Security Effectiveness and a Hardware Firewall for MPSoCs

Miltos D. Grammatikakis*, Kyprianos Papadimitriou*, Polydoros Petrakis*, Antonis Papagrigoriou*, George Kornaros*, Ioannis Christoforakis*, Marcello Coppola†

* Technological Educational Institute of Crete, GR

† STMicroelectronics, IT

***Abstract*— There is a constant increase in the interest shown for trusted computing in the embedded domain. In an MPSoC each processing element such as a CPU could request accessing any physical resource of the device such as a memory or an I/O component. Along with normal requests, malevolent ones could occur produced by malware applications or processes running in one or more CPUs. A protection mechanism is required to prevent injection of malicious data across the device, e.g. unsafe data written by a CPU into a memory address, which are read later by another CPU. A considerable amount of research has been devoted in security for MPSoCs, but limited work exists in performing protection at the source instead of the target, thus cutting-off malicious content at an early stage prior to entering the on-chip network.**

In the present work we focus on the side of the CPU connected to the SoC network. We are envisioning a self-contained NoC firewall, which by checking the physical address of a request to a memory-mapped device against a set of rules, rejects untrusted CPU requests to the on-chip memory, thus protecting all legitimate applications running in a shared-memory SoC. To sustain high-performance we implemented the firewall in hardware, while rule-checking is performed at segment-level based on deny rules. To evaluate the impact of security mechanisms we developed a novel framework based on gem5, coupling ARM technology and an instance of a commercial point-to-point interconnect from STMicroelectronics called Spidergon STNoC. Tests include several scenarios with legitimate and malicious processes running in different CPUs requesting access to shared memory. Preliminary results show that the incorporation of a security mechanism in the network interface can have a positive effect on network performance by reducing both the end-to-end delivery time of packets, and the power consumed from unnecessary transmissions. From the network aspect, this effect is independent of the performance of implementation itself, e.g. either a hardware or a software solution equally relieves the network from unnecessary loads. Finally, we compare the performance of our hardware approach over a simple equivalent software solution. Certainly, this comparison favours hardware by considerable margins, however we use it only as reference to illustrate the merit from implementing protection in hardware.

The purpose of the present study is three-fold. First, we present the proposed hardware NoC firewall. Then we examine the effect on network transmissions from incorporating a security mechanism in the network interface; to do this we developed a novel framework. Finally, we include preliminary performance results of our NoC firewall and a simple yet indicative comparison with a software solution.

I. Introduction

The complexity of modern Operating Systems - large number of code lines, developed by different groups - raises different security vulnerabilities resulting from software misbehaviors. These are exploited by cyber-criminals attempting to subvert the security mechanisms supported by the OS in order to get control of the device and data. For instance, overwriting data or function pointers, dynamic memory allocation (double-freeing, referencing or writing to free memory), zero-length allocations, and buffer overflows are well-known techniques to bypass any security protection imposed by the OS. 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 communication architectures for interconnecting SoC components, i.e. NoC [2] or bus [3], have been widely studied during the last decade, but security aspects have recently attracted the interest of researchers. Releases of bug reports show that defending an MPSoC against attacks remains an important issue. The lack of proper and efficient isolation of source code and data among trusted and untrusted applications/processes constitutes one of the main challenges in shaping a secure architectural solution without jeopardizing performance.

We propose a hardware security module to manage the memory requests invoked by any initiator connected to the on-chip-network. Initiators can be CPUs, DMA controllers or peripherals requesting access to any memory location including memory-mapped registers. In the present work we conduct experiments using only CPUs as initiators. The security module is integrated in the network interface (NI) connecting the CPU to the NoC. It acts as a firewall checking the CPU accesses to memory against a lookup table containing the access rights; this way it prevents malicious content from entering the on-chip network. Protection takes place after virtual-to-physical address translation, thus rule-checking is applied on the physical address. The firewall ensures protection of the memories distributed across the device that are accessed by multiple on-chip processors, thus it should be incorporated in the NI of each CPU. We assume that in each CPU several independent processes are initiated; for example each process could be related with a completely different application such as a billing service, voice transmission, multimedia, gaming etc, each of which can request accessing any memory in the SoC. Our approach allows to isolate different applications or processes running even on the same CPU of the SoC. This is important as for example a malicious process invoked in a CPU is restricted from writing an address also allocated by a legitimate process running either on the same or a different CPU of the device.

