Online Detection of Hardware Trojan Enabled Packet Tampering Attack on Network-on-Chip: A Bayesian Approach

Online Detection of Hardware Trojan Enabled Packet Tampering Attack on Network-on-Chip: A Bayesian Approach

Xiaohang Wang**††, Ge Cao*, Yiming Zhao‖, Yingtao Jiang†, Amit Kumar Singh‡, Mei Yang†, Liang Wang§, Yinhe Han¶, Fen Guo*

** State Key Laboratory of Blockchain and Data Security, Zhejiang University, China
* School of Software Engineering, South China University of Technology, China
‖ Department of Computer Sciences, Metropolitan State University of Denver, United States
† Department of Electrical and Computer Engineering, University of Nevada, United States
‡School of Computer Science and Electronic Engineering, University of Essex, UK
§School of Computer Science and Engineering, Beihang University, Beijing, China
¶Institute of Computing Technology, Chinese Academy of Sciences, China
††Hangzhou High-Tech Zone (Binjiang), Institute of Blockchain and Data Security, China

Abstract—Hardware Trojans (HTs), proven difficult to be detected and removed at the offline post-silicon stage, can secretly launch dangerous packet tampering attacks on the network-on-chip (NoC) of a many-core chip. In this paper, we present an online HT detection scheme that is based on continuous, on-the-fly assessment of how likely any single node in the NoC includes an HT. In specific, the scheme first collects the routing path information of any data packet flowing through the NoC. The probability of a node being infected with HTs will next be determined based on each packet's authentication result and this probability is iteratively updated through Bayesian analysis. A node shall be marked as a high-risk node, if its probability of infection exceeds a threshold, and all the high-risk nodes thus discovered will be bypassed by any future traffic. Since the proposed scheme only needs end-to-end authentication, as opposed to costly hop-to-hop authentication, the hardware overhead is kept low. To help further reduce the bandwidth and computation overheads, three approximate schemes are also proposed. Experiments have confirmed that the proposed HT detection methods can effectively locate the malicious nodes and thus reduce the infection rate to below 5%.

Index Terms—Networks-on-chip, online detection of hardware Trojan

This work was supported in part by the National Key Research and Development Program of China No. 2023YFB4404404, in part by the National Natural Science Foundation of China under Grants 92373205 and 62374146, in part by the key R&D programme of Zhejiang Province No. 2024C01012, in part by the by Ant Group through CCF-Ant Research Fund, in part by the Key Technologies R&D Program of Jiangsu (Prospective and Key Technologies for Industry) under Grant BE2023005-2, and in part by CIE-Smartchip research fund No. 2023-004.
Corresponding author: Xiaohang Wang.
E-mail addresses: xiaohangwang@zju.edu.cn (X. Wang), 1585440301@qq.com (G. Cao), yimingzhao3@gmail.com (Y. Zhao), yingtao.jiang@unlv.edu (Y. Jiang), a.k.singh@essex.ac.uk (A.K. Singh), mei.yang@unlv.edu (M. Yang), lwang20@buaa.edu.cn (L. Wang), yinhes@ict.ac.cn (Y. Han), csguofen@scut.edu.cn (F. Guo).

1 Introduction

The excessive proliferation of multi-/many-core chips has met with escalating security threats caused by hardware Trojans (HTs) that, unfortunately, are hard to be completely detected and removed at an offline circuit testing stage [1] [2]. These HTs typically have a small trace in the sea of transistors, and their extremely short on-time behaviors and operation modes make them particularly hard to get caught.

In the literature, various types of HTs that initialize and sustain diversified attacks [15] have been studied. Among them, the HTs that cause packet tampering attacks [5] are found particularly dangerous. In this case, the HT in a malicious node deliberately modifies one or a few data fields of a packet passing through it, resulting in false data, information leakage, and/or system performance degradation.

To defend against such a packet tampering attack launched by the HT-infected NoC routers, authentication methods (e.g., cyclic redundancy check and algebraic manipulation codes) are applied to check packet integrity at runtime, as in the cases of [7] [8]. These authentication methods can be broadly divided into two categories: (1) End-to-End (E2E) methods where packet integrity is checked solely at the destination end of a routing path (i.e., at the destination node), and (2) Hop-to-Hop (H2H) methods that inspect packets at each hop

along a routing path. E2E methods might fail to pin down the HTs that they are conditionally triggered. For instance, if an HT returns to hibernation after launching an attack, the E2E methods simply cannot detect the HT. H2H methods, on the other hand, are more capable of detecting these HTs, but at a high cost of power and area.

In this paper, we propose an effective and lightweight probabilistic detection strategy against the packet tampering attack caused by HTs. The proposed Probability Based Detection Strategy (PBDS) has a hardware cost as low as E2E methods, but its detection capability is in par with that of H2H methods. It is also suitable to work with both XY and adaptive routing algorithms.

In the proposed PBDS strategy, for each node in the NoC, the defender attempts to determine the probability that it is infected with an HT. At runtime, these probabilities are iteratively updated to determine whether a packet is being tampered or not. A node is regarded as infected with HT if its probability of being infected exceeds a threshold. Once a node is believed to be HT-infected, the countermeasure starts to work. The experiments show that the proposed method is effective to detect packet-tampering HTs which are conditionally triggered, and thus helps to reduce the infection rate to below 5%.

Despite its strong ability to detect HTs, PBDS comes with a relatively high bandwidth and computation overheads. This issue can be further addressed in this paper by adopting a packet probing technique and simplified probability calculation that together helps to drastically reduce the overheads at nearly no performance penalty.

In summary, our main contributions are the following:

- We propose an online detection strategy against the packet tampering attack caused by conditionally triggered HTs.
- Three approximate methods are also proposed to reduce the runtime bandwidth and computation costs of the original method.
- We have evaluated the proposed methods through extensive experiments, and the results confirm that all the proposed methods can effectively detect HTs with low overheads.

The rest of this paper is organized as follows. Section 2 surveys the related work. Section 3 introduces the threat model and provides the details of the proposed HT detection strategies, and Section 4 presents a few approximate methods to implement these strategies. Section 5 reports the experimental results when the proposed detection methods are applied in various scenarios. Finally, Section 6 concludes the paper.

2 Related Work

In this section, various attacks initiated and enabled by HTs against NoCs are surveyed. For the scope considered in this paper, we will particularly focus on the detection methods and countermeasures against packet tampering attacks.

2.1 HT Attacks

The HTs can launch various attacks in NoC to cause damages to many-core chips. At its worst, it can bring them to a total malfunction [3] [10] [4] [11] [12] [13] [14]. These attacks can be categorized as: 1) flooding attacks [15], where a large number of useless data packets flood and saturate a victim node; 2) packet drop attack [16], where some packets get dropped or directed to malicious nodes so that the victim node will lose some packets destined to it; 3) privilege escalation attack [17], in which an ordinary user process is illegally promoted to become a supervisor so that it can steal protected information like passwords [18]; 4) routing loop attack [2], where packets passing through the malicious nodes are routed back to the source node, effectively blocking the source node from communicating with any other nodes; 5) packet misrouting attack [27], where the HTs can apply attacks by misrouting the packets to cause deadlocks and virtually link failures, resulting in the network performance degradation; 6) eavesdropping attack [35], where the attacker secretly listens to the on-chip communication in order to steal sensitive information; 7) side channel attack [36], where the attacker exploits non-functional behaviors, such as time, power, electromagnetic radiation, heat, and acoustic waveforms, to attack a supposedly secure system; and 8) packet tampering attack, where the HTs can deliberately tamper data packets travelling in the NoC and thus compromise the integrity of the data communications. A packet tampering attack can be particularly dangerous [19], since flipping a few bits in a data packet can either compromise data integrity or divert the packet to a node other than its destination. An HT that can launch and sustain a packet tampering attack could be made very simple with the help of a fault injection vector [15].

