 KB) in order to avoid denial of access to memory segments that may belong to other kernel or user processes. Stated otherwise, we cannot have a larger segment rule containing all these noncontinuous descriptors, since that would cause denial of access to memory segments that do not belong to the application.

Our framework relies on ARM CPUs and shared memory distributed across a Spidergon STNoC. We have experimented with different configurations by modifying these parameters as follows:

1) number of nodes, i.e., on-chip CPUs and memories;
2) number of legitimate and malicious processes running in each CPU requesting memory access;
3) enabling or disabling the security.

TABLE II
SIMULATION PARAMETERS

Parameter	Type/Value
Interconnect topology	Spidergon STNoC
CPU type	ARM v7 Cortex A-9
Nodes setup	1CPU/2MEM; 2CPU/2MEM; 4CPU/4MEM; 8CPU/8MEM; 16CPU/16MEM
Processes per CPU (Safe,Malicious)	8S1M; 4S1M; 3S1M; 2S1M; 1S1M; 1S2M; 1S3M; 1S4M; 1S8M
Segments allocated for safe processes	5
Segment size	4 KBytes
NoC's link width	16 Bytes
Message type	w; wx
Packet size in Bytes	72 Bytes (header=8; body=64)
Packet size in flits	5
# of Flit credits	5
Status of security mechanism	enabled; disabled
NI, Router, Link delay (in cycles)	{1, 2, 0}

All other factors are kept constant throughout the experimentation process; STNoC link width, packet size, and number of flits per packet have constant values as shown in Table II. This does not limit the generality of our approach, while it allows for a straightforward comparison among different scenarios, e.g., when security is enabled or disabled; when none or many malicious processes run in a CPU, etc.

The above framework enables examining the effect of security on network load and power. This is the consequence of performing access control on requests that are released to the STNoC. For modeling MPSoC application primitives we used ARM-FM and the gem5 simulator. For obtaining cycle-accurate results, we have time-annotated the gem5

STNoC model and NoC firewall from executable specifications (RTL simulation and synthesis). In addition, we used Orion 2.0 [20] for evaluating the STNoC power consumption (65 nm process technology). Our framework can be used to evaluate more intricate networks with more CPUs and memories. In this paper, we present experiments assuming a uniform distribution of processes across CPUs, i.e., all CPUs execute the exact same number of legitimate and malicious processes, however, this number can be modified at will.

D. Type of Measurements

The above framework allows us to study the time a packet remains in the NI queue until it reaches its destination. After a packet is broken down in flits, each flit passes through routers which store the flit in their queue and relay it to the proper path. The framework can be extended to measure delays starting from an earlier phase, e.g., since when a message is generated by a process at the application-layer.

VI. Experimental Scenarios

We experiment with different number of nodes connected to a Spidergon STNoC. Fig. 5 illustrates two cases: 1) 4-node and 2) 8-node Spidergon topology. In our initial work, we conduct experiments with up to 8-nodes only [4]. This paper comes with more experiments for configurations with up to 32-nodes. For each case, we consider input traffic scenarios by varying the number of safe and malicious processes. We even deploy scenarios in which almost all network traffic is malicious. Although this constitutes an extreme situation it is quite realistic, since a virus or malicious program can replicate itself extremely fast, and there is no limit on the malicious traffic that can be generated to memory. Recent DDoS attacks at network gateways have reached bandwidths of over 300 Gb/s [21], while virus propagation models in the literature point to exponential growth, unless the virus enters long periods of inactivity if trying to remain unnoticed [22], [23].

Table II has the parameters of the simulation framework. We ran simulations with one CPU/two memories (actually two CPUs but 1 is inactive), two CPUs/two Memories, four CPUs/four Memories, eight CPUs/eight Memories, and 16 CPUs/16 Memories. We test different scenarios with regard to the number of safe and malicious processes running in the CPUs; in Table II, we represent this as xSyM, where x is the number of safe processes and y the number of malicious processes. The process requesting access to memory is selected with a random policy, and it can be either a "write" or "write-execute" request. A request is performed to a random memory address, which belongs to one of the five allocated memory segments. We perform the same experiments by enabling and disabling the NoC firewall. When enabled, the NoC firewall denies access to every request generated from a malicious processes, since all requests are destined to memory segments that are protected and can be used by safe processes only. Therefore, malicious requests are rejected at the NI, which results in reducing the rate at which traffic enters the NoC. On the other hand, when the NoC firewall is disabled, malicious requests are transmitted to the NoC.

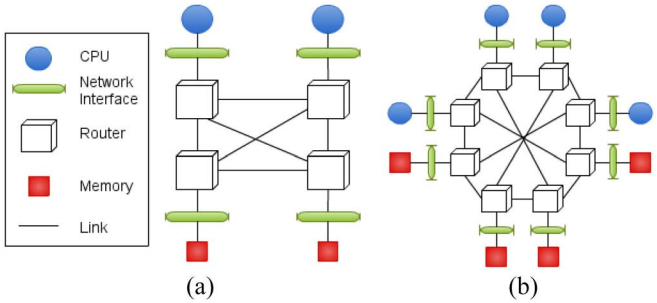


Fig. 5. We use different configurations to study the effect of security. We perform experiments for up to 32-nodes connected to STNoC. Case (a) corresponds to two different configurations; either both CPUs or only one CPU can be active. (a) 4-node STNoC. (b) 8-node STNoC.