Protecting an embedded system from malicious attacks has consequences in both HW and SW design, as well as on physical characteristics such as performance, area cost, and power. The authors in [4] used these factors, as well as the memory cost, as metrics to evaluate their approach in protecting the memory in embedded systems. Towards a similar direction, we study whether a security mechanism results in diminishing returns, such as incurring overhead to network delay and power consumption. We show that in the presence of malicious processes a security mechanism accounts for the reduction of end-to-end transmission delays and power consumed in NoC. In addition, our concern is whether a mechanism added in the system for ensuring security, which examines continuously the data before they are actually transmitted, incurs significant overhead even in case a process does not convey suspicious content. Our contributions are:

- a hardware-based NoC firewall performing efficient rule-checking of memory requests at segment-level;
- a scalable modeling infrastructure based on gem5 relying on ARM and Spidergon STNoC, augmented with security features;
- experiments with multiple legitimate and malicious processes requesting access to memory.

The remaining paper is organized as follows. Section II overviews the related work with regard to hardware security. Section III describes the components and functionality of the proposed hardware NoC firewall. Section IV presents the framework we developed to model the problem and conduct the experiments. Section V deploys the different scenarios to examine the effect of a security mechanism when both legitimate and malicious requests occur. In Section VI, we evaluate the effect of a security mechanism on the delivery time of the packets and the power consumption of the network. In that Section we also compare the performance of our HW approach against a SW solution used for reference purposes. Finally, Section VII summarizes the paper and presents our future work.

II. Background and Motivation

In the past, NoC research has focused on QoS topics, while security is rarely mentioned. Only recently there have been efforts to consider NoC security in the context of shared memory MPSoCs with high performance and power-efficient circuits for authentication, confidentiality and integrity support. These specialized hardware security modules can be used to thwart attacks to external physical memory, mitigating risks associated to denial-of-service (DoS), side-channel attacks, malware intrusion, or buffer overflow vulnerabilities [5], [6].

In addition to crypto-based solutions, NoC firewall mechanisms provide domain protection by controlling memory access rights through hardware multi-compartment isolation [7]–[10], 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. These software methods are often supported by low-level hardware primitives and associated driver functions.

Most previous memory protection schemes propose fine-grain page-level security. Hence, decisions are made on whether to accept or reject a specific request based on rules hidden within a page memory descriptor or via an independent memory unit. Unlike fine-grain page-level protection, our work considers for the first time coarse-grain, segment-level protection based on the physical address of the NoC transaction request. Thus, the proposed NoC firewall, not only supports isolation by extending the (end-to-end) network interface layer, but also attempts to harmonize the system security infrastructure by implementing the NoC firewall module as a new distributed service of the network interface.

Unlike the hardware memory protection module of Porquet [9] and Wiggings [10] which consider *page-level implementations* based on MMU security, the proposed NoC firewall relies on *segment-level protection*. Since rules are configured and used directly at the network interface, the proposed scheme has relatively less area overhead and latency. Moreover, by avoiding caching, which involves a complex table walk mechanism to fetch rules from external memory to an internal table-lookaside buffer (TLB), isolation of mixed-criticality applications with hard real-time constraints can be realized.

Similar to the authors in [7], [8], our NoC firewall is integrated in the network interface. However, while [7], [8] use dedicated virtual channels to pass specialized rule-checking information based on processor and thread identifiers, we consider generic segment-level rules based on the NoC transaction's *physical address bits* (excluding offset). In addition, while [7], [8] compare different initiator and target implementations of the memory protection module in terms of area cost and power consumption, we concentrate *only on the initiator side* (assuming that processor requests rather than memory accesses must be protected).

Finally, the proposed segment-level protection approach supports efficient hardware-based multi-compartment philosophy, thus *extending existing protection mechanisms in commercial multicore SoC technologies*, such as ARM v7 TrustZone or derivative solutions, e.g. SamSung Knox Workspace. More specifically, ARM TrustZone currently defines *only two hardware-supported security domains (secure and non-secure)* identified using a (special) NS bit available in *all memory page descriptors* [11], a significant overhead in today's commercial multi-faceted isolation solutions.

III. The Hardware NoC Firewall

We propose a lightweight, non-intrusive security module composed of a novel hardware IP (and corresponding driver components) providing isolation in integrated NoC-based MPSoC solutions. It is placed between a Network-on-Chip and a system component, e.g. CPU, memory controller or hardware accelerator, ideally at the network interface (NI). It acts as a firewall between the NoC and the system component, adapting concepts from enterprise network firewalls to the level of the on-chip communication architecture of MPSoC devices. Typically, there are multiple NoC firewalls distributed in the system, i.e. one for each IP connected to the Network-on-Chip. Within each NoC firewall component different access rules are defined, thus regulating access to the protected memory regions.