What makes HTs even harder to be detected is that these small circuits are often designed to benefit from conditional triggering. That is, the HTs are trigged and become active only when the specific triggering conditions are met. For instance, the attacker can use synchronous and asynchronous counters to activate a timing-triggered HT. A combination of memory data access patterns can also be used as HT triggers [20]. In [11], either temperature or voltage fluctuation was employed as the HT triggers. Similarly, an HT in [21] could be activated by a specific complex bit pattern embedded into the input messages, after which all the following packets can be diverted away from their destined addresses.

2.2 Detection and Countermeasures against HT Attacks

The methods for detecting HT attacks broadly fall into two categories: 1) offline detection approaches that are able to detect the malicious components when the chip is running payload-free, and 2) online detection approaches that are applied to detect the HTs on the fly.

Categorized as offline detection, various circuit-level HT detection methods have been attempted. In [9] [22], these techniques were attempted to detect HTs through logic testing and side channel analysis which are part of the post-silicon test/verification processes. Logic testing methods need to apply a test pattern to activate the HTs if they ever exist, and correspondingly analyze the chip's power consumption for

the sake of HT detection. However, there is no guarantee that these methods are able to generate all the needed patterns to trigger all the possible HTs. The detection methods based on the side channel analysis, on the other hand, rely on measuring a few key system parameters (e.g., supply current or path delay). Any significant departure from the nominal values hints a possibility of unexpected design modifications and hence the existence of an HT. Since device parameters may vary greatly from chip to chip, and more so from wafer to wafer, effectiveness of HT detection through side channel analysis is fairly questionable.

There are also some offline HT detection methods that are based on the Bayesian theorem. In [33], the authors applied a Bayesian inference-based calibration technique to detect the HTs that could be inserted during the manufacturing phase of the integrated circuits. A system level approach was adopted in [34], where HT detection is done through side-channel power analysis with machine learning. Essentially, naïve Bayesian is applied to separate the power traces of the hardware Trojan free systems from those injected by the hardware Trojans. In [38], transient power supply currents can be obtained and analyzed to detect hardware Trojans, and these extracted current measurements can be classified by the Naïve Bayesian classifiers. Since these methods only work in the testing stage, they are less flexible and are skeptical to find out some hidden HTs.

There are also some recent offline HT detection methods that take advantage of machine learning techniques. For instance, the work presented in [42] falls into this category, where the model is trained by learning different bit flip patterns between Trojan-infected design and secure design.

Apart from offline detection, there are many ways to detect HTs at runtime with the circuit and architectural support, and many of the methods rely on the use of machine learning techniques for analysis and classification. In [30], a detection circuit looks into the threshold voltage degradation to detect a very specific data-snooping attack at runtime. In [37], the authors proposed a lightweight and real-time DoS attack detection mechanism which relies on the latency data gathered by the NoC components to detect and localize the malicious IP. In [28], a learning-based design framework which uses an artificial neural network was proposed to detect HTs. On a separate study [39], the power profile of the microcontrollers instruction sets is extracted and then applied to train a machine learning algorithm using Naïve Bayesian; the HTs are detected by classifying the power traces at runtime. This scheme, however, cannot be applied to detect the HTs that are turned ON and OFF dynamically. With the emerging wireless networks-on-chip (WiNoCs), coding method and machine learning classifier can be applied to detect and fight against eavesdropping and jamming attacks targeting wireless medium [31]. In [43], a systematic security architecture was proposed to detect and defense against a variety of attacks on NoC communications. Since its packet integrity check is performed at every hop of the NoC, the efficiency of packet transmission is adversely impacted and there tends to be a high power consumption as a result.

Online HT detection can also be approached by performing authentication [19], especially for the detection of the HTs capable of packet tampering, and a number of methods have been tried along this line. In [8], an E2E detection that uses cyclic redundancy check (CRC) and algebraic manipulation codes (AMD) was able to protect data packets against fault injection attacks. The work in [44] is also based on the authentication performed at the destination; as so, the method can figure out nothing but detect whether an attack is launched or not. One obvious drawback of any E2E method is that it cannot locate the malicious nodes, putting it into a big disadvantage over an H2H approach described in [19]. Since the authentication in H2H is performed at each hop along the packet's routing path, this approach comes with high costs of power consumption and area. Actually, an H2H method consumes as much as 3× more power than its E2E counterpart [9]. Such a big power consumption gap between E2E and H2H was tackled in [6] by combining the two approaches. Basically, an E2E authentication method is applied first to detect an attack, while another H2H authentication will follow to locate the malicious nodes. However, this hybrid approach, referred as energy efficient hierarchical approach (EETD) [6], cannot detect the HTs that quickly return to hibernation after waking up for only a short period of time to launch an attack.

Table 1 summarizes the features and merits of the exemplary online and offline detection methods [6] [8] [9] [19] [22].

3 The Proposed Detection Strategy

The proposed detection strategy is an online detection method based on Bayesian probability analysis. In this section, the threat model and details of the proposed HT detection strategy are introduced.

3.1 Threat Model

The target system consists of in-house designed IP cores that can be trusted, including the processors, memories, network interfaces and suspicious NoC IP cores designed by third parties. The trusted processors are supposed to communicate with each other through the untrusted network-on-chip that has a 2D mesh topology and supports either an adaptive or deterministic XY routing method. Note that the proposed detection strategy can be readily extended to NoCs built upon other topologies and routing algorithms.

Implanted in the NoC routers, the HTs can be turned on or off at runtime, and they can be triggered (turned on) by internal signals or specific events [11] [20] [24] [40] [41]. We assume that a malicious program is running on the processor. This malicious program can send some special packets to trigger the HTs implanted in NoC via memory access. Here we assume that when an HT is turned on, the payload of a packet passing though it can be tampered. The same HT can be turned off to escape from being detected. The E2E authentication methods [15] [23] in this case can be applied to check the packet integrity.

3.2 Overview

The proposed countermeasure has four components, which are shown in Fig. 1, and the backbone NoC is untrusted (in black) as described in Section 3.1.

TABLE 1: Different Detection and Countermeasures against the HT Attacks

	Online detection	Ability to precisely locate HTs	Ability to detect randomly switching HTs	Power/area overhead
Offline methods [9] [22]	no	[9]: no; [22]: yes	no	- -
E2E [8]	yes	no	no	low
H2H [19]	yes	yes	yes	high*
EETD [6]	yes	yes	no	low
The proposed method	yes	yes	yes	low

(*Note: each router should be equipped with a hop-to-hop detection unit)