In our traffic scenarios, we are interested in examining NoC response for a different number of safe and malicious requests, rather than comparing its behavior for different percentages of malicious requests over total ones. More specifically, 1S1M scenario corresponds to 50% malicious requests within a time-window of sending two packets, while 1S8M scenario corresponds to 88.8% malicious requests within a time-window of sending nine packets. In both scenarios, when NoC firewall is enabled only one packet will be released to the NoC within their corresponding time-window. Also, consider the 1S1M and 10S10M scenarios; these generate the same percentage of malicious requests, i.e., 50%. Their difference is the following: 1S1M corresponds to a time-window of two packets in which safe and malicious requests are generated in an one-by-one alternate sequence, i.e., one safe, one malicious, then again one safe, and so on; 10S10M corresponds to a time-window of 20 packets in which a chunk of safe requests is first produced, followed by a chunk of malicious requests, i.e., 1–10 are safe requests in a sequence, 11–20 are malicious requests in a sequence, then again 1–10 safe requests, and so on. When comparing these scenarios, the actual number of packets that are eventually transmitted when the NoC firewall is enabled, will be exactly the same for a given simulation time period, i.e., in our case 40 000 clock cycles. However, their major difference lies on the amount of safe packets arriving at the NoC in succession. Having big chunks of safe content rather that smaller ones, imposes high demands from the side of NoC, i.e., the NoC should be able to sustain a high serving rate. For instance, the more demanding case from the aforementioned ones is the 10S10M scenario. In particular, in the 16CPU/16MEM setup, there will be time-windows in which all 16 CPUs produce concurrently ten safe requests in a sequence; this stands for each CPU, and all are expected to be served by the NoC. After the safe traffic, the malicious traffic follows, which is perceived again as a big chunk of malicious content. However, this will be stopped by the NoC firewall. On the other hand, when the NoC firewall is disabled, being concerned about the percentage of malicious requests is meaningless, since all packets are allowed to enter the NoC, e.g., in the 10S10 scenario all 20 packets are transmitted to the NoC, and this will be repeated until completing the simulation runtime. In conclusion, in this paper, it is much more interesting

to examine system behavior in relation to the amount of packets that are eventually released to the NoC. This will allows us to reveal cases in which the NoC is flooded and becomes saturated.

A setup-phase is needed to configure the rule table in each NoC firewall. For our experiments we do this with five CPU commands—one per segment—for every malicious process. For example, for three malicious processes, during warm-up phase, each CPU sends $3 \times 5 = 15$ commands, resulting in a total of 15 deny rules. This way, during normal operation the firewall will block a malicious process from accessing protected segments. No setup rules are needed for the legitimate processes, since NoC firewall operation relies on deny rules only. During setup-phase, the packet injection rate (in the gem5 Network_test simulation script) is lowered to guarantee arrival of setup packets from the CPU to the NI prior to the time (safe and malicious) processes start issuing actual memory requests.

By selecting different parameter values we perform experiments for several configurations. As explained in Table II, five different node setups, nine different combinations of safe and malicious requests, and two options for the security mechanism (enabled/disabled), give a total of $5 \times 9 \times 2 = 90$ different configurations. Regarding NoC parameters, the packet size of a write command is 72 bytes, while the link width is 16 Bytes, i.e., 16 bytes at most can traverse the link per transaction. Hence, each packet is split into five flits, since $72/16 = 4.5$.

Another parameter that affects our experiments is the packet injection rate, i.e., the rate at which packets are released from the application layer in each CPU. Variations in the injection rate—even small ones—affect the end-to-end delivery time of the packets tremendously when the NoC starts experiencing heavy network congestion which result in saturation conditions. We repeated the experiments numerous times in order to verify such occurrences. In fact, we did this for large simulation times, starting from a few hundred up to 400 000 clock cycles, and we made interesting observations related to the effect of security on network saturation.

VII. Effect of Security and NoC Firewall Performance

In this section, we discuss the way we employ the above framework to evaluate the effect of security on end-to-end delivery time and power consumption at network-layer. Then, we compare performance of our proposed NoC firewall against a SW solution of equivalent functionality.

A. Effect of Security on Network Transmission

Our framework enables measuring end-to-end delivery times at network level, and power consumption of the network routers and links. We measure two types of delay; the time a packet is waiting in the NI queue before it is actually broken down into flits and released to the network (called NI queue delay), and then, after the flit leaves the NI queue, the time spent to traverse the network until it reaches its destination. The latter delay is called network traversal delay, and it is the time required for the flit to pass through all the routers on its path and arrive at the target NI. Table III shows the results for a specific configuration when NoC firewall is disabled or enabled. The last column shows the impact from activating the firewall. Negative values indicate relative improvement, thus in all cases firewall activation affects positively network metrics.

TABLE III
Effect of Security on Queue Delay, Network Delay and Power at the STNoC Level. Scenario Concerns 4CPU/4MEM Setup With 1*S*2*M* Requests

Metric	without firewall	with firewall	+/-(%)
NI queue delay	28.0905 cycles	6.1824 cycles	-77.99%
Network traversal delay	10.5503 cycles	8.3907 cycles	-20.47%
Router power (total)	0.5364 Watt	0.3746 Watt	-30.16%
Router clock power	0.1917 Watt	0.1917 Watt	0%
Router static power	0.1189 Watt	0.1189 Watt	0%
Router dynamic power	0.2258 Watt	0.0640 Watt	-71.66%
Link power	0.0297 Watt	0.0084 Watt	-71.68%

The chosen case concerns the 4CPU/4MEM node setup, with one safe and two malicious CPU processes requesting memory access. It is obtained that when NoC firewall is active, both the traversal delay and the delay in NI queues decrease. This is due to the fact that malicious requests are prevented from entering the network, thus fewer packets are released, resulting in lower network traffic and smaller queues. Moreover, the total power of routers decreases by 30.16%, mainly due to the drastic decrease in dynamic power. Finally, the use of firewall reduces the power consumed at network links by almost 72%.

The above analysis indicates that a security mechanism relieves the network from unnecessary load in the presence of malicious processes. The values in Table III are average ones and concern all CPUs. We perform experiments for two type of messages, e.g., “write” and “write-execute.” For these specific experiments the amount of flits transmitted is 2280 flits when firewall is on, and 7315 flits when firewall is off. The above use case demonstrates the way the framework is used to evaluate the effect of security both on delay and on power consumption at the NoC layer. Next, we include further results on the measured delays and power consumption for all configurations of Table II.

Fig. 6 shows the total power consumed at the NoC routers for different node setups and for different number of safe and malicious processes running in each CPU, both when security is enabled and disabled. Significant savings are obtained when the number of nodes connected to the network becomes large, i.e., 16-nodes and 32-nodes setups, and the number of malicious requests increases, in Fig. 6. We also observe that when security is disabled, power consumption remains essentially unchanged for a given node setup regardless of the number of safe and malicious processes. This is due to the fact that requests are always served, thus packet are always transmitted to the network; hence, for every given node setup of Table II, the power consumption is not reduced.