Executable system-level specifications have been developed for a coarse-grain NoC Firewall solution which is implemented at the NI layer of a Network-on-Chip (NoC). We have developed and validated both a cycle- and bit-accurate SystemC virtual prototype model, and a higher-level transaction-level (TLM) gem5 model, which we further time-annotated from SystemC results.

Below, we examine the architecture and behavioral-level characteristics of the NoC firewall. It operates at the NI layer by:

- tagging NoC transactions with a 6-bit field (called PID) that is mapped to a specific linux process identifier,
- statically or dynamically configuring rules for user processes to access contiguous physical address regions (either memory pages or generic segments),
- 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 potentially compromised process cannot access data owned by another secure process, thus successfully subverting Denial-of-Service attacks from malware applications and/or corrupt DMA engines.

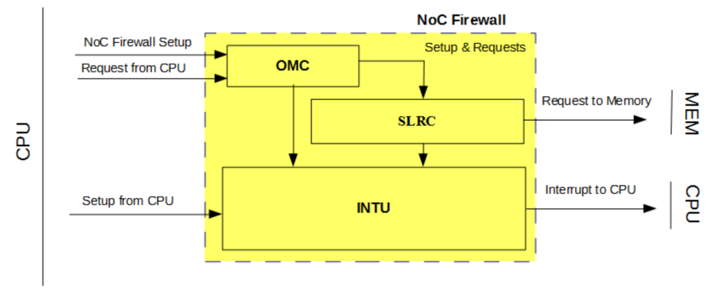

Fig. 1. The top-level NoC Firewall module.

Figure 1 shows the top-level architecture. It consists of the following submodules:

- The Operating Mode Controller (OMC), which accepts, decodes and dispatches NoC firewall commands.
- The Segment-Level Rule-Checking module (SLRC), which processes incoming memory access transaction requests and most configuration commands; notice that within the SLRC, a number of memory structures are responsible for implementing deny rules at at segment-level with a deny policy (we assume an allow-by-default mode). The SLRC also includes internal monitors implemented with timers and event counters to record NoC-related activity, and report performance and/or security issues to the Interrupt Unit.
- The Interrupt Unit (INTU), which accepts in parallel interrupt requests from the OMC block (e.g. invalid commands) and the SLRC block (resulting from rule-checking), and reports interrupt contexts to the CPU Interrupt Unit.

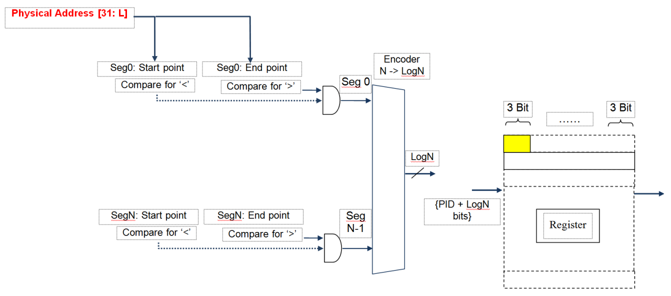

Fig. 2. Range search data structures using register-based parallel search.

As shown in Figure 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 comparators and register sets that specify the startpoint and endpoint of each segment. Search within these segments relies on comparators, which check the physical address of the incoming access requests after extracting lower order bits that correspond to the offset L; the value of L is 12 bits for the ARM v7 architecture.

This process involves checking whether the incoming physical address is in the preconfigured address range of a segment. We have implemented a policy based on deny rules. Thus, if there is no match in the segment structures specified by the corresponding (start, end) points, then normal access is allowed to the physical address. Otherwise, if an incoming physical address falls in a particular segment address range, then this match implies that the physical address belongs to that particular segment, and hence the rule table needs to be accessed to retrieve and test for the specific rule. Each segment rule is 3-bits long, specifying access control based on any combination of read, write and execute privileges.

In Figure 2 a 6-bit process identified (PID) is used 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. The PID can be dynamically provided by the OS serving a multiprocess environment.

It is implementation-dependent (based on silicon cost) whether a CAM structure is faster and/or less complex than the register-based comparators shown in Figure 2. Based on RTL simulation results, it appears that if the number of supported segments is small (below 128), then the proposed solution based solely on parallel comparisons using registers is adequate in terms of performance. This option is much more flexible, since it allows easier 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 configuration of network interface [12].