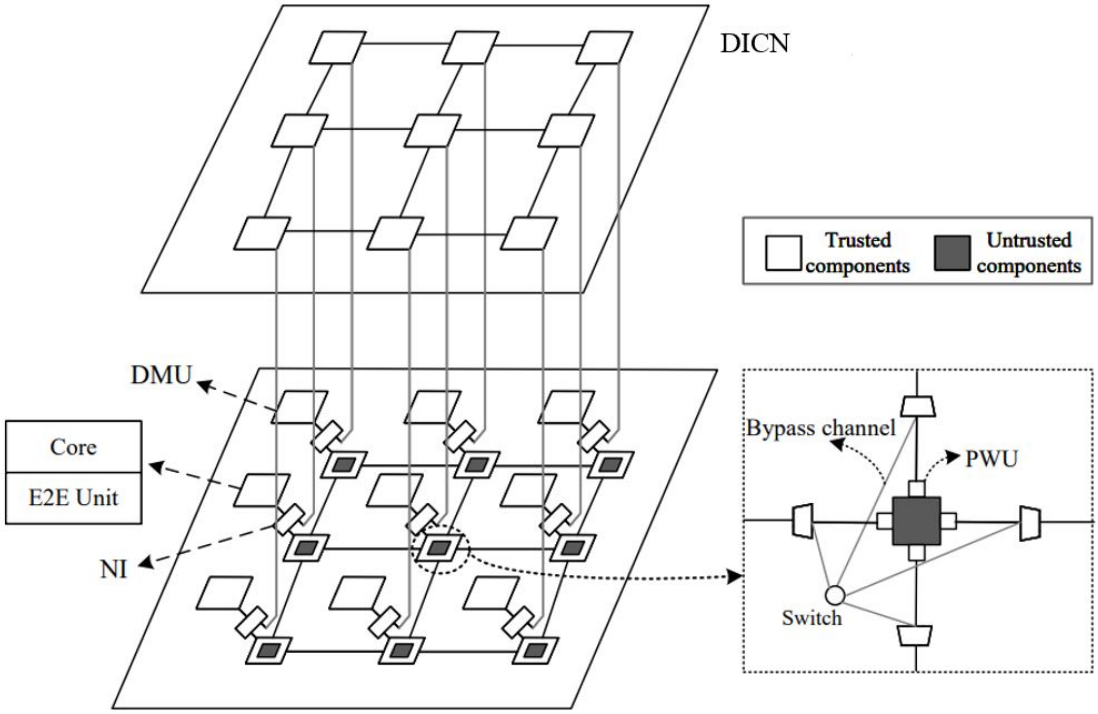


Fig. 1: Architectural overview of the proposed detection strategy.

- The defense management unit (DMU): This is a detection program running at the designated core (e.g., the core located at the top left corner of the NoC with coordinates of (0,0) in Fig. 1) to assess the probability that a node has an implanted HT. In the case that the core serving as the global manager is compromised, some other cores can be elevated to become the defense management units. In this manner, a total of β cores are designated as the backup global managers and they are turned on periodically. Once the current global manager is set to be offline (disconnected), one of the β cores will be selected as the next global manager.
- The E2E authentication units: Every network interface (NI) has an attached E2E authentication unit, and this unit can adopt a robust error detection scheme, such as algebraic manipulation detection or cyclic redundancy check codes, to check the integrity of a received packet [8].
- The detection information communication network (DICN): The core (all the cores are trusted) sends the packets carrying the detection information including packets' routing paths and authentication results to the DMU via a trusted communication network named DICN, and the DMU notifies the detection results to the cores which connect via DICN [43]. As shown in Fig. 1, the DICN is an independent network fabricated in a trusted foundry, which is lightweight for transferring the detection information. It has narrower flit width and fewer buffers compared to the untrusted NoC. Trusted cores and the DMU are connected to both the DICN and the untrusted NoC via the cores' NIs (network interfaces), which include packet makers/deassemblers and buffers of two different bit-widths [43] corresponding to the untrusted NoC and the DICN.
- Trusted secure components in untrusted NoC: As Fig. 1 shows, the secure components include PWUs (Packet Writing Units) and the bypass channels for HT detection and countermeasure, which connects to the ports of the routers in the untrusted NoC.

The PWUs are attached to the input ports of NoC

routers, and each physical port connects to one PWU. The PWUs have to be designed, fabricated, and assembled by the trusted parties. The PWUs are powered off when not being used, and when they are turned on, they insert the node's ID into the packets that pass through the node. The PWU design is illustrated in Fig. 2.

As Fig. 2 shows, the Probe Flag and Node ID are preset during reset in PWU for each node. The probe flag of a packet is compared against Probe Flag, and if both are identical, the signal en is asserted. The IDs of all the nodes that the probe packet has traversed are added into the payload field, and the index field, I_n points to the payload of the current node. As a node's ID is added to Payload[I_n], I_n is incremented by 1, meaning that, in the next downstream router, its node ID is ready to go to I_{n+1}.

The bypass channels shown in Fig. 1 are connected to the untrusted NoC using a 4-port switch and multiplexers with the same flit width as the untrusted NoC to protect the packet from being attacked by the malicious node [47]. Once the trusted core has received the detection results and its untrusted NoC router is high-risky, the core configures the MUX to select the bypass channels where the selection signal is connected to the core's bus [47].

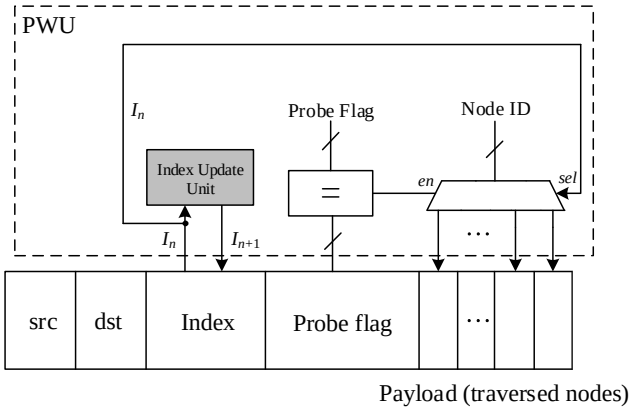


Fig. 2: The design of the proposed PWU.

In the proposed detection strategy, the defense management unit (DMU) computes the probabilities of the nodes having implanted HTs. These probabilities are updated iteratively based on the information provided by the E2E authentication units and PWUs. The E2E authentication units check whether a packet is tampered or not, and the PWUs record a packet's routing path. The nodes that have the probabilities of being HT-implanted higher than a preset threshold are marked as high-risk nodes. Correspondingly, all the subsequent packets can skip the high-risk nodes by the bypass channels to avoid attacking.

As Fig. 3 shows, packet p is routed from its source to destination with its routing path recorded by the PWUs of each node on the routing path. At the destination, packet p's integrity is checked by an E2E authentication unit. Afterwards, packet p's routing path and its authentication result are sent to the defense management unit that updates the probability of being implanted an HT for all the nodes that are traversed by packet p. Note that the routing paths and authentication results are sent from the trusted core to the DMU via DICN [6] [43]. The DMU sends the detection results (e.g., the set of high-risk NoC nodes) to the cores via the DICN [6] [43]. Then the cores configure the multiplexers to select the bypass channels so the packets bypass the high-risk NoC nodes.

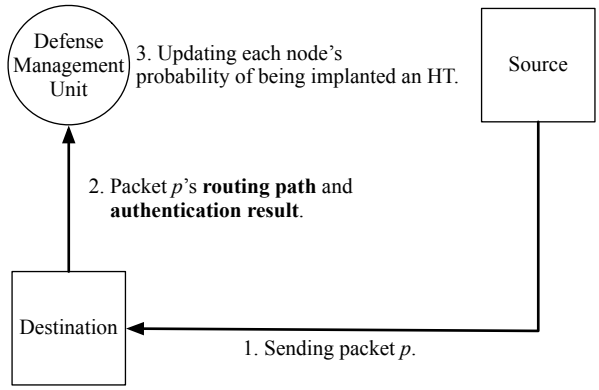


Fig. 3: The defense management unit uses the packets' routing path information and their authentication results to update a node's probability of having an implanted HT.

The next subsection provides the details of the proposed detection algorithm.

3.3 Probability Based Detection Algorithm

All the mathematical notations used in this paper are listed in Table 2.

TABLE 2: Notations

m	Number of HTs in the chip.
n	Number of cores in the chip.
$\mathbb{P}(\cdot)$	The event that node (i,j) is implanted with an HT.
α	The estimated activation rate of each HT from the detector's perspective.
Ω_p	The set of nodes on the path that packet p passes.
L_p	Packet p's authentication result, equals to 1 if p is tampered, 0 otherwise.
$I^p_{i,j}$	An event that packet p is attacked by node (i,j)'s HT.
$\bar{I}^p_{r,s}$	An event that packet p is not attacked by node (i,j)'s HT.
R	Risk threshold. The nodes whose probabilities of being malicious (PoMs) are higher than the risk threshold are marked as high-risk nodes.
K^t_p	An intermediate variable used in the equation for updating the PoMs.