Fig. 7 shows the total NoC delay, i.e., the sum of NI queue delay and network traversal delay, for different node setups and for different number of safe and malicious processes running in each CPU, both when security is enabled and disabled.

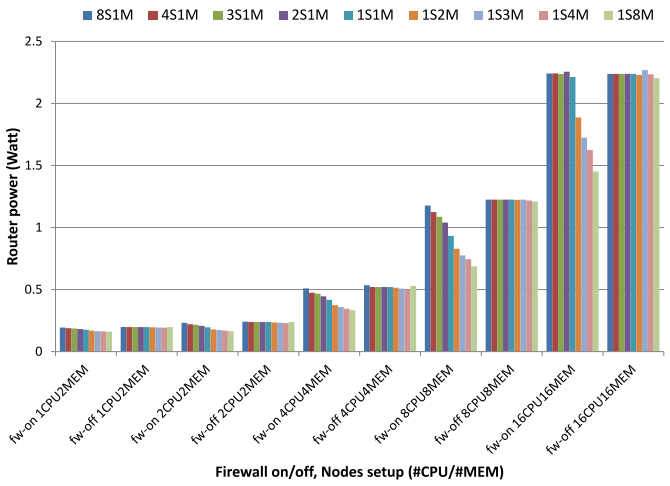


Fig. 6. Power consumption at the routers for all setups of Table II, when firewall is on or off.

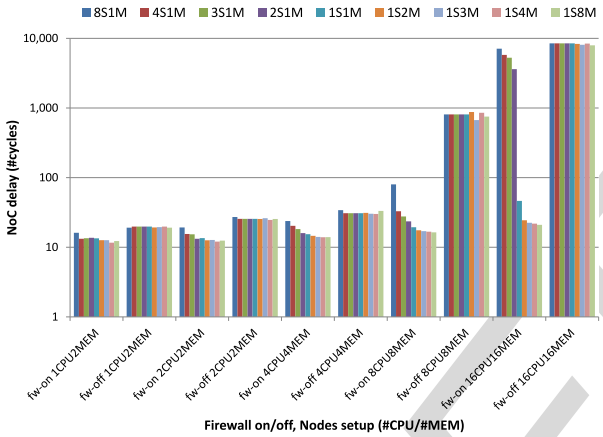


Fig. 7. NoC delay for all setups of Table II, when firewall is on or off (scale is logarithmic).

The total NoC delay is computed as the average time required for a flit to remain in the queue of the NI, and then traverse the network by passing through the routers until it reaches its destination. It appears that the greater benefit from incorporating a security mechanism comes when the number of malicious requests is large. For example, we observe that if malicious processes are invoked per CPU—in Fig. 7 this is represented with 1*S*8*M*—the NoC delay when security is active has the smallest value; this holds for all node setups, from 2CPU/2MEM to 16CPU/16MEM. It is also observed that when security is disabled, NoC delay remains unchanged for all scenarios of the chosen setup. Again, this is due to the fact that requests are always served, thus they are always transmitted to the network.

In Fig. 7 an interesting result is derived from observing network behavior in the two largest setups, i.e., 8CPU/8MEM and 16CPU/16MEM. In both cases, the benefit from incorporating a security mechanism is higher when the number of relative malicious processes increases. Especially in the

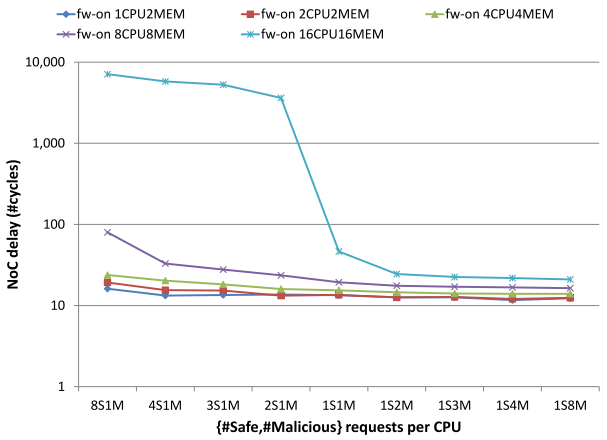


Fig. 8. NoC delay for all setups of Table II, when firewall is on (scale is logarithmic).

16CPU/16MEM setup, the security mechanism clearly prevents the network from being saturated, by prohibiting unnecessary packets from being released to the network. In fact, for the specific setup when firewall is on, we notice that the NoC delay is reduced dramatically when the number of malicious processes is equal to, or, larger than safe processes. As shown, for 2*S*1*M* the NoC delay is 3616 clock cycles (scale in Fig. 7 is logarithmic), while for 1*S*1*M* the NoC delay is 46 clock cycles. This phenomenon is related to the packet injection rate, since the firewall mechanism relieves the network, and consequently reduces network congestion by cutting off early all malicious packets.

Fig. 8 has a different representation for assessing the effect of security. It concerns only the case where security is activated, covering all setups of Table II. Again, we obtain that the more nodes are connected to the NoC, the higher the benefit. Also, as expected, the NoC delay increases for a large number of safe processes per CPU. Moreover, this figure illustrates that for 8*S*1*M* the gap of NoC delay between the 16CPU16MEM setup and the other setups is much bigger.

B. Saturation Analysis

If all our experiments so far we have used a default injection rate of 0.1 packets per CPU per cycle. Moreover, depending on the Spidergon topology, i.e., number of CPUs/memories, amount of legitimate al versus malicious processes, and NoC firewall status (enabled or disabled), the network reaches a saturation point in certain setups for the selected injection rate.