IV. Experimental Framework

To evaluate the effect of security in MPSoCs we created a framework combining CPUs and shared memory, interconnected with a network-on-chip topology. Furthermore, we enhanced the framework by integrating the functionality of the NoC firewall described previously. Currently, within the framework we can measure end-to-end delivery times and power consumption at the network-layer. The NoC firewall is integrated at the network interface that lies on a higher layer, i.e. transport-layer of the OSI model, therefore its overhead is not “seen” in the above measurements. Hence, we do not yet study the effect of NoC firewall on system-level performance. Instead, within the context of the framework it is viewed only as a security mechanism, and we evaluate the effect from activating/deactivating it on the data delivery time and power consumption at network-layer and below, i.e. link-layer and physical-layer. In the future we will include the capability of performing measurements not only at the network-layer, but also at the application-layer. This will allow to perform measurements from the time a request is generated at application-layer, until it reaches its destination.

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; network interface (NI) that functions as access point to the interconnect, simple non-programmable router, and physical link. STNoC relies on network operation, thus it provides programmable services such as changing the routing information and quality of service (QoS). It is equipped with a backpressure mechanism for regulating the traffic around the network. STNoC can be customized according to the given specifications to efficiently connect MPSoC components, i.e. CPUs, memories, and peripherals [13]. Figure 3 shows an example topology for attaching different components to the STNoC. From the network point of view, each component is a node, and information - instructions and data - is transmitted across the network in packets. Prior to transmission a packet is broken down in smaller units called flits, which in turn 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 customization procedure, along with the topology, a designer should also tune-up these parameters; the link width, the packet size and the number of flits comprising each packet. For example, for an STNoC with 16 Bytes link-width, if the selected packet is 128 Bytes (including header), it is split up in 8 flits.

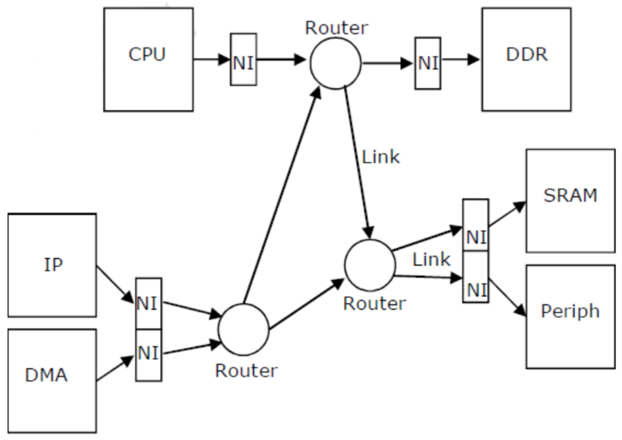

Fig. 3. Example of STNoC topology interconnecting the MPSoC components. STNoC consists of three basic building ingredients: network interface (NI), router, and link.

To conduct experiments we described the above system in the gem5 simulator [14], a platform widely used in computer system architecture research, encompassing processor and system-level microarchitecture. It is highly-configurable and includes support for multiple ISA and diverse CPU models including ARM, with detailed memory systems and interconnect models. Within the gem5 environment, we created an STNoC-like network model instance, which we further enriched with time-annotation derived from a cycle-accurate RTL model. Also we implemented a gem5 transaction-level model of the proposed NoC firewall and integrated it in the network interfaces of STNoC. Then, we time-annotated the NoC firewall model based on preliminary results from a cycle- and bit-accurate design implemented in SystemC.

To study the system reaction under different conditions we developed a method that generates legitimate and malicious requests within the gem5 simulator. We consider two type of processes that request access to memory. 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 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 the ARM Fast Models simulator environment [15]. This environment provides a programmer’s view (PV) of ARM processors. In general, PV models are used to confirm software functionality, but cycle counts or low-level component interactions are not accurately modeled. The ARM FastModels is a functionally accurate environment in which ARM v7 CPU models are implemented as instruction set simulators. It simulates the ARM Cortex-A9 architecture by loading a linux image, so we were able to perform full system simulation by executing linux. The functional behavior of the model is equivalent to real hardware, however timing accuracy is sacrificed for the sake of efficient simulation.

The S_i linux process allocates an array of 4096 integers and then continuously writes data to array elements in an infinite loop. This enabled examining linux page tables and obtaining the memory descriptors that correspond to specific (virtual) array accesses at the moment they occur.