Definition 1. Probability of a node being malicious (PoM) is the probability measure of a node being implanted an HT.

The DMU (i.e., the defense management unit) maintains a PoM for each node and updates it regularly at runtime. In each iteration, the DMU updates the PoMs based on the information provided by PWUs and E2E authentication units. In the t-th iteration, node (i,j)'s PoM is denoted as $\mathbb{P}^t(H_{i,j})$, where event $H_{i,j}$ denotes that node (i,j) is implanted with an HT.

Definition 2. Evidence. Each packet has an associated evidence. Used by the DMU to update a node's PoMs, the evidence of packet p is registered as a pair made of a packet's routing path and its authentication result, given as (Ω_p, L_p), where Ω_p contains packet p's routing path and L_p is the packet's authentication result.

1) The routing path of packet p, denoted as $\Omega_p = \{(x_1,y_1),...,(x_v,y_v)\}$, includes all the nodes that packet p passes when it is routed from its source located at (x_1,y_1) to the destination (x_v,y_v).
2) The authentication result of packet p, L_p is binary, assuming a value of 1 when packet p is attacked while it is routed through its routing path, or 0, otherwise.

We denote α as the estimated activation rate of each HT from the detector's perspective. Its value can be set experimentally.

The proposed probability-based algorithm is to update the PoMs iteratively by examining the packets' routing paths and authentication results. The iteration process can be expressed as follows,

$$\mathbb{P}^0(H_{i,j}) = \frac{m}{n} \tag{1}$$

$$\mathbb{P}^{t+1}(H_{i,j}) = F(\mathbb{P}^t(H_{i,j}), \Omega_p, L_p) \tag{2}$$

The PoMs are initialized to be $\frac{m}{n}$ by assuming that HTs are distributed evenly across the chip, with m being the number of HTs and n being the number of nodes. As the defender does not have a priori knowledge of the value of m, this ratio $\frac{m}{n}$ can only be set experimentally.

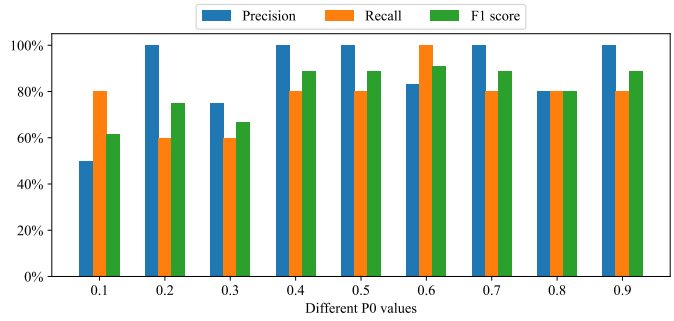


Fig. 4: The experimental results with different P^0.

To evaluate the sensitivity of the detection performance to P^0, we conducted experiments varying P^0 from 0.1 to 0.9 in steps of 0.1. One can see from Fig. 4 that (1) when $P^0 \leq 0.3$, detection performance decreases, likely due to the model being too conservative in updating risk; (2) when P^0>0.3, detection performance improves and stabilizes, indicating the model becomes more responsive to observed evidence. Based on these findings, we choose P^0=0.6 in all experiments to achieve a good balance between responsiveness and stability. Importantly, we found that P^0 has only a minor effect on system performance, provided it is set above a reasonable threshold. In Eqn. 2, $F(\cdot)$ is the function that computes $\mathbb{P}^{t+1}(H_{i,j})$, the updated PoMs in the next iteration.

In the t-th iteration, when the defense management unit obtains the packet p's routing path Ω_p and authentication result L_p, it computes the updated PoMs for each node (i,j) on p's routing path ($(i,j) \in \Omega_p$). That is,

$$\mathbb{P}^{t+1}(H_{i,j}) = L_p \times \mathbb{P}^t(H_{i,j}|L_p=1) + (1-L_p) \times \mathbb{P}^t(H_{i,j}|L_p=0) \tag{3}$$

According to Eqn. 3, if packet p is attacked, the authentication creates a result of $L_P = 1$. Correspondingly, $\mathbb{P}^{t+1}(H_{i,j}) = \mathbb{P}^t(H_{i,j}|L_p=1)$. This means $\mathbb{P}^{t+1}(H_{i,j})$ can be represented as the node (i,j) 's PoM , given that packet p is found attacked in the t-th iteration. On the other hand, if packet p is not attacked, $L_P = 0$, and correspondingly, $\mathbb{P}^{t+1}(H_{i,j}) = \mathbb{P}^t(H_{i,j}|L_p=0)$, which means $\mathbb{P}^{t+1}(H_{i,j})$ can be represented as the PoM of node (i,j), given that packet p is not attacked in the t-th iteration.

Even if packet p is not attacked, it is possible that HTs may still exist (undetected yet) in any node on the path and they are just turned OFF (i.e., hibernated) throughout the detection phase. When the undetected HT is turned on, node (i,j)'s PoM will show a steady increase; as a result, the HTs will be detected after multiple detection iterations.

Theorem 1.

1) With the authentication result of $L_p = 1$,

$$\mathbb{P}^t(H_{i,j}|L_p=1) = \frac{\alpha + (1-\alpha)(1 - \frac{\prod_{(r,s)\in\Omega_p}(1-\alpha\cdot\mathbb{P}^t(H_{r,s}))}{1-\alpha\cdot\mathbb{P}^t(H_{i,j})})}{1-\prod_{(r,s)\in\Omega_p}(1-\alpha\cdot\mathbb{P}^t(H_{r,s}))} \times \mathbb{P}^t(H_{i,j}) \tag{4}$$

2) With the authentication result of $L_p = 0$,

$$\mathbb{P}^t(H_{i,j}|L_p=0) = \frac{1-\alpha}{1-\alpha\cdot\mathbb{P}^t(H_{i,j})} \times \mathbb{P}^t(H_{i,j}) \tag{5}$$

Proof: According to Bayesian theorem,

$$\mathbb{P}^t(H_{i,j}|L_p=1) = \frac{\mathbb{P}^t(L_p=1|H_{i,j})}{\mathbb{P}^t(L_p=1)} \times \mathbb{P}^t(H_{i,j}) \tag{6}$$

$$\mathbb{P}^t(H_{i,j}|L_p=0) = \frac{\mathbb{P}^t(L_p=0|H_{i,j})}{\mathbb{P}^t(L_p=0)} \times \mathbb{P}^t(H_{i,j}) \tag{7}$$

where $\mathbb{P}^t(H_{i,j})$ is node (i,j)'s PoM in the t-th iteration. The two terms $\mathbb{P}^t(L_p=1)$ and $\mathbb{P}^t(L_p=1|H_{i,j})$ in Eqn. 6 are computed as follows.