Additional simulations in all cases of 16CPU/16MEM setup revealed the extent to which the NoC firewall can improve network performance, and therefore allow for higher injection rates. For example, to avoid NoC saturation in the case of 16CPUs/16MEM and when NoC firewall is disabled, injection rate should be reduced down to 0.04 packets per cycle in each CPU. When enabling NoC firewall, the injection rate can be gradually increased without the danger of causing saturation. Table IV illustrates that the benefits increase with the number of malicious requests. In fact, it appears that when NoC firewall is enabled, the maximum packet injection rate that can be served by the NoC (without causing saturation)

TABLE IV
MAXIMUM PACKET INJECTION RATES—PER CPU CLOCK CYCLE—THAT THE NOC SERVES WITHOUT BEING SATURATED. RESULTS CORRESPOND TO 16CPUS/16MEMS SETUP, WHEN BOTH SECURITY IS DISABLED, I.E., FW-OFF AND ENABLED, I.E., ALL OTHER CASES

fw-off	8S1M	4S1M	3S1M	2S1M	1S1M	1S2M	1S3M	1S4M	1S8M
0.04	0.05	0.05	0.06	0.07	0.09	0.13	0.17	0.2	0.4

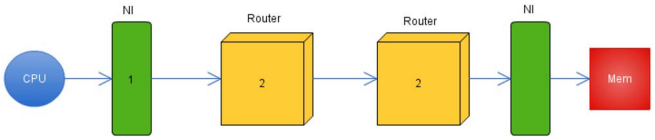


Fig. 9. Network latency measured from NI-to-NI. Figure illustrates all components found in the path of a packet transfer; NI source buffering, routers, NI at the destination. We propose integrating the security mechanism at the NI source.

increases with the number of malicious packets attempting to enter the NoC. For example, in the extreme 1*S*8*M* scenario the injection rate can be set up to 0.4; this is one order of magnitude higher than in the case of disabled firewall. Although Table IV corresponds to a certain setup described in Table II, it is quite representative as it shows the importance of NoC security for maintaining NoC services in the presence of malicious requests.

C. Analysis of the STNoC Latency

At this point, it is worth examining the latency at the NoC layer, which allows for explaining the way the packets are transferred across the NoC. Fig. 9 shows an example of a (5-flit) packet transmission; it is sent by a CPU, transmitted on the NoC by passing through the routers, and consumed by the destination NI. By breaking down the transfer time, one clock cycle (cc) is spent at the STNoC NI and two clock cycles are spent at each router. The aggregation of these values is the latency of the path. In an uncongested network, successive flits are ideally released cycle-by-cycle. Thus for the first flit (packet header) the amount of time required to be delivered at the destination NI is five clock cycles, while one additional cycle is required for each successive flit, i.e., six clock cycles are needed for the second flit, etc. Hence, once a packet has entered the source NI, it requires a total of nine clock cycles to be delivered at the destination NI. It should be noted that all flits of a packet are routed through the same path.

D. HW Versus SW Implementation of the NoC Firewall

To assess the performance of our NoC firewall we compare it with a software implementation. This is used as a reference just to demonstrate the benefits from implementing a dedicated security module in hardware. In particular, we compare the overhead of our hardware implementation versus an equivalent software implementation in ARM v7 Cortex-A9 processor. We consider timing of different events that correspond to the NoC firewall driver functions. Results in Table V indicate that the merit from implementing protection in hardware is large.

To generate ARM v7 assembly code, we cross-compile the firewall driver configuration and access request functions using arm-linux-gnueabi-gcc and flags "-c -g -Wa, alh, -ad -fverbose-asm" [24], [25]. Then, we evaluate the delays of the cross-compiled source code functions using cycle-approximate delay for each ARM assembly instruction (e.g., load, store, add, subtract and multiply, compare and branch, move and shift). Timing information is available from the technical architecture reference manual for Cortex-A9 [26].

TABLE V
TIME OVERHEAD OF NOC FIREWALL: HARDWARE VERSUS SOFTWARE IMPLEMENTATION

NoC firewall action	# cycles in HW	# cycles in SW
Initialization of 16 segments	∼160	62,066 to 65,904
Access request	2 or 5	112 to 2,127
Add segment table entry	3	81 to 1,019
Delete segment entry for given PID	3	79 to 513
Delete all segment entries	3	1,977 to 2,411

VIII. COARSE-GRAIN VERSUS FINE-GRAIN APPROACH

This section offers insights on the advantages over alternative firewall solutions by examining the overhead added by the distinct sub-phases that occur during handling memory requests. To study further the efficiency of our coarse-grain approach, we proceed with an implementation of a fine-grain approach that performs rule checking at page-level, and then conduct a straightforward comparison among them. First, we provide some details on the implementation of our fine-grain approach, but since this is not the main scope of this paper, we avoid delving into all the details. Our fine-grain solution supports a 4 KB page size and consists of three tables. A unique first level table (FLT) at the NoC firewall is accessed using process identifier (PID) information to provide a base address (32 bits) to a second level table (SLT). For each PID, SLT holds a physical base address to the pages of rules (PoR) table that is located in memory. We assume that PoRs can be located in noncontiguous physical memory.

We consider a PID field of 6 bits, which allows support of up to 64 processes as in our coarse-grain solution, a page size of 4 KB bytes, an address pointer size of 4 Bytes, an address space of 4 GB, and rule size of 8 bits. Hence, the PoR memory structure must support 1M pages (as a result of the division 4 GB ÷ 4 KB), i.e., its size is 1 MBytes (1M pages × 1 Byte rule). In addition, the FLT data structure located at the NoC Firewall must provide a base address pointer of 4 Bytes for each PID (set in noncontiguous address space) and has a size of 256 Bytes. Finally, the SLT must serve a total of 256 entries (256 × 4K rules = 1M rules), thus the SLT has a size of 64 KBytes (1 KByte for each PID). We have also used a 64 × 8 CAM structure (from micrometer) to cache up to 64 rules. The CAM allows for one search, one read and one write operations to proceed in parallel; the access time for search is one cycle, and ten cycles for read and write. Taking this functionality into account, we can now compare the coarse-grain with the fine-grain approach as follows.

