ML-Assisted Attack Detection on NoC-Based Many-Cores Through On-Chip Traffic Monitoring

Andrea Galimberti*, Rohan Purkait†, Nahian Islam†, Amlan Ganguly†, Mark Indovina†,
Michael Zuzak†, Sai Manoj Pudukotai Dinakarrao‡, Davide Zoni*, William Fornaciari*
*DEIB, Politecnico di Milano, Milano, Italy, email: {andrea.galimberti,davide.zoni,william.fornaciari}@polimi.it
†Rochester Institute of Technology, Rochester, NY, USA, email: {rp1982,mi4525}@g.rit.edu,{axgeec,maieee,mjzeec}@rit.edu
‡George Mason University, Fairfax, VA, USA, email: spudukot@gmu.edu

***Abstract*—Multi- and many-core processors based on a network-on-chip (NoC) interconnect are pervasive in computing platforms ranging from server farms to embedded systems. Such complex systems often make wide use of third-party intellectual property elements from untrusted organizations. This manuscript proposes a methodology that combines on-chip traffic monitoring, through the insertion of lightweight counters in the NoC routers, and on-chip analysis, through machine-learning techniques, into a blue-team approach that detects the execution of unintended applications with an average accuracy of 89% and limited overheads in terms of area, power, performance, and timing.**

***Index Terms*—multi-core processor, network-on-chip, interconnect traffic, hardware performance monitors, machine learning, artificial neural network, attack detection**

I. Introduction

Modern computing platforms have evolved towards many-core architectures mixing general-purpose CPU cores and different hardware accelerators to overcome the slowdown in the improvement of their performance and efficiency with the end of Moore's law and Dennard scaling and to maximize the performance and energy efficiency of different workloads. Network-on-chip (NoC) interconnects are crucial in these large and complex platforms, connecting their cores with each other and to memory, caches, and peripherals.

NoC-based computing platforms have been the focus of extensive research aimed at thwarting the critical threats posed by the widespread adoption of third-party intellectual property (IP) cores from untrusted external entities in their design. On the one hand, the open literature contains a variety of works that studied the design of malicious hardware [1] and in particular of hardware Trojans (HTs) to attack NoC-based computing platforms [2], also employing machine learning (ML)-based techniques and generative adversarial networks to process the data collected by HTs and retrieve the applications under execution [3]. Conversely, a vast research effort has been devoted to the detection of such malicious hardware [4], e.g., by leveraging few-shot learning techniques [5]. On the other hand, it has been shown that malicious applications on multi-core system-on-chips (SoCs) and NoC-based platforms, including in multi-tenant scenarios, can leak information, sabotage systems, cause denial-of-service, and engage in snooping [6].

This work was supported by the European Union's Chips Joint Undertaking (Chips JU) program under grant agreement No. 101112274 (ISOLDE).

The open literature is missing, to the best of our knowledge, a methodology that exploits in a blue-team, i.e., defensive, approach the NoC traffic information and that leverages ML-based techniques to detect whether there is an attack ongoing on the monitored computing platform.

Threat model: We consider an adversary that aims to host, on one or more cores of a multi-core processor, an application that can be of any form. We assume the adversary can subvert the security provisions of the host processor and the OS scheduler and launch an arbitrary unintended process in the NoC. A successful countermeasure must detect the unintended application run by the adversary and notify the user to initiate a remediating action, e.g., killing the process. This is consistent with the threat models from prior works on NoC security [6]–[8]. In this paper, we focus on detecting anomalous applications through traffic monitoring over the NoC, the only trusted infrastructure in the system.

Contributions: This paper introduces a methodology that leverages NoC traffic information in a blue-team approach to detect the presence of unintended applications under execution on a NoC-based multi-core processor, assuming its NoC interconnect as the sole trusted element. It combines an on-chip monitoring infrastructure, which collects data related to the NoC traffic, with an on-chip analyzer based on ML techniques, that is tasked with the detection of unintended applications by employing solely such traffic data. The proposed methodology provides three main contributions:

1) *Effective detection* – It leverages information related to the number of packets traversing the various NoC routers to obtain an up to 89% average accuracy in detecting the execution of unintended applications.
2) *Limited overhead* – It requires solely the insertion of few small counters for on-chip traffic monitoring and a simple ANN-based analyzer, not affecting the performance and timing of the NoC interconnect and with minimal area and power overheads.
3) *Flexible solution* – It is effective also in scenarios with multiple processes running concurrently, with an accuracy drop of up to 10% on average, and we also evaluate how reducing the number of monitor-embedded NoC routers impacts the detection of unintended applications running in the system.