Along with 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 [16]. 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 FastModels. The corresponding descriptors from linux pagemaps analysis of the S_i process correspond to the heap range from 0x11000 to 0x36000; notice that 5 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).

Recapitulating, the above framework relies on ARM CPUs and shared memory distributed across the MPSoC, interconnected with the STNoC. We experimented with different configurations by varying the following parameters:

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

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

The above framework enables examining the effect of security on network load and power. This is the consequence from controlling the requests that are eventually released to the STNoC. To model the MPSoC primitives we used the gem5 simulator and ARM Fast Models. To obtain cycle-approximate results we time-annotated the STNoC and NoC firewall models with data gathered from cycle- and bit-accurate simulation in HDL and SystemC respectively. The latter feature cannot be used here for the experiments, but we will use it once we add the capability for performing measurements at the application-layer. In addition, we used the Orion 2.0 model [17], which is encapsulated in gem5, to evaluate the power consumed at the NoC. Our framework can be used to evaluate more intricate networks with more CPUs and memories. Currently, a restriction of our approach is that the distribution of the processes running in the CPUs is uniform, i.e. all CPUs run the exact same number of legitimate and malicious process. We are planning to fix this, and once we completely verify the framework in terms of timing compatibility (cycle-accurate instead of cycle-approximate), releasing it to the research community.

B. Type of Measurements

Currently, the framework allows to study the time a packet remains in the NI queue until it reaches its destination. This includes the time needed for breaking down the packet in flits. A flit passes through one or more routers, which store the flit in their queue and relay it to the proper path. In the future, we would like to conduct measurements not only at network-layer but also to measure delays in different phases, starting from the time a message is generated by a process at the application-layer. This will allow to take into account the time overhead of the NoC firewall.

V. Experimental Scenarios

We experimented with different number of nodes connected to the STNoC as shown in Figure 4; for each case we generated different combinations with regard to the number of safe and malicious processes.

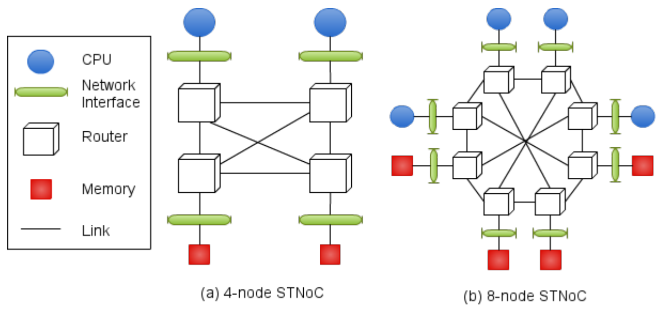

Fig. 4. The different configurations we used to study the effect of security. Case (a) corresponds to two different configurations; either both CPUs or only one CPU can be active.

Table I has the parameters of the simulation framework. We ran simulations with 1 CPU/2 Memories (actually 2 CPUs but 1 is inactive), 2 CPUs/2 Memories, and 4 CPUs/4 Memories. We tested different scenarios with regard to the number of safe and malicious processes running in the CPUs; in Table I we represent this as $xSyM$, where x is the number of safe processes and y the number of malicious processes. The process that requests 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 address, which belongs to one of the five allocated memory segments. We performed the same experiments by enabling and then disabling the NoC firewall. When enabled, the NoC firewall denies access to every single request from 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 network interface, which results in prohibiting unnecessary load from entering the NoC. On the other hand, when the NoC firewall is disabled, malicious requests are also transmitted to the NoC.

A setup-phase is needed to configure the rule table in each NoC firewall. For our experiments we do this with 5 CPU

TABLE I
SIMULATION PARAMETERS

Parameter	Type/Value
Interconnect topology	Spidergon STNoC
CPU type	ARM v7 Cortex A-9
Nodes setup	1CPU/2MEM; 2CPU/2MEM; 4CPU/4MEM
Processes per CPU (Safe,Malicious)	4S1M; 3S1M; 2S1M; 1S1M; 1S2M; 1S3M, 1S4M;
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
NI, Router, Link delay (in cycles)	{1, 2, 0}
Status of security mechanism	enabled; disabled

commands - one per segment - for every malicious process. For example, for 3 malicious processes, during configuration phase each CPU will send $3 * 5 = 15$ commands, resulting to a total of 15 deny rules. This way, during normal operation the firewall will block a malicious process from accessing the protected segments. No setup rules are needed for the legitimate processes since the operation of NoC firewall relies on deny rules only. During setup-phase, the packet injection rate is lowered due to gem5 limitations when using $Network\_test$ cache coherence protocol; this is because it cannot guarantee packet arrival from the CPU to the network interface. Since the environment in which we perform the experiments is under our control, we ensure that the setup-phase is completed before the processes start issuing memory requests.