(1) $\mathbb{P}^t(L_p=1)$ is the probability that packet p gets tampered in the routing path, and it is determined as,

$$\mathbb{P}^t(L_p = 1) = 1 - \mathbb{P}^t(L_p = 0) \tag{8}$$

The event $L_p = 0$ marks that packet p is not tampered along the routing path, and $\mathbb{P}^t(L_p = 0)$ is the probability that none of the nodes on Ω_p tampers packet p. Actually, the event $L_p = 0$ can be viewed as a series of events that no node on the path ever attacked packet p. That is, $\mathbb{P}^t(L_p = 0) = \mathbb{P}^t(\bigcap_{(r,s)\in\Omega_p} \bar{I}^p_{r,s})$, where $\bar{I}^p_{r,s}$ is the event that packet p is not attacked by node (r, s)'s HT. Assuming all the nodes are independent from each other, we arrive at

$$\mathbb{P}^t(L_p = 0) = \mathbb{P}^t(\bigcap_{(r,s)\in\Omega_p} \bar{I}^p_{r,s}) = \prod_{(r,s)\in\Omega_p} \mathbb{P}^t(\bar{I}^p_{r,s}) \tag{9}$$

where $\mathbb{P}^t(\bar{I}^p_{r,s})$ is the probability that node (r, s) does not tamper packet p.
Since $I^p_{r,s}$ and $\bar{I}^p_{r,s}$ are mutually exclusive, we have

$$\mathbb{P}^t(\bar{I}^p_{r,s}) = 1 - \mathbb{P}^t(I^p_{r,s}) \tag{10}$$

where $\mathbb{P}^t(I^p_{r,s})$ is the probability that node (r, s) tampers packet p. $\mathbb{P}^t(I^p_{r,s}) = 1$ if the following two conditions are satisfied simultaneously: (i) an HT is implanted in node (r, s), (ii) the implanted HT is being turned on when packet p traverses node (r, s).
Assume the probability that an HT is turned on while a packet traverses a malicious node is α. Then we have

$$\mathbb{P}^t(I^p_{r,s}) = \alpha \times \mathbb{P}^t(H_{r,s}) \tag{11}$$

Combining Eqns. 8 through 11, we have

$$\mathbb{P}^t(L_p = 1) = 1 - \prod_{(r,s)\in\Omega_p} (1 - \alpha \times \mathbb{P}^t(H_{r,s})) \tag{12}$$

(2) $\mathbb{P}^t(L_p = 1|H_{i,j})$ is the probability that packet p is tampered if node (i, j) is a malicious node, can be written as,

$$\mathbb{P}^t(L_p = 1|H_{i,j}) = \mathbb{P}^t(I^p_{i,j}|H_{i,j}) + \mathbb{P}^t(\bar{I}^p_{i,j}|H_{i,j})\mathbb{P}^t(\bigcup_{(r,s)\in\Omega_p,(r,s)\neq(i,j)} I^p_{r,s}) \tag{13}$$

where $\mathbb{P}^t(I^p_{i,j}|H_{i,j})$ is the probability that node (i, j) tampers packet p while node (i, j) is implanted with an HT, and $\mathbb{P}^t(\bar{I}^p_{i,j}|H_{i,j})\mathbb{P}^t(\bigcup_{(r,s)\in\Omega_p,(r,s)\neq(i,j)} I^p_{r,s})$ is the probability that the attack is caused by nodes $((r, s) \in \Omega_p, (r, s) \neq (i, j))$ other than node (i, j). The term $\mathbb{P}^t(I^p_{i,j}|H_{i,j})$ is the probability that node (i, j)'s HT is turned on when packet p traverses the node. We thus have,

$$\mathbb{P}^t(I^p_{i,j}|H_{i,j}) = \alpha \tag{14}$$

The term $\mathbb{P}^t(\bar{I}^p_{i,j}|H_{i,j})\mathbb{P}^t(\bigcup_{(r,s)\in\Omega_p,(r,s)\neq(i,j)} I^p_{r,s})$ is the probability that node (i, j)'s HT is turned off when packet p traverses it, but packet p is actually tempered by some other nodes on the routing path than node (i, j). In this case,

$$\mathbb{P}^t(\bar{I}^p_{i,j}|H_{i,j}) = 1 - \alpha \tag{15}$$

$$\mathbb{P}^t(\bigcup_{(r,s)\in\Omega_p,(r,s)\neq(i,j)} I^p_{r,s}) = 1 - \mathbb{P}^t(\bigcap_{(r,s)\in\Omega_p,(r,s)\neq(i,j)} \bar{I}^p_{r,s}) \tag{16}$$

Based on Eqns. 15 and 16, we have,

$$\begin{aligned}&\mathbb{P}^t(\bar{I}^p_{i,j}|H_{i,j})\mathbb{P}^t(\bigcup_{(r,s)\in\Omega_p,(r,s)\neq(i,j)} I^p_{r,s}) \\ &= (1-\alpha)(1 - \mathbb{P}^t(\bigcap_{(r,s)\in\Omega_p,(r,s)\neq(i,j)} \bar{I}^p_{r,s})) \\ &= (1-\alpha)(1 - \prod_{(r,s)\in\Omega_p,(r,s)\neq(i,j)} \mathbb{P}^t(\bar{I}^p_{r,s}))\end{aligned} \tag{17}$$

Combining Eqns. 13, 14, and 17, we have

$$\mathbb{P}^t(L_p = 1|H_{i,j}) = \alpha + (1-\alpha)(1 - \frac{\prod_{(r,s)\in\Omega_p}(1 - \alpha \cdot \mathbb{P}^t(H_{r,s}))}{1 - \alpha \cdot \mathbb{P}^t(H_{i,j})}) \tag{18}$$

Combining Eqns. 12 and 18, we can derive Eqn. 4, which can be used to update the case of $L_p = 1$.

Similarly, we can prove the second part of the theorem. Putting all things together, we can expand Eqn. 7 to become,

$$\begin{aligned}&\mathbb{P}^t(H_{i,j}|L_p = 0) \\ &= \frac{\mathbb{P}^t(L_p = 0|H_{i,j})}{\mathbb{P}^t(L_p = 0)} \times \mathbb{P}^t(H_{i,j}) \\ &= \frac{\mathbb{P}^t(\bar{I}^p_{i,j}|H_{i,j})\mathbb{P}^t(\bigcap_{(r,s)\in\Omega_p,(r,s)\neq(i,j)} \bar{I}^p_{r,s})}{\mathbb{P}^t(L_p = 0)} \times \mathbb{P}^t(H_{i,j}) \\ &= \frac{(1-\alpha)\prod_{(r,s)\in\Omega_p,(r,s)\neq(i,j)}(1 - \alpha \cdot \mathbb{P}^t(H_{r,s}))}{\prod_{(r,s)\in\Omega_p}(1 - \alpha \cdot \mathbb{P}^t(H_{r,s}))} \times \mathbb{P}^t(H_{i,j}) \\ &= \frac{1-\alpha}{1 - \alpha \cdot \mathbb{P}^t(H_{i,j})} \times \mathbb{P}^t(H_{i,j})\end{aligned} \tag{19}$$

(End of proof)

By substituting Eqns. 4 and 5 into Eqn. 3, we have,

$$\begin{aligned}\mathbb{P}^{t+1}(H_{i,j}) &= L_p \frac{\mathbb{P}^t(L_p = 1|H_{i,j})}{\mathbb{P}^t(L_p = 1)} \times \mathbb{P}^t(H_{i,j}) \\ &+ (1 - L_p)\frac{\mathbb{P}^t(L_p = 0|H_{i,j})}{\mathbb{P}^t(L_p = 0)} \times \mathbb{P}^t(H_{i,j}) \\ &= L_p \frac{\alpha + (1-\alpha)(1 - \frac{\prod_{(r,s)\in\Omega_p}(1-\alpha\cdot\mathbb{P}^t(H_{r,s}))}{1-\alpha\cdot\mathbb{P}^t(H_{i,j})})}{1 - \prod_{(r,s)\in\Omega_p}(1 - \alpha \cdot \mathbb{P}^t(H_{r,s}))} \times \mathbb{P}^t(H_{i,j}) \\ &+ (1 - L_p)\frac{1-\alpha}{1 - \alpha \cdot \mathbb{P}^t(H_{i,j})} \times \mathbb{P}^t(H_{i,j})\end{aligned} \tag{20}$$

By denoting $K^t_p = \prod_{(r,s)\in\Omega_p}(1 - \alpha \cdot \mathbb{P}^t(H_{r,s}))$, Eqn. 20 can be rewritten as

$$\begin{cases} K^t_p = \prod_{(r,s)\in\Omega_p}(1 - \alpha \cdot \mathbb{P}^t(H_{r,s})), \\ \mathbb{P}^{t+1}(H_{i,j}) = \mathbb{P}^t(H_{i,j})\frac{(1-\alpha)(1-K^t_p)+\alpha L_p(1-\mathbb{P}^t(H_{i,j}))}{(1-\alpha\mathbb{P}^t(H_{i,j}))(1-K^t_p)} \end{cases} \tag{21}$$

The proposed detection strategy rests on calculting Eqn. 21 iteratively. From the detector's perspective, the exact activation rate of each HT is not known a priori. Instead α can be taken as the estimated activation rate of each HT, and consequently, its value can be determined experimentally, as detailed in Section 5.1.