1) If the segment does not exist (thus request will be allowed), the coarse-grain approach adds two

TABLE VI
IMPACT OF THE FINE-GRAIN NoC FIREWALL ON NoC PERFORMANCE AND POWER CONSUMPTION FOR DIFFERENT SCENARIOS IN THE 2CPU2MEM SETUP. INJECTION RATE IS 0.1 PACKETS PER CYCLE IN EACH CPU. NoC IS SATURATED IN ALL SCENARIOS

	8S1M	4S1M	3S1M	2S1M	1S1M	1S2M	1S3M	1S4M	1S8M
NoC delay (#cycles)	3319.4	3324.4	3325.6	3333.3	3334.6	3355.1	3293.7	3316.8	3321.0
Router power (Watt)	0.2176	0.2116	0.2078	0.2018	0.1904	0.1783	0.1729	0.1697	0.1626

cycles delay. On the other hand, in the fine-grain approach, if a page does not exist in the TLB (implemented as CAM), the delay rises to two clock cycles—one clock cycle for CAM search (valid bit) and one for comparison—plus twice the network roundtrip traversal time of a packet for rules page walk, i.e., for fetching the rules from external memory. Notice that each network roundtrip traversal time includes the time for delivering all packet flits for one memory read operation.

2) If the segment exists (request will be either allowed or denied after further processing), the coarse-grain approach adds a fixed delay of five cycles. In the fine-grain implementation, if the page is in the TLB, a total of 12 cycles are required; one for CAM search (valid bit), ten for TLB read, and one for the comparison.

We have performed experiments of limited extent with the fine-grain NoC firewall within the gem5 environment. So far, we have studied only the best-case in which the page already lies in the TLB, thus a page walk is not performed. We studied the 2 CPU/2 MEM node setup, for all scenarios, with an injection rate of 0.1 packets per cycle in each CPU; we used the same rate in the experiments with our coarse-grain solution described in Section VII-B. Table VI indicates that when the fine-grain NoC firewall is enabled, NoC becomes saturated in all scenarios. This is mainly due to the fact that the injection rate equals to one packet per ten cycles per CPU, while the fine-grain NoC firewall itself requires 12 cycles to examine the packet; thus the injection rate is higher than the serving rate of NoC firewall. To avoid saturation phenomena the injection rate should be lower than the NoC firewall serving rate. In the coarse-grain solution, we did not observe saturation in this set of experiments, i.e., all packets are delivered within a reasonable amount of time (see Fig. 7).

We expect that in the worst-case, i.e., when a page walk is needed for fetching the rules from external memory, the overhead of the fine-grain approach will be even bigger. The rules table walk makes it at least twice as much as the one for the best-case of fine-grain due to accessing remote memory, thus traversing the NoC for fetching the rule. In conclusion, the flexibility of a fine-grain approach, which allows for each page having a different rule, is gained not at the expense of memory space, but at the expense of performance due to the—at minimum—two-level rules table walk.

IX. NoC FIREWALL IMPLEMENTATION PROTOTYPE

The goal of the coarse-grain NoC firewall is to control accesses to regions of system memory, or to any memory-mapped slave peripheral at the NoC initiator NI. The number of memory partitions (segments) and their range and the corresponding rules that are enforced are system-defined. To validate the functionality of our NoC firewall, we have prototyped it using a hardware infrastructure on the basis of an AXI4 interconnect in the Xilinx Zynq-7020 FPGA of a ZedBoard platform. Fig. 10 outlines the system architecture. The NoC firewall is attached to the AXI4-to-AXI4 bridge. The core is connected to the processing system (PS) master general purpose port, and to the PS slave high performance (HP) port; the latter is used to connect the programmable logic (PL) to the external memory (DDR3) through a switch. The AXI-to-AXI bridge transfers CPU requests to the DDR3 memory controller, abstracting the NoC fabric that interconnects all initiators to the system memory. The bus width is 32-bit and the PL is clocked at 100 MHz.

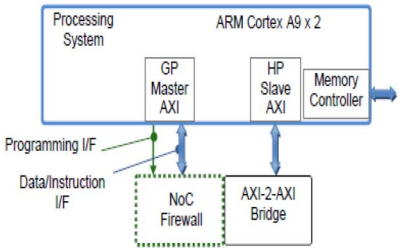


Fig. 10. NoC firewall implemented in a Xilinx Zynq-7020 FPGA.

The NoC firewall processes all incoming physical addresses, searching whether they are subject to preconfigured rules. We employ the following policy to secure access control on the incoming transaction requests.

1) If there is no match in the NoC firewall data structures, the transaction request is allowed to proceed.
2) If an incoming physical address is in between a preconfigured address range, then this match implies that the address belongs to a particular segment subject to restrictions, and hence the rules table is accessed to decide if this access will be permitted.

The coarse-grain protection implementation currently supports 16 memory segments. The maximum number of segments that can be implemented without affecting timing is technology-dependent. The 20-bit high order part of the physical address is utilized to concurrently search the predefined ranges of each programmed segment. In this case, the corresponding enable signal for this segment is asserted. The part of the physical address that is used currently is fixed but can be dynamically programmed. In a multiprocess environment the OS can dynamically provide the six-bit PID in order to access the base address of the rules defined for a particular PID. The encoded result is used to index the discovered rule inside the set of rules for this PID.

Each rule in the implementation consists of eight nonencoded bits, hence rule memory is of size 1024×8; this is in

TABLE VII
LATENCY IN CLOCK CYCLES (3.3 ns) FOR READ/WRITE ACCESS USING THE NOC FIREWALL ATTACHED TO AXI4-COMPLIANT BRIDGE ON Zynq7020 FPGA

Operation(Type/Number)	#10	#100	#1000
Wr_br	3,628	15,366	503,575
Wr_mctrl	732	5,580	58,670
Rd_br	1,911	17,460	175,633
Rd_mctrl	538	4,966	51,172

TABLE VIII
AREA COST AND FREQUENCY OF ONE INSTANCE OF THE NOC FW AND ONE INSTANCE OF THE BRIDGE IN THE Z7020 FPGA. REPORTED STNOC RESOURCES ARE FOR COMPARISON PURPOSES AND CORRESPOND TO 7×6 NETWORK, IMPLEMENTED IN A LARGER FPGA, OF THE SAME TECHNOLOGY

Block	#LUTs	#Registers	Clock(MHz)
NoC firewall (64 PIDs,16 segments)	655	1,082	348.8
AXI-to-AXI bridge (full AXI4)	2,764	3,441	220.6
STNoC	24,939	17,983	129.3

contrast to the three-bit encoded information used to specify a rule in the gem5 model. Each eight-bit rule in the implementation is formed by specifying eight subfields (operating modes): 1) read; 2) write; 3) data; 4) execute; 5) privileged; 6) nonprivileged; 7) secure; and 8) nonsecure domain. However, notice that not all subfields can be set independently.