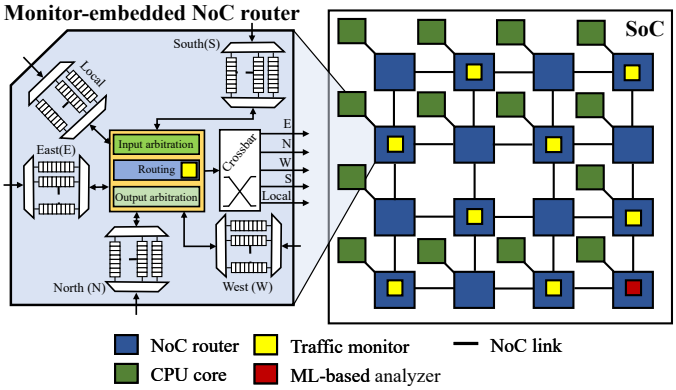

Fig. 1: Architecture of the proposed methodology, enhancing NoC routers with traffic monitors and the ML-based analyzer.

II. Methodology

A. On-chip traffic monitoring

The NoC interconnect is enhanced with monitors, optionally inserted in the routing block of each NoC router, that count the number of data packets traversing the latter. Each monitor is a n-bit counter that counts up each time a packet enters the monitor-embedded router. A separate n-bit down counter functioning as a timer maintains the observation window. Each monitor captures the number of packets passing through the router over the duration of a kernel which is typically in a 1-10 million cycles range [9]. This indicates that a 20-24 bit counter in each monitor is sufficient to measure the packet count. Data notably requires multiple clock cycles to traverse the switch as it consists of multiple flits [10], hence the monitor size is sufficient since the packet count will be less than the duration of the counting process.

The monitor does not interfere with the data path of the switch since it is not sequential to the routing logic and the counting occurs in parallel to the routing, thus the traffic monitoring mechanism does not impact the performance and the timing of the NoC interconnect. The monitor-embedded NoC routers packetize the counts utilizing the Network Interface (NI) wrappers [10] and transmit it to the on-chip detector using the NoC for analysis. The small area and power footprint of the monitoring logic constitutes a minimal overhead, compared to the area and power consumption of a multi-core NoC-based SoC. The additional logic to add on-chip traffic monitoring capabilities consists solely of few counters, while an NoC router can occupy around 30 to 40 thousands of gates [10].

As an example, Fig. 1 depicts a multicore processor, with 16 cores in a 4×4 2D mesh topology, enhanced with on-chip traffic monitoring capabilities by inserting monitors in 8 of the 16 NoC routers, selected according to a checkerboard pattern.

B. On-chip detection of unintended applications

Data from the monitors is processed by an on-chip ML-based analyzer, connected to one node of the NoC interconnect, to determine whether an unintended application is under execution. As an example, in Fig. 1, the on-chip ML-based analyzer is placed in the bottom right NoC router. The ML analyzer deploys an artificial neural network (ANN) to analyze the traffic characteristics data sent by the monitors. ANNs were shown to be capable of mapping complex patterns effectively and resilient to random noise and variations, outperforming other ML algorithms such as KNN, SVM, and threshold-based analysis in terms of accuracy in such traffic analysis problems [3]. Moreover, supervised ML techniques such as ANN achieve higher performance, in the presence of labeled data, compared to the more computationally complex unsupervised and reinforcement learning techniques [11]. In the proposed approach, as the defender has knowledge of the intended applications, we assume the presence of labeled data.

The dataset consists of a feature, corresponding to the packet count, per each NoC router with monitoring capabilities, and data from features corresponding to each intended application are labeled with that application's name. For example, our experimental evaluation considers a 16-core system, thus the dataset consists of 16 features, i.e., 16 packet counts for packets traversing the corresponding NoC routers in the system. The dataset is created in the same fashion as an actual system designer, who may be able to create it by either employing an emulator or a simulator of the real system during its design phase. Various permutations of the intended applications are executed on the simulator to create traffic traces to be used in the training dataset, equivalent to those that would be visible by the monitors in a real-world scenario, that consist of the number of packets traversing the monitor-embedded switches.