When a packet is found tampered, the PoM of all the nodes on the routing path will be increased according to Eqn. 21. On the other hand, if a packet is not tampered, the PoM of the nodes on the path will decrease. After several iterations, the malicious nodes and the normal nodes can be separated out in terms of their respective PoM values.

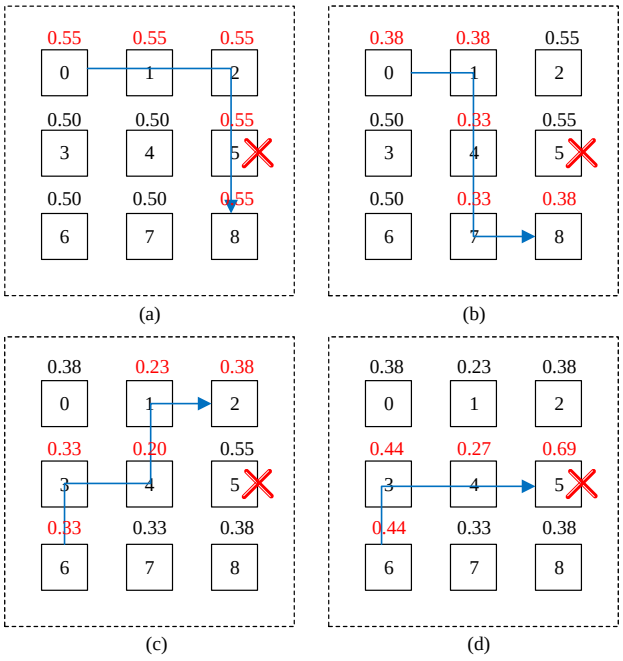


Fig. 5: An example that illustrates the PoM updating process.

Fig. 5 illustrates the PoM updating process. Assume the initial PoM of each node is 0.5, and the value of α that needed for PoM updating is set to 0.5. Node 5 is set to be HT-infected, and an adaptive routing algorithm DyAD [46] is used in this case. In Fig. 5(a), there is a probe packet going from Node 0 to Node 8. If this packet is attacked at Node 5, then at the destination node, the authentication result L_p is 1. The DMU calculates the PoM of each node in the routing path according to Eqn. 21, and the PoM of each node in the path changes to 0.55. Similarly, in Fig. 5(b)-(d), the PoMs of each node are updated in the next iterations. Assume the HT in Node 5 is turned on when the packets traverse it, and every packet passing through it gets attacked. In Fig. 5(d), the PoM of Node 5 increases to 0.69, which is apparently higher than PoM's of other normal nodes. After a few more iterations, the gap between Node 5 and other normal nodes in terms of PoM values will be even wider, and eventually, the PoM of Node 5 will exceed the preset threshold, in this case 0.95, and correspondingly, this node will be confirmed HT-infected.

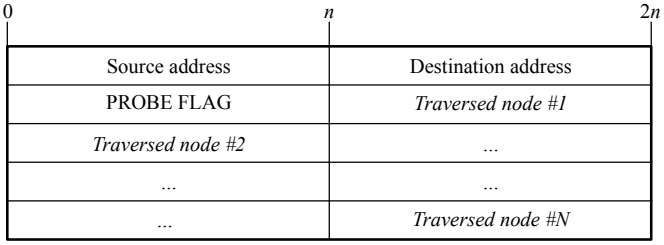


Fig. 6: The format of probe packet.

After each updating iteration, the DMU will check the nodes whose PoMs are higher than a preset threshold and mark them as high-risk nodes. The DMU sends the set of high-risk NoC nodes to the cores via the DICN.

3.4 Evidence Collection

To update the PoMs according to Eqn. 21, the DMU needs to obtain (i) each packet's routing path (Ω_p), including all the nodes that a packet traverses, and (ii) each packet's authentication result (L_p). They are obtained as follows.

1) The routing path can be obtained by analyzing a packet's source and destination when the NoC uses deterministic XY routing. To obtain the routing path when an adaptive routing is used, a straightforward but naïve approach is to add all the traversed nodes' IDs into the payload of the packets.
2) The authentication result is obtained by the E2E authentication unit [15] [23] at a packet's destination.

Storing the IDs of all the traversed nodes in every data packet can increase the bandwidth usage. Instead, a sampling-based approach can be applied. In this case, probe packets can be used to infer the routing paths of normal packets. As a result, there is no need to authenticate the probe packets at the destination node. The DMU demands the same source nodes to send probe packets every a few cycles to the same destinations as any normal packets. The traversed nodes collected by the destination of a normal packet make Ω_p and their PoMs updated according to Eqn. 21. As shown in Fig. 6, a probe packet has a format that includes the source address, the destination address, a PROBE FLAG and the payload which stores traversed nodes' IDs. The PROBE FLAG is used to activate the PWUs sitting in the routers that the packet has passed. The PWUs add their host nodes' IDs to any packet that passes them through, once they see that the PROBE FLAG is set.

Algorithm 1 shows the main workflow of the proposed HT detection strategy, named as PBDS. In order to reduce the bandwidth or power overheads incurred in implementing this algorithm, approximate Bayesian computation methods can be applied, as described in the next section.

3.5 Hierarchical Attack Detection

For a large scale NoC, collecting the detection information (i.e., each probe packet's routing path (Ω_p) and each probe

Algorithm 1: Probability Based Detection Strategy (PBDS)

Input: α: The estimated probability that an HT being turned on while a packet passes through an HT-infected node.
R: The threshold for classifying high-risk nodes.
T: The maximum number of iterations.
$F[i,j]$: The nodes' PoMs.
Output: HRN: The set of high-risk nodes.
Function: Locating the high-risk nodes.

$t = 0$;
while $t < T$ do
  A packet p is routed from its source to destination;
  The DMU fetches packet p's routing path and authentication result (Ω_p, L_p);
  /* Computing the intermediate variable. */
  $K_p^t = \prod_{(r,s)\in\Omega_p}(1 - \alpha \cdot F[r,s])$;
  for (i,j) in Ω_p do
    /* Updating traversed nodes' PoMs. */
    $F[i,j] = F[i,j]\frac{(1-\alpha)(1-K_p^t)+\alpha L_p(1-F[i,j])}{(1-\alpha F[i,j])(1-K_p^t)}$;
  end
  $t = t + 1$;
end
$HRN = \{\}$;
for each (i,j) in NoC do
  if $F[i,j] > R$ then
    Add (i,j) to HRN;
  end
end
return HRN;

packet's authentication result (L_p)) can take a long time to complete due to the long distance between a destination node and defense management unit (DMU). As the DMU needs to manage a high number of NoC nodes in a large system, there is a high cost to compute the PoMs. Correspondingly, the detection time will be greatly increased, and there is an adverse implication on the detection rate.

To address aforementioned challenges in detecting large-sized NoC systems, the information propagation and PoM computation can be done in a hierarchical manner. The network can be virtually grouped into several clusters, and each cluster has its own local defense manager. Each local defense manager collects information from the nodes within its cluster and aggregates the information and performs PoM update, after which the aggregated information is sent to the global defense manager for further information processing. An architecture and information dataflow that can support such hierarchical attack detection are shown in Fig. 7, where intra-cluster (inter-cluster) information flow is drawn in the blue (red) lines. This scheme bears a great deal of similarity to the one adopted in [45].