A preliminary set of bare-metal experiments that measure the latency of read/write operations using the embedded SCU timer is drafted in Table VII. Operations with suffix "_br" represent accesses that utilize the custom AXI bridge to connect the initiating operation (read or write) from the processor inside the PS through the PL to the HP port that connects to the memory controller. The operations with the suffix "_mctrl" refer to processor accesses through the internal PS interconnect to the DDR3 memory. As shown in Table VII, write operations through the custom bridge present a large variance, which is due to DDR3 memory refresh occurring at the same time as the transfer of data. The CPU accesses the DDR3 memory with an average latency of 52 clock cycles (a clock cycle is 3.3 ns), while accesses through the NoC firewall and the bridge exceed a latency of 175 clock cycles. A malware access captured by the NoC firewall causes a CPU interrupt. The service routine finishes by clearing the interrupt signal that is generated by the firewall. There is a delay of over 350 clock cycles between the interrupt being asserted and the service routing clearing the control bit; in fact, during iterative generation of interrupts this delay reaches up to 520 cycles.

On the basis of these system latencies, even though one actual firewall operation completes every four clock cycles (or 4.74 ns as the Xilinx P&R tools report), the actual latency of CPU for the transfer to PL and latency of the interrupt service is much larger. Each malware access that is dropped causes at least 175 clock cycles delay to serve by the Cortex-A9 CPU, which is comparable to one successful read access when using the AXI4 bridge that embeds the NoC firewall.

As a real-world use case, the NoC firewall is ported into a Zedboard running GNU/Linux. The Linux driver initializes memory segments and configures rules by performing ioremap() to map physical addresses of the segment and corresponding rule registers and the irq reset register. Then, the module calls request_irq[91, (irq_handler_t) my_irq_handler, IRQ_TYPE_EDGE_RISING, "noc_firewall," NULL] to register my_irq_handler as the handler for irq 91. The module also provides standard read/write functions for capturing (resp. modifying) the status of the NoC Firewall. Within this context, we have a threat scenario that focuses on protection from a malicious (or corrupt) device driver accessing physical memory (via I/O read/write) by setting appropriate deny rules. Notice that if left alone, the malicious driver (implemented as a *k*th read) may cause undesired behavior or even system crash, e.g., if sensitive info is overwritten. This scenario is useful in validating our firewall for read/write data access, i.e., by considering all possible rule setups and operating system modes (more than 65K cases). Moreover, by profiling [caling ktime_get()], we can get an insight into driver performance. During module entry:

1) kmalloc, data initialization, virtual to physical address translation, as well as writing a segment or rule register take between 100 to 1K ns;
2) request_mem_region, ioremap (for range and rule registers) take on the order of 1K to 10K ns each;
3) ioremap for IRQ register takes 10K to 100K ns;
4) registering the interrupt handler takes 70K to 700K ns.

During normal operation, successful data reads/writes takes 100 to 1K ns (writes are by 100 ns), while reads that cause interrupt (denied read) take 10K to 100K ns. Finally, during module exit, freeing memory takes 100K ns.

Based on the above profiling, we notice that interrupt handling under Linux is very slow compared to other operations (expensive ones occur rarely), taking approximately 90% of the total access time for both read and write accesses, thus ratifying our previous data from bare metal applications. Based on these results, it would be better to generate interrupts in a controlled way, e.g., issue only one interrupt per segment, or, as needed by the level of "suspicion" of the intrusion detection system (IDS). Interrupt coalescing schemes can also hold back interrupts either until reaching a certain threshold, or until a timeout timer triggers. In this way, we anticipate to reduce interrupt load associated to system call handling and related protocol processing by at least one order of magnitude.

Table VIII summarizes the FPGA implementation results. The NoC firewall occupies less resources than the AXI-to-AXI bridge and the latency to process an incoming transaction is 4 clock cycles. For comparison reasons, a similar size (7×6) STNoC fabric with 32-bit AXI-3 interfaces occupies significant area on the Virtex-6 FPGA of a Versatile Express platform and operates with a maximum frequency of 129.3 MHz. In total, our firewall utilizes 53% of the Slice Look-Up Tables (LUTs) and 21% of the Slice Registers of the Zynq-7020 FPGA.

We do not include power consumption of our design on the Zedboard, since estimates based on spreadsheets from Xilinx are currently inaccurate [27]. In this direction, we plan to consider ongoing efforts attempting to improve power estimation across the board [28].

X. Conclusion

Due to the wide adoption of NoC-based MPSoC technology, security aspects and memory protection services are of major concern. In this context, we have implemented a hardware NoC firewall with deny rules statically configured at the NI of all initiator nodes. This allows preventing the injection of malicious requests at an early phase, prior to releasing them into the NoC.

Using an instance of the industrial STNoC interconnect as the baseline architecture, we have created a cycle-accurate framework based on the gem5 environment that combines ARM v7 architecture with our STNoC model and security features. Within this framework, we have studied the effectiveness of security and revealed a significant reduction of end-to-end delivery time and dynamic power consumption at the network layer, especially when the number of malicious requests increases. An advantage of incorporating security at the initiator NIs is that it can prevent the NoC from becoming saturated. Furthermore, using an FPGA-based implementation of the NoC firewall on ZedBoard running GNU/Linux, we have showed that the required resources are relatively limited, while the high cost of interrupt can be relieved by implementing an IDS. The IDS can implement monitoring functionality (interrupt counters) that depends on the application domain and takes into account the precise ARM v7 operating mode, i.e., secure/nonsecure world, privileged/nonprivileged mode (root/user) and access type (read/write, data/instruction). This would enable the design of hierarchical security policies, e.g., combining a quarantine mode denying read/write/execute for secure IP with a deny write/execute rule for non-critical IPs, which allows viewing system files or taking backups.