As the traffic patterns, i.e., the packet count, depend on the executing applications, the on-chip traffic, and the mapping of the applications to the cores, the observed patterns will not be constant for a given set of applications. Random noise with a negative binomial distribution is appended to the dataset to correctly construct a training dataset for the ANN that is representative of the variations owing to network congestion, avoiding over-fitting of the ML analyzer and augmenting the simulation data for each application to emulate a real system in the field. The binomial noise is generated such that it does not affect the distribution of the dataset dramatically, serving the purpose of a large dataset for the ANN as well as emulating noise in the observations generated by the concurrent presence of other processes in the system.

The three-layer ANN implemented by the analyzer is composed of 1) an input layer with a neuron for each monitor-embedded router and with a rectified linear unit (ReLU) activation function, 2) a hidden layer comprising 50 neurons and a ReLU activation, and 3) an output layer with a softmax activation function. Applying dropout to the hidden layer, with a 0.5 dropout rate, minimizes overfitting. Such ANN architecture is mainly suitable for multi-class classification, with the number of neurons aligning with the unique classes, and it is compiled using the *Adam* optimizer and the sparse categorical cross-entropy loss function. In the training phase, the dataset undergoes a preprocessing that includes data standardization and label encoding.

TABLE I: Detection performance in the baseline scenario.

Unintended app	Accuracy	Precision	Recall	F1 score
blackscholes	97.3%	98.0%	97.3%	97.3%
bodytrack	92.1%	90.5%	92.1%	90.9%
canneal	86.8%	88.2%	86.8%	85.5%
dedup	89.5%	88.3%	89.5%	88.2%
ferret	86.8%	86.5%	86.8%	85.3%
fluidanimate	86.8%	86.6%	86.8%	85.4%
freqmine	81.6%	85.3%	81.6%	78.8%
raytrace	94.7%	93.4%	94.7%	93.7%
streamcluster	89.5%	88.8%	89.5%	88.4%
swaptions	94.7%	93.0%	94.7%	93.5%
vips	86.8%	88.8%	86.8%	85.1%
x264	81.6%	85.1%	81.6%	81.5%
Average	89.0%	89.4%	89.0%	87.8%

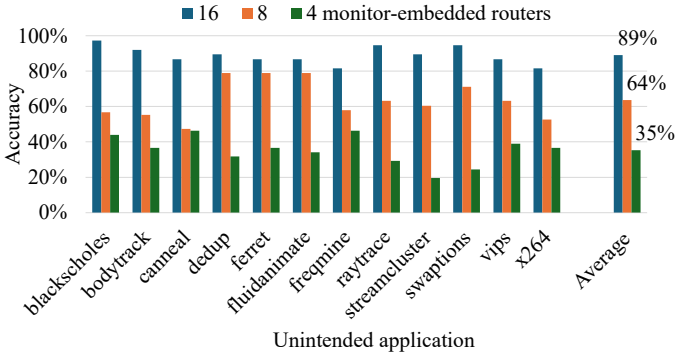

Fig. 2: Impact of variation in the number of monitor-embedded routers on the detection accuracy.

III. Experimental evaluation

A. Experimental setup

Target system configuration: The target architecture, simulated in the *gem5* cycle-level simulator [12], version *v22.1.0.0*, is a 16-core CPU with cores based on the x86 ISA and implementing *gem5*'s out-of-order detailed CPU model. The CPU cores are connected through single-cycle-latency routers of a 16-node NoC interconnect configured in a 2D 4×4 mesh topology with X-Y routing and implementing *gem5*'s Garnet network model. The Ruby cache memory and coherence model is configured in a two-level cache hierarchy with MESI cache coherence protocol. L1 caches, split into instruction and data caches, are private to each core, while L2 caches are shared among them. The simulated system includes a 3GB, 2400 MT/s, dual-channel DDR4 memory.

Simulator and workload: gem5 executes full-system simulations that boot the *Ubuntu 18.04* OS with a *Linux 4.19.83* kernel. The workload is constituted by 12 applications from the *PARSEC* benchmark suite [9]. One or more *PARSEC* processes are executed at a time, each possibly parallelized on multiple threads and assigned to a subset of the CPU cores, with each core executing up to one thread, by leveraging the *taskset* command from the *util-linux* package. The *gem5* simulations output the number of packets traversing each of the 16 NoC routers during the execution of the *PARSEC* applications.