By selecting different parameter values we performed experiments for several configurations. In specific, 7 different combinations of safe and malicious requests, 3 different node setups, and 2 options for the security mechanism (enabled/disabled), give a total of $3 \times 7 \times 2 = 42$ different configurations. Regarding NoC parameters, the packet size of a write command is 72 bytes, while we configured a 16 Bytes link-width, i.e. 16 bytes at most can traverse the link per transaction. Hence, each packet is split up in 5 flits since $72/16 = 4.5$.

VI. EFFECT OF SECURITY AND NOC FIREWALL PERFORMANCE

In the present Section we show how we use the above framework to evaluate the effect of security on the end-to-end delivery time and power at the network-layer. We recall, that this study is not affected by the performance characteristics of the security component. Then, we compare the performance of our proposed NoC firewall against a SW equivalent solution.

A. *Effect of Security on Network Transmission*

Our framework enables measuring the end-to-end delivery times at the network-layer, and the power consumption on the routers and links. In particular we measure two type of delays; the time a packet remains in the NI queue before it is actually released to the network, and then, after the flit leaves the NI queue, the time spent to traverse the network until it reaches its destination. The second delay includes the time for the flit to pass through all routers of the path. Table II has the results for a specific configuration when NoC firewall is either disabled or enabled. The last column shows the impact from activating the firewall. In all cases activation of security affects positively the network, i.e. negative values indicate improvement.

TABLE II
EFFECT OF SECURITY ON QUEUE DELAY, NETWORK DELAY AND POWER AT THE STNOC LEVEL. SCENARIO CONCERNS A 2CPU/2MEM SETUP WITH {1S2M} REQUESTS.

Metric	without firewall	with firewall	+/-(%)
NI queue delay	17.894 cycles	5.973 cycles	-66.62%
Network traversal delay	7.588 cycles	6.619 cycles	-12.77%
Router power (total)	0.2345 Watt	0.1794 Watt	-23.51%
Router clock power	0.0958 Watt	0.0958 Watt	0%
Router static power	0.0594 Watt	0.0594 Watt	0%
Router dynamic power	0.0792 Watt	0.0241 Watt	-69.60%
Link power	0.0113 Watt	0.0034 Watt	-69.91%

The chosen case concerns the $2CPU/2MEM$ node setup, with 1 safe and 2 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 decreases. This is due to that malicious requests are prevented from entering the network, thus fewer packets are released, resulting in smaller network traffic and less busy queues. Moreover, the total power of routers decreases by -23.51%, mainly due to the drastic decrease in the dynamic power. Finally, the use of firewall reduces the power consumed at the network links by almost 70%.

The above analysis indicates that security relieves the network from unnecessary load in the presence of malicious processes. The values are average ones and concern all CPUs. We performed experiments for two type of messages, e.g. "write" and "write-execute". In our tests the amount of flits transmitted was 1130 flits when firewall was on, and 3715 flits when firewall was off. Below, we concentrate on measuring the delays for all configurations of Table I, and we do not include further results on power consumption. We are planning to do this in our future work. Nevertheless, with the above use case we demonstrated the way the framework is used to evaluate the effect of security on the power consumption of NoC.

Figure 5 shows the total NoC delay (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 NoC firewall is enabled and disabled. The total NoC delay is computed as the average time required for a flit to remain in the queue of the network interface, 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. Thus, the more the malicious processes, the smaller the NoC delay becomes. For example,

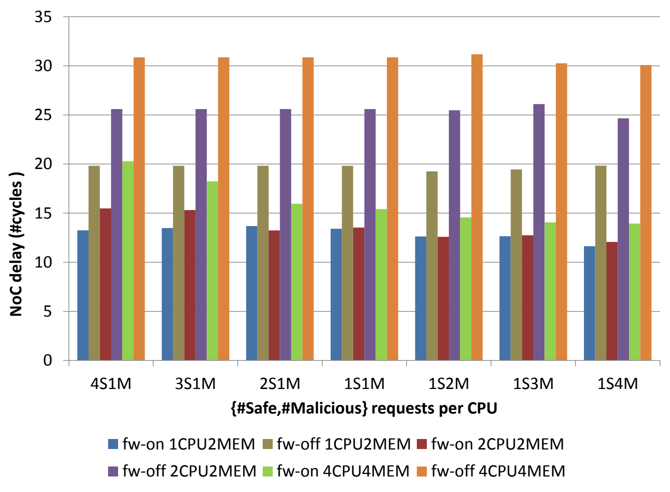

Fig. 5. NoC delay for all setups of Table I, when firewall is on or off.