In this paper, we have extended the gem5 framework with time-annotated NoC firewall and gem5 STNoC models; the STNoC model will be released to gem5 community in June 2016. With some extra effort, we can study the effect of NoC firewall on application performance, thus estimating the NoC firewall contribution to the total system overhead.

Another future step is to study how to effectively synchronize all NoC firewalls across the MPSoC when an update of rules is performed. Rules can be changed dynamically, however, during that time it is necessary to guarantee that pending transactions across all NoC firewalls are consistent with the current set of rules. One possible way to do this is via a central barrier operation on the processor; in addition, it is necessary to enable the rule update process to control the AXI protocol handshake, thereby blocking outgoing traffic while rules are being updated at each NoC firewall.

Acknowledgment

The authors would like to thank the anonymous reviewers for their critical comments and suggestions.

References

[1] M. Coppola, M. Grammatikakis, G. Kornaros, and A. Spyridakis, "Trusted computing on heterogeneous embedded systems-on-chip with virtualization and memory protection," in *Proc. 4th Int. Conf. Cloud Comput. GRIDs Virtual.*, Valencia, Spain, 2013, pp. 225–229.

[2] M. Coppola, R. Locatelli, G. Maruccia, L. Pieralisi, and A. Scandurra, "Spidergon: A novel on-chip communication network," in *Proc. 4th Int. Symp. Syst. Chip*, Tampere, Finland, Nov. 2004, pp. 15–26.

[3] M. Mitić and M. Stojčev, "A survey of three system-on-chip buses: AMBA, coreconnect and wishbone," in *Proc. 41st Int. Conf. Inform. Commun. Energy Syst. Technol. (ICEST)*, Sofia, Bulgaria, 2006, pp. 282–285.

[4] M. D. Grammatikakis *et al.*, "Security effectiveness and a hardware firewall for MPSoCs," in *Proc. IEEE Int. Conf. High Perform. Comput. Commun. (HPCC)*, Paris, France, Aug. 2014, pp. 1032–1039.

[5] L. Fiorin, G. Palermo, and C. Silvano, "A security monitoring service for NoCs," in *Proc. 6th IEEE/ACM/IFIP Int. Conf. Hardw./Softw. Codesign Syst. Syn. (CODES+ISSS)*, Amsterdam, The Netherlands, 2008, pp. 197–202.

[6] L. Fiorin, G. Palermo, S. Lukovic, V. Catalano, and C. Silvano, "Secure memory accesses on networks-on-chip," *IEEE Trans. Comput.*, vol. 57, no. 9, pp. 1216–1229, Sep. 2008.

[7] J. Porquet, A. Greiner, and C. Schwarz, "NoC-MPU: A secure architecture for flexible co-hosting on shared memory MPSoCs," in *Proc. Design Autom. Test Europe (DATE)*, Grenoble, France, Jul. 2011, pp. 591–594.

[8] A. Wiggins, S. Winwood, H. Tuch, and G. Heiser, "Legba: Fast hardware support for fine-grained protection," in *Proc. 8th Asia-Pac. Conf. Adv. Comput. Syst. Archit. (ACSAC)*, Aizuwakamatsu, Japan, Jul. 2003, pp. 320–336.

[9] Verizon. (2015). *Security Management Program (SMP)*. [Online]. Available: http://www22.verizon.com/wholesale/solutions/solution/SecurityManagementProgram.html

[10] R. Panigrahy and S. Sharma, "Sorting and searching using ternary CAMs," *IEEE Micro*, vol. 23, no. 1, pp. 44–53, Jan./Feb. 2003.

[11] M. LeMay and C. A. Gunter, "Network-on-chip firewall: Countering defective and malicious system-on-chip hardware," *ACM Comput. Res. Reposit.*, arxiv:1404:3485, Apr. 2014.

[12] A. Gutierrez *et al.*, "Sources of error in full-system simulation," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Monterey, CA, USA, Mar. 2014, pp. 13–22.

[13] G. Palermo, C. Silvano, G. Mariani, R. Locatelli, and M. Coppola, "Application-specific topology design customization for STNoC," in *Proc. 10th Euromicro Conf. Digit. Syst. Design Archit. Methods Tools (DSD)*, Lübeck, Germany, Aug. 2007, pp. 547–550.

[14] N. Binkert *et al.*, "The gem5 simulator," *ACM SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, May 2011.

[15] The Gem5 Simulator System. (2015). *Events*. [Online]. Available: http://www.m5sim.org/Events

[16] The Gem5 Simulator System. (2015). *Interconnection Network*. [Online]. Available: http://www.m5sim.org/Interconnection_Network

[17] N. Agarwal, T. Krishna, L.-S. Peh, and N. K. Jha, "GARNET: A detailed on-chip network model inside a full-system simulator," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Boston, MA, USA, Apr. 2009, pp. 33–42.

[18] N. Rodman, "ARM fast models-virtual platforms for embedded software development," *Inf. Quart. Mag.*, vol. 7, no. 4, pp. 33–36, 2008.

[19] *Linux Community. Pagemap Collection, Analysis and Statistics-Linux Page Tables*. [Online]. Available: http://www.eqware.net/Articles/CapturingProcessMemoryUsageUnderLinux

[20] A. Kahng, B. Li, L.-S. Peh, and K. Samadi, "ORION 2.0: A fast and accurate NoC power and area model for early-stage design space exploration," in *Proc. Int. Conf. Design Autom. Test Europe (DATE)*, Nice, France, Apr. 2009, pp. 423–428.

[21] rt.com. (2015). '*Biggest Ever'? Massive DDoS-Attack Hits EU, U.S.* [Online]. Available: http://rt.com/news/biggest-ddos-us-cloudflare-557/

[22] X. Wei *et al.*, "Smartphone viruses propagation on heterogeneous composite networks," in *Proc. 2nd IEEE Netw. Sci. Workshop (NSW)*, West Point, NY, USA, Apr. 2013, pp. 106–109.