ANN analyzer: The classifier for ML-based analysis is implemented by leveraging the *Keras* deep-learning framework. The training spans 50 epochs, with a batch size of 32, and the model's performance is evaluated through a set of performance metrics to showcase its suitability for classification tasks. The ANN has 12 output classes, one for each intended considered *PARSEC* application plus one class for the unintended applications. Based on the number n of applications expected or allowed to run concurrently in the system due to multi-tenancy, an n-hot encoding is adopted on the output layer to indicate which and applications are being hosted in the system. In the various considered scenarios, the traffic data including the execution of the applications designated as unintended is not used as part of of the training set for the ANN analyzer, while it is used instead in the testing phase.

B. Experimental results

Baseline scenario: We consider first the scenario in which a single application is hosted on the multi-core system at a time, with up to 16 threads running concurrently, each on a separate core of the multi-core CPU, and packet data is available for all of the 16 switches in the NoC. In particular, we evaluate 12 separate cases, one for each of the considered *PARSEC* applications. Each case consists of one application being classified as an unintended one: training is therefore performed on a dataset that includes the execution of all the other 11 application, while inference is then carried out on all 12 applications, including the unintended one, to test the performance of the ML analyzer.

Table I lists the performance metrics for the ML model in such scenario, with each case of a specific *PARSEC* application being the unintended one corresponding to a row in the table. The leftmost column reports indeed the unintended application in each of the 12 considered cases. The results of the experimental evaluation reported in Table I highlight average accuracy, precision and recall of 89% and an average F1 score of 88%, with all the 12 considered cases providing accuracy, precision, recall, and F1 score higher than 81%, 85%, 81%, and 78%, respectively.

Number of monitored routers: We explore scenarios where the number of monitor-embedded NoC routers is equal to 8 and 4, instead of all 16, whereas the number of apps running concurrently and marked as unintended ones are left unchanged.

The clustered bar chart in Figure 2 depicts how the top-1 accuracy, chosen for the sake of simplicity as a metric representative of detection performance, varies in the 12 cases, one per each application, by decreasing the number of monitor-embedded routers from 16 to 8 and 4. In particular, the results show that the accuracy declines as the number of monitor-embedded routers decreases for each of the 12 cases. The average accuracy reduces from 89% with 16 monitor-embedded routers to 64% and 35% with 8 and 4, respectively. We note that both the precision, recall, and F1 score metrics follow the exact same trend as the top-1 accuracy, thus we do not depict them and we do not discuss them separately.

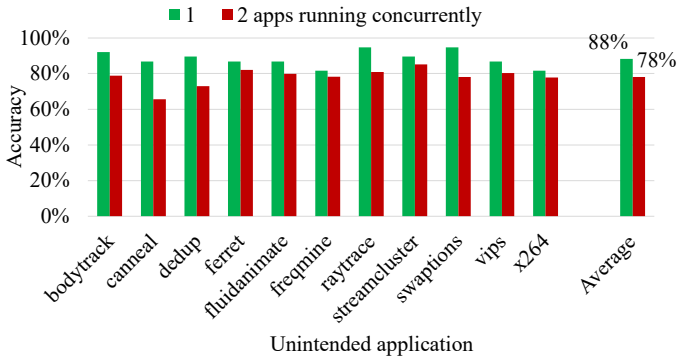

Fig. 3: Impact of variation in the number of applications running concurrently on the detection accuracy.

Number of apps under execution: We consider a scenario where we monitor all 16 NoC routers, two applications are under execution at a time, and their threads are running on distinct sets of CPU cores. We still consider 11 separate cases, each with a single unintended application from the *PARSEC* benchmark suite. The training dataset does not contain therefore data corresponding to the execution of pairs of applications that include instances of the unintended one.

Figure 3 compares the accuracy in this scenario with two applications executing concurrently, in which one application varies while the other is *blackscholes*, against the one obtained for the baseline scenario with a single application, whose performance metrics were previously listed in Table I. Each cluster of bars in Figure 3 corresponds to a use case where the unintended application is the one on the X-axis, both for the baseline scenario and the scenario with two applications running concurrently.

The results show a detection accuracy of more than 78% on average with 2 apps running concurrently, demonstrating that the proposed methodology is still effective even with multiple applications under execution on the system. Detection accuracy drops by between 3% and 21%, corresponding to cases in which *freqmine* and *canneal* are the unintended applications, respectively, compared to the baseline scenario with only one application running at a time. The precision, recall, and F1 score metrics follow the exact same trend as the top-1 accuracy, thus we do not depict and discuss them for the sake of brevity.