we observe that in case 4 malicious processes are invoked per CPU - in the Figure this is represented with $1S4M$ - the NoC delay when security is active has the smallest value. It is also observed that when security is off, NoC delay remains unchanged regardless of the chosen setup. This is due to that requests are always served, thus they are always transmitted to the network.

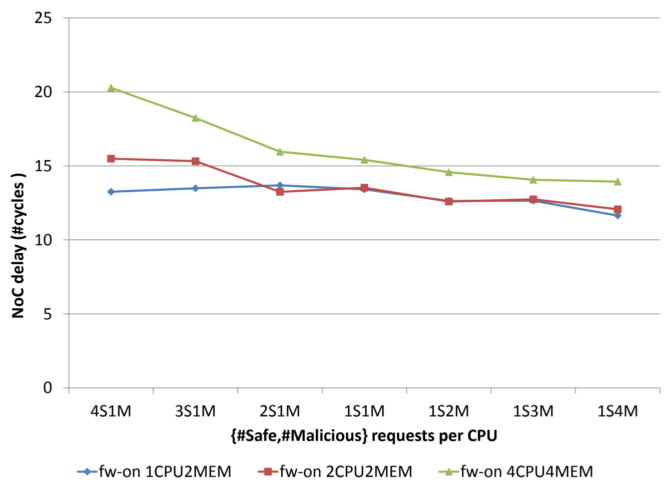

Fig. 6. NoC delay for all setups of Table I, when firewall is on.

Figure 6 has a different representation for assessing the effect of security. It concerns only the case where security is activated, covering all setups of Table I. We obtain that the more nodes are connected to the NoC, the bigger the NoC delay is. This is more obvious for large number of safe processes per CPU. In fact, it is observed that for 4S1M the gap of NoC delay between the $4CPU/4MEM$ setup and the other two setups increases.

B. *HW vs. SW implementation of the NoC firewall*

To asses 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 module in hardware. In particular, we compare the overhead of our hardware implementation versus an equivalent software implementation in ARM v7 Cortex-A9 processor. Results in Table III indicates that the merit from implementing protection in hardware is large.

TABLE III
TIME OVERHEAD OF NOC FIREWALL: HARDWARE VS. SOFTWARE IMPLEMENTATION

NoC firewall action	cycles in HW	cycles in SW
Initialization of 32 Segments	~160	62066-65904
Access Request	2 or 5	112-2127
Add Segment Table Entry	5	81-1019
Delete Segment Entry for a given PID	5	79-513
Delete All Segment Entries	5	1977-2411

To generate the ARM v7 assembly code, we cross-compiled the corresponding configuration functions and request access transactions using arm-linux-gnueabi-gcc and flags ”-c -g -Wa,alh,-ad -fverbose-asm” [18], [19]. After this step, we evaluated 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 was obtained from the technical architecture reference manual for Cortex-A9 [20].

VII. CONCLUSIONS

Due to the wide adoption of NoC-based MPSoC technology, security aspects and system address space protection services become of major concern. In this context, we implemented a hardware NoC firewall with deny rules statically configured at the network interface of all initiator nodes. This allows to prevent the injection of malicious requests at an early phase, thus prior releasing them into the NoC.

Using an instance of the industrial STNoC interconnect technology as the baseline architecture, we created a framework based on the gem5 environment which combines ARM architecture and security features. Within this framework, we study the effectiveness of security and reveal a significant reduction of the end-to-end delivery times and dynamic power consumption at the network level, especially when the relative number of malicious requests increases. Currently, we have extended the gem5 framework with time-annotated values from cycle- and bit-accurate SystemC in order to produce a realistic timing behaviour for both the STNoC and the NoC firewall. One weakness of our current setup is that it does not yet allow to study the effect of NoC firewall on the total system performance within the integrated framework. We are working on this so as to be able to measure the contribution of NoC firewall overhead as part of the total system overhead.