[23] J. Jackson and S. Creese, "Virus propagation in heterogeneous Bluetooth networks with human behaviors," *IEEE Trans. Depend. Secure Comput.*, vol. 9, no. 6, pp. 930–943, Nov./Dec. 2012.

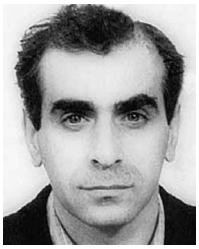
[24] IBM. (2015). *Install the GNU ARM Toolchain Under Linux*. [Online]. Available: http://www.ibm.com/developerworks/linux/library/l-arm-toolchain/

[25] Mentor Graphics. (2015). *Mentor Embedded Portal*. [Online]. Available: https://sourcery.mentor.com/sgpp/lite/arm/portal/subscription

[26] ARM, "ARM Cortex-A9 technical reference manual (appendix B: Cycle timings)," ARM, Cambridge, U.K., Tech. Rep. DDI 0407, 2012.

[27] Xilinx Inc. (2014). *Xilinx Power Estimator*. [Online]. Available: http://www.xilinx.com/ise/power_tools/license_7series.htm

[28] OFFIS. (2015). *Total Power Estimation on Zynq Board*. [Online]. Available: https://contrex.offis.de/home/index.php/news/88-european-mixed-criticality-cluster-workshop



Miltos D. Grammatikakis received the M.Sc. and Ph.D. degrees from the University of Oklahoma, Norman, OK, USA, in 1985 and 1991, respectively, both in computer science.

He was with Academia, research and industry in France, Germany, and Greece. He is currently a Professor with the Technological Educational Institute of Crete, Heraklion, Greece. He has participated in numerous European and National Research and Development projects. He has collaborated with STMicroelectronics for over 15 years and has published over 70 technical articles and holds one patent. He is a co-author of two scientific books entitled *Parallel Systems: Communications and Interconnects* in 2001 and *Design of Cost-Efficient Interconnect Processing Units: Spidergon STNoC* [CRC Press (Taylor & Francis)] in 2008.

Kyprianos Papadimitriou received the Diploma, M.Sc., and Ph.D. degrees in electrical communication engineering from the Technical University of Crete, Chania, Greece, in 1998, 2003, and 2012, respectively.

He is currently a Scientific Staff Member with the Technical University of Crete, and a Research Associate with the Technological Educational Institute of Crete, Heraklion, Greece. He was with FO.R.TH and ATMEL, San Jose, CA, USA. His current research interests include system architecture and design, reconfigurable computing, run-time systems, interconnection networks, and real-time systems. He holds 38 scientific publications and one U.S. patent, and has participated in several EU-funded and nation-wide projects.

Polydoros Petrakis received the B.Sc. degree in computer science from the University of Crete, Heraklion, Greece, in 2009.

Since 2011, he has been with the Technological Educational Institute of Crete, Heraklion, Greece, researching on research and development projects. His current research interests include embedded computing, multicore systems, and interconnection networks and simulation tools, such as SystemC and gem5.

Antonis Papagrigoriou received the Diploma and M.Sc. degrees in electrical communication engineering from the Democritus University of Thrace, Xanthi, Greece, in 2000 and 2002, respectively.

Since 2004, he has been a Software Engineer with the Department of Information Technology, Forthnet, Athens, Greece. He is involved in research projects with the Technological Educational Institute of Crete, Heraklion, Greece, with particular emphasis on SystemC models, high-level synthesis, and GNU/Linux driver development, for five years.

George Kornaros received the Diploma degree in computer engineering from the University of Patras, Panepistimioupoli Patron, Greece, in 1992, the M.Sc. degree in computer engineering from the University of Crete, Heraklion, Greece, in 1997, and the Ph.D. degree from the Technical University of Crete, Chania, Greece, in 2013.

He is currently an Assistant Professor with the Technological Educational Institute of Crete, Heraklion. He was a System Architect of a few single-chip network processor designs for industry. His current research interests include multicore architectures, high-speed communication architectures, networking systems, embedded and reconfigurable systems, and both full- and semi-custom IC design. He has published 40 technical articles and has edited the book entitled *MultiCore Embedded Systems* (CRC Press, Taylor & Francis) in 2010.

Prof. Kornaros is a member of the Technical Chamber of Greece.

Ioannis Christoforakis received the B.Sc. and M.Sc. degrees from the Technological Educational Institute of Crete, Heraklion, Greece, in 2012 and 2014, respectively.

He is currently a Researcher with the Technological Educational Institute, researching on multicore architecture, network-on-chip, and embedded and reconfigurable design/verification using EDA tools. He has co-authored ten technical publications.

Othon Tomoutzoglou received the B.Sc. degree from the Technological Educational Institute of Crete, Heraklion, Greece, in 2014, where he is currently pursuing the M.Sc. degree.

He is curenly a Design Engineer in research projects with the Technological Educational Institute of Crete. His current research interests include multicore and heterogeneous architectures, embedded and reconfigurable systems, RTL Design, high-level synthesis, and operating systems. He has published recently on runtime adaptation of embedded tasks.

George Tsamis received the B.Sc. and M.Sc. degrees from the University of Crete, Heraklion, Greece, in 2009 and 2012, respectively.

Since 2010, he has been a Researcher with the Technological Educational Institute of Crete, Heraklion. His current research interests include 3-D data visualization, mobile applications, software engineering, information retrieval, and real-time embedded systems, especially bandwidth regulation for mixed criticality systems.

Marcello Coppola received the M.Sc. degree from the University of Pisa, Pisa, Italy, in 1992.

He is currently an Advanced Architecture and Innovation Director with Digital Sector Group, STMicroelectronics, Grenoble, France, in charge of several projects related to STNoC technology and MPSoC, with particular emphasis to architecture, modeling, verification, network-on-chip, and programming models. He has published different books, over 70 technical articles, and holds a number of patents.