4 Approximate Methods

As an extension of the PBDS framework, we herein present three approximate methods that can be applied to reduce the runtime bandwidth requirements and/or computation costs:

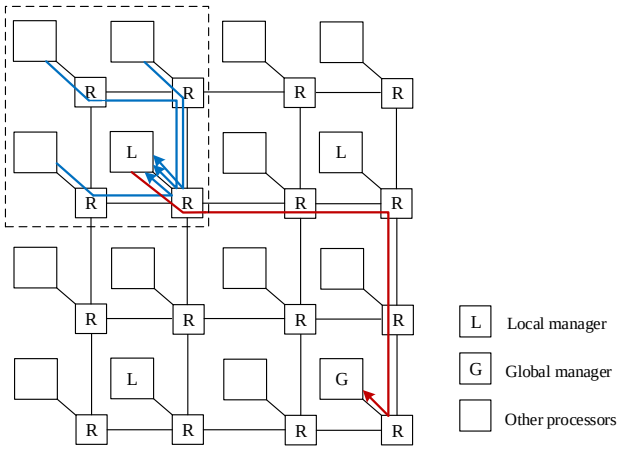


Fig. 7: The dataflow of information aggregation in the hierachical architecture.

1) PBDS-A1: It improves practicality by using probe packets to approximate the routing paths of normal packets. These probe packets, sent between the same source-destination pairs as the normal packets, help approximate real routing path under adaptive routing algorithms. The original PBDS equations are updated accordingly to accommodate this approximation. It can be applied to reduce the number of probe packets and thus to reduce the bandwidth overhead.
2) PBDS-A2: It is a simplified variant of PBDS that assumes at most one malicious node (HT) exists along any packet's route. This is a reasonable assumption in scenarios where the number of HTs is small. The simplification reduces computational complexity without degrading detection performance.
3) PBDS-A3: It is a combination of PBDS-A1 and PBDS-A2. It uses probe packets route approximation and assumes a single malicious node per route, thus reducing both bandwidth overhead and computation overhead.

4.1 PBDS-A1

In PBDS-A1, the routing paths of the probe packets are used to estimate those of the normal packets. The evidence defined in Defintion 2 is extended to become a three-tuple, denoted as $(\tilde{\Omega}_p, \gamma^p, L_p)$, which includes (i) the approximate routing path $(\tilde{\Omega}_p)$, (ii) the passing probabilities $\gamma^p = \{\gamma^p_{i,j} | (i,j) \in \tilde{\Omega}_p\}$ for each node (i,j) in the set $(\tilde{\Omega}_p)$, and (iii) the authentication result (L_p).

Definition 3. The approximate routing path $\tilde{\Omega}_p$ of packet p is made of all the nodes that packet p has routed through from the source to the destination.

Definition 4. The passing probability $\gamma^p = \{\gamma^p_{i,j} | (i,j) \in \tilde{\Omega}_p\}$ of packet p is the set of probabilities that packet p passes the nodes on the approximate routing path $\tilde{\Omega}_p$.

To obtain packet p's approximate routing path and corresponding passing probabilities, the probe packets are transmitted from packet p's source node to its destination node every a few cycles, and the sending frequency of the probe packets is a little slower than that of PBDS. Specifically, for packet p, N (which can be set experimentally) probe packets can be used to obtain $\tilde{\Omega}_p$ and γ^p. Each probe packet's routing path is denoted as Γ^p, where $i = 1, 2, ..., N$. The probe packets have the same source and destination addresses as packet p, and the nodes traversed by packet p can be approximated by those traversed by the probe packets Γ^p. Packet p's approximate routing path can be obtained by combining all the probe packets' routing paths, given as below,

$$\tilde{\Omega}_p = \bigcup_{i=1}^{N} \Gamma_i^p \tag{22}$$

After obtaining the approximate routing path $\tilde{\Omega}_p$, the passing probabilities $\gamma^p = \{\gamma_{r,s}^p | (r,s) \in \tilde{\Omega}_p\}$ can be computed as follows,

$$\gamma_{r,s}^p = \frac{\sum_{i=1}^{N} \mathbf{1}\{(r,s) \in \Gamma_i^p\}}{N}, (r,s) \in \tilde{\Omega}_p \tag{23}$$

where $\mathbf{1}\{\cdot\}$ is the characteristic function such that when the event X is true, $\mathbf{1}\{X\} = 1$, otherwise, $\mathbf{1}\{X\} = 0$. The frequency of a node appearing in the approximate routing path can be used to estimate the probability that packet p actually passes through such a node. Therefore, a node's passing probability is the ratio of the number of times it appears in the approximate routing paths to the number of total probe packets (N in Eqn. 22).

Fig. 8 illustrates how the approximate routing paths and passing probabilities are computed. In this example, two probe packets need to be sent out and their source and destination addresses are $(3,3)$ and $(0,0)$, respectively. Thus, we shall have:

$$N = 2 \tag{24}$$

$$\Gamma_1 = \{(3,3),(2,3),(1,3),(0,3),(0,2),(0,1),(0,0)\} \tag{25}$$

$$\Gamma_2 = \{(3,3),(2,3),(1,3),(1,2),(1,1),(1,0),(0,0)\} \tag{26}$$

$$\tilde{\Omega} = \bigcup_{i=1}^{2} \Gamma_i = \{(3,3),(2,3),(1,3),(0,3),(0,2),(0,1),(0,0),(1,2),(1,1),(1,0)\} \tag{27}$$

$$\gamma_{3,3} = \gamma_{2,3} = \gamma_{1,3} = \gamma_{0,0} = \frac{2}{2} = 1 \tag{28}$$

$$\gamma_{0,3} = \gamma_{0,2} = \gamma_{0,1} = \gamma_{1,2} = \gamma_{1,1} = \gamma_{1,0} = \frac{1}{2} = 0.5 \tag{29}$$

After receiving the evidence $(\tilde{\Omega}_p, \gamma^p, L_p)$, the defense management unit updates all the PoMs of the nodes in set $\tilde{\Omega}_p$ according to Eqn. 30.

$$\begin{cases} K_p^t = \prod_{(r,s)\in\tilde{\Omega}_p} (1 - \alpha \cdot \gamma_{r,s}^p \cdot \mathbb{P}^t(H_{r,s})), \\ \mathbb{P}^{t+1}(H_{i,j}) = \mathbb{P}^t(H_{i,j}) [\frac{(1-K_p^t)(1-\alpha\gamma_{i,j}^p) + \alpha\gamma_{i,j}^p L_p (1-\mathbb{P}^t(H_{i,j}))}{(1-\alpha\gamma_{i,j}^p \mathbb{P}^t(H_{i,j}))(1-K_p^t)}] \end{cases} \tag{30}$$

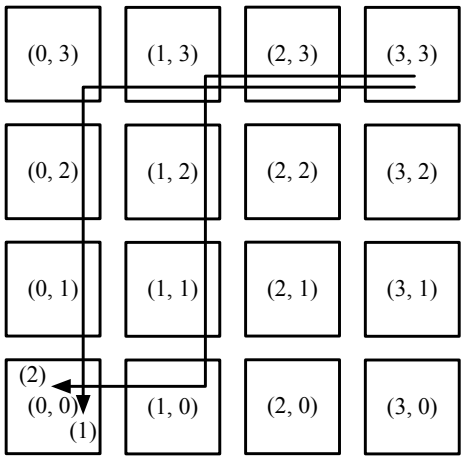


Fig. 8: An example of computing approximate routing path and passing probabilities.

The paths of the probe packets can be treated as a good approximation of the routing paths that the data packets actually flow through.