Overhead analysis: Each monitor occupies 2,325µm^2 and consumes 0.189µW of additional power, thus their power overhead contribution grows linearly with the number of monitor-embedded routers, reaching a maximum value of around 3µW when all 16 routers include a monitor. The on-chip ANN analyzer occupies instead an area of 34,448.79µm^2, with a power consumption of 6,299.3µW and a delay of 0.41ns. We note that the power consumption of the overall SoC is typically in the order of Watts, hence the power overhead due to on-chip monitoring and analysis is negligible.

We further remark that, since the monitors and the analyzer are not on the main datapaths of the routers the proposed methodology does not affect in any way the performance and the timing of the overall system. The packetized packet count transmitted from the monitors to the analyzer add negligible traffic to the application traffic as they are sent once at the end of a kernel execution.

IV. Conclusions

This paper described a methodology that detects unintended applications running on NoC-based computing platforms by leveraging on-chip monitoring of the NoC traffic and an ANN-based analysis. An extensive experimental campaign highlighted a detection accuracy of up to 89% on average, studied how embedding traffic monitors in different numbers of NoC routers impacts accuracy, and demonstrated the effectiveness of the methodology even with multiple applications running concurrently, with an average accuracy reduction of 10% compared to the single-application scenario,

References

[1] S. T. King, J. Tucek, A. Cozzie, C. Grier, W. Jiang, and Y. Zhou, "Designing and implementing malicious hardware," in *Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*, ser. LEET'08. USA: USENIX Association, 2008.

[2] R. JS, K. Chakraborty, and S. Roy, *Hardware Trojan Attacks in SoC and NoC*. Cham: Springer International Publishing, 2018, pp. 55–74. [Online]. Available: https://doi.org/10.1007/978-3-319-68511-3_3

[3] A. Dhavlle, M. M. Ahmed, N. Mansoor, K. Basu, A. Ganguly, and S. M. Pudukotai Dinakarrao, "Defense against on-chip trojans enabling traffic analysis attacks based on machine learning and data augmentation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 12, pp. 4681–4694, 2023.

[4] D. John, M. Matthew, S. Jared, T. Adrian, W. Adam, S. Simha, and S. Salvatore, "On the feasibility of online malware detection with performance counters," in *In ACM SIGARCH Computer Architecture News*, 2013.

[5] S. Kasarapu, S. Shukla, R. Hassan, A. Sasan, H. Homayoun, and S. M. PD, "CAD-FSL: code-aware data generation based few-shot learning for efficient malware detection," in *ACM Great Lakes Symposium on VLSI (GLSVLSI)*, 2022.

[6] D. M. Ancajas, K. Chakraborty, and S. Roy, "Fort-nocs: Mitigating the threat of a compromised noc," in *Proceedings of the 51st Annual Design Automation Conference*, 2014, pp. 1–6.

[7] R. F. Faccenda, G. Comarú, L. L. Caimi, and F. G. Moraes, "A comprehensive framework for systemic security management in noc-based many-cores," *IEEE Access*, vol. 11, pp. 131 836–131 847, 2023.

[8] M. D. Grammatikakis, K. Papadimitriou, P. Petrakis, A. Papagrigoriou, G. Kornaros, I. Christoforakis, O. Tomoutzoglou, G. Tsamis, and M. Coppola, "Security in mpsocs: A noc firewall and an evaluation framework," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 8, pp. 1344–1357, 2015.

[9] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: Characterization and architectural implications," in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 72–81. [Online]. Available: https://doi.org/10.1145/1454115.1454128

[10] P. P. Pande, C. Grecu, M. Jones, A. Ivanov, and R. Saleh, "Performance evaluation and design trade-offs for network-on-chip interconnect architectures," *IEEE Transactions on Computers*, vol. 54, no. 8, pp. 1025–1040, 2005.

[11] Y. Chen, M. Mancini, X. Zhu, and Z. Akata, "Semi-supervised and unsupervised deep visual learning: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 46, no. 3, pp. 1327–1347, 2024.

[12] 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," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, p. 1–7, aug 2011. [Online]. Available: https://doi.org/10.1145/2024716.2024718