With regard to NoC firewall we consider several extensions. We examine the implementation of a hybrid-architecture combining segment- and page-level rule-cheking; the two rule-checking architectures can operate in tandem to perform both coarse- and fine-grain rule-checking. Another extension is to dynamically update the request types and deny rules in hardware so as to generalize the NoC firewall mechanism; this will enable the adaptation of memory protection solution to any predefined set of rules depending on the application domain.

Analysis is based on gem5 simulation results from a cycle-approximate STNoC model. Once tested for timing compatibility, our intention is that the gem5 STNoC model integrating the proposed NoC firewall protection mechanism will be released to the open gem5 community.

Acknowledgment

This work is supported by the EU through the FP7 TRESCCA project (GA No. 318036).

References

[1] M. Coppola, M. D. Grammatikakis, G. Kornaros, and A. Spyridakis, "Trusted Computing on Heterogenous Embedded Systems-on-Chip with Virtualization and Memory Protection," in *4th International Conference on Cloud Computing, GRIDs, and Virtualization*, May 2013.

[2] M. Coppola, R. Locatelli, G. Maruccia, L. Pieralisi, and A. Scandurra, "Spidergon: a Novel On-Chip Communication Network," in *4th International Symposium on System-on-Chip*, Nov 2004.

[3] M. Mitić and M. Stojčev, "A Survey of Three System-on-Chip Buses: AMBA, CoreConnect and Wishbone," in *ICEST*, 2006.

[4] R. Vaslin, G. Gogniat, J.-P. Diguet, R. Tessier, D. Unnikrishnan, and K. Gaj, "Memory Security Management for Reconfigurable Embedded Systems," in *Proc. International Conference on Field-Programmable Technololgy (FPT)*, Dec 2008, pp. 153–160.

[5] C. H. Gebotys and Y. Zhang, "Security Wrappers and Power Analysis for SoC Technologies," in *Proc. 1st IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2003, pp. 162–167.

[6] S. Lukovic and N. Christianos, "Enhancing network-on-chip components to support security of processing elements," in *5th Workshop on Embedded Systems Security (WESS)*, July 2010, pp. 1–9.

[7] L. Fiorin, G. Palermo, and C. Silvano, "A Security Monitoring Service for NoCs," in *Proc. 6tj IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2008, pp. 197–202.

[8] L. Fiorin, G. Palermo, S. Lukovic, V. Catalano, and C. Silvano, "Secure Memory Accesses on Networks-on-Chip," *IEEE Transactions on Computers*, vol. 57, no. 9, pp. 1216–1229, 2008.

[9] J. Porquet, A. Greiner, and C. Schwarz, "NoC-MPU: a Secure Architecture for Flexible Co-hosting on Shared Memory MPSoCs," in *Proc. Design Automation and Test in Europe*, July 2011, pp. 591–594.

[10] A. Wiggins, S. Winwood, H. Tuch, and G. Heiser, "Legba: Fast Hardware Support for Fine-Grained Protection," in *8th Asia-Pacific Conference on Advances in Computer Systems Architecture (ACSAC)*, July 2003, pp. 320–336.

[11] R. G. D.H. Mansell and S. Bile, "Data processing apparatus and method for controlling access to secure memory by virtual machines executing on processing circuirty," United States Patent, Tech. Rep. US20090222816, sept 2009.

[12] R. Panigrahy and S. Sharma, "Sorting and searching using ternary cams," in *IEEE Micro Magazine*. IEEE Computer Society, 2003.

[13] G. Palermo, C. Silvano, G. Mariani, R. Locatelli, and M. Coppola, "Application-Specific Topology Design Customization for STNoC," in *Proc. 10th Euromicro Conference on Digital System Design Architectures, Methods and Tools (DSD)*, Aug 2007, pp. 547–550.

[14] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The Gem5 Simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, 2011.

[15] N. Rodman, "ARM Fast Models-Virtual Platforms for Embedded Software Development," *Information Quarterly Magazine*, vol. 7, no. 4, pp. 33–36, 2008.

[16] "Linux community. pagemap collection, analysis and statistics-linux page tables," http://www./, [Online; accessed May 2014].

[17] A. B. 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. of the Conference on Design, Automation and Test in Europe (DATE)*, Apr 2009, pp. 423–428.

[18] http://www.ibm.com/developerworks/linux/library/l-arm-toolchain/, [Online; accessed May 2014].

[19] https://sourcery.mentor.com/sgpp/lite/arm/portal/subscription, [Online; accessed May 2014].

[20] ARM, "ARM Cortex-A9 Technical Reference Manual (Appendix B: Cycle Timings)," ARM, ARM, Tech. Rep., 2012.