4.2 PBDS-A2

The number of malicious nodes is typically small, compared to the total number of the nodes in the chip. To find a better approximation to Eqn. 21, we assume that any packet encounters no more than one malicious node going from the source to the destination. This assumption is experimentally found to be true in the majority of cases. Our experiment also shows that when the percentage of malicious nodes is lower than 8%, the probability that a packet traverses at most one malicious node is higher than 87.5%. Correspondingly, Eqn. 21 can be simplified, giving rise to an approximate strategy, hereinafter referred as PBDS-A2.

Theorem 2. Assume that a packet sees no more than one malicious node as it is routed from the source to the destination. The PoMs can be updated by,

$$\begin{cases} K_p^t = \prod_{(r,s)\in\Omega_p} (1 - \alpha \cdot \mathbb{P}^t(H_{r,s})), \\ \mathbb{P}^{t+1}(H_{i,j}) \approx \mathbb{P}^t(H_{i,j}) [L_p \frac{\alpha}{1-K_p^t} + (1-L_p) \frac{1-\alpha}{K_p^t}] \end{cases} \tag{31}$$

Proof: Based on Eqns. 3, 6, and 7, the PoMs can be updated as follows,

$$\mathbb{P}^{t+1}(H_{i,j}) = L_p \frac{\mathbb{P}^t(L_p = 1|H_{i,j})\mathbb{P}^t(H_{i,j})}{\mathbb{P}^t(L_p = 1)} + (1-L_p) \frac{\mathbb{P}^t(L_p = 0|H_{i,j})\mathbb{P}^t(H_{i,j})}{\mathbb{P}^t(L_p = 0)} \tag{32}$$

Based on Eqns. 13 and 5, the terms $\mathbb{P}^t(L_p = 1|H_{i,j})$ and $\mathbb{P}^t(L_p = 0|H_{i,j})$ can be expanded as follows,

$$\mathbb{P}^t(L_p = 1|H_{i,j}) = \mathbb{P}^t(I_{i,j}^p|H_{i,j}) + \mathbb{P}^t(\bar{I}_{i,j}^p|H_{i,j})\mathbb{P}^t(\bigcup_{(r,s)\in\Omega_p,(r,s)\neq(i,j)} I_{r,s}^p) \tag{33}$$

$$\mathbb{P}^t(L_p=0|H_{i,j}) = \mathbb{P}^t(\bar{I}^p_{i,j}|H_{i,j})\mathbb{P}^t(\bigcap_{(r,s)\in\Omega_p,(r,s)\neq(i,j)} \bar{I}^p_{r,s}) \tag{34}$$

In Eqn. 33, $\mathbb{P}^t(L_p=1|H_{i,j})$ is the probability that packet p is tampered, given node (i,j) is HT-infected. Since packet p is assumed to pass at most one malicious node in its route, this probability can be approximated as the probability that node (i,j) tampers packet p. That is,

$$\mathbb{P}^t(L_p=1|H_{i,j}) \approx \mathbb{P}^t(I^p_{i,j}|H_{i,j}) = \alpha \tag{35}$$

In the same token, $\mathbb{P}^t(L_p=0|H_{i,j})$ in Eqn. 34, which is the probability that packet p is not tampered, given node (i,j) is HT-infected, can be approximated as the probability that node (i,j) does not tamper packet p and other nodes are not HT-infected.

$$\mathbb{P}^t(L_p=0|H_{i,j}) \approx \mathbb{P}^t(\bar{I}^p_{i,j}|H_{i,j}) = 1-\alpha \tag{36}$$

By substituting Eqns. 35 and 36 into Eqn. 32, we have,

$$\frac{\mathbb{P}^{t+1}(H_{i,j})}{\mathbb{P}^t(H_{i,j})} \approx L_p\frac{\mathbb{P}^t(I^p_{i,j}|H_{i,j})}{1-\mathbb{P}^t(L_p=0)} + (1-L_p)\frac{\mathbb{P}^t(\bar{I}^p_{i,j}|H_{i,j})}{\mathbb{P}^t(L_p=0)} \tag{37}$$

$$\begin{cases} K^t_p = \prod_{(r,s)\in\Omega_p}(1-\alpha\cdot\mathbb{P}^t(H_{r,s})), \\ \mathbb{P}^{t+1}(H_{i,j}) \approx \mathbb{P}^t(H_{i,j})[L_p\frac{\alpha}{1-K^t_p} + (1-L_p)\frac{1-\alpha}{K^t_p}] \end{cases} \tag{38}$$

The number of malicious nodes is typically smaller than the number of total nodes, and the assumption that there is one HT in the routing path holds true in most circumstances. Since PBDS-A2 requires calculating a simpler mathematical formula, the computation cost associated with the updating process in each iteration tends to be reduced.

4.3 PBDS-A3

Once methods PBDS-A1 and PBDS-A2 are combined, we arrive at PBDS-A3. In each detection iteration, the number of probe packets is reduced, and the simplified probability updating formula proposed in PBDS-A2 is used to calculate the PoMs. This method does reduce not only the runtime bandwidth overheads, but also the computation costs. The bandwidth overheads decrease because there are fewer probe packets travelling in the network, and the computation costs are reduced because the simplified probability updating formula is easy to calculate.

4.4 Approximation by Lookup Table

To further speed up the computation of Eqn. 31, a lookup table (LUT) is used, which helps avoid performing expensive multiplications and divisions. The iteration with LUT can be described as,

$$\mathbb{P}^{t+1}(H_{i,j}) = LUT(\mathbb{P}^t(H_{i,j}), L_p, K^t_p) \tag{39}$$

Which essentially transforms the computation problem to a search. The computation and construction of the LUT can be done off line using the probability updating formula given in Eqn. 31. At runtime, the LUT can be referred to through the index of $(\mathbb{P}^t(H_{i,j}), L_p, K^t_p)$ to retrieve the value of $\mathbb{P}^{t+1}(H_{i,j})$. In order to reduce the storage footprint, compression methods, such as Huffman coding [29], can be adopted.

While the proposed detection framework is demonstrated using packet tampering as the primary form of HT behavior, the underlying Bayesian inference model is more broadly applicable. The core requirement is the ability to obtain the routing paths information for the packets being analyzed. The framework can be extended to other forms of HTs, such as link disabling or denial-of-service, where the HT disrupts normal connectivity, and selective packet dropping, where certain packets are dropped without modification.

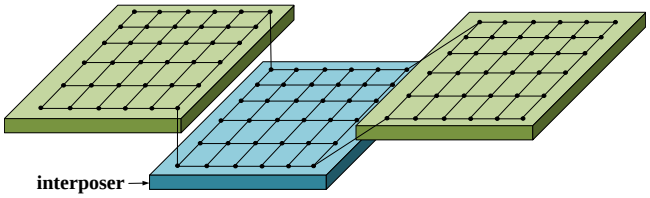


Fig. 9: Two chiplets each with 6×6 mesh are connected by a 6×6 network on interposer.

5 Experimental Results

5.1 Experimental Setup

Experiments are performed to evaluate the accuracy of the proposed HT detection methods, using a cycle-accurate event-driven many-core simulator [26], with DSENT 3.0 [32] which can simulate the electric router/links integrated as the power model. Table 4 lists the benchmarks used in the simulation, and they are selected from PARSEC and SPLASH-2. A set of random traffic which covers a wide range of traffic scenarios is also used.

DSENT is exclusively used for area/power modeling, not for functional or timing simulations. Functional and timing models simulation is performed using a cycle-accurate event-driven many-core simulator as indicated in Section 5.1. To enhance relevance, we have added one additional experiment using a chiplet-based network, shown in Fig. 9, where two chiplets, each configured as a 6×6 mesh, are connected via a 6×6 network on interposer. Additionally, we include Fig. 10, which shows the detection rate as a function of packet injection rate. One can see that the injection rate leads to improved detection accuracy, approaching 100% under high-traffic conditions.

The simulated architecture is a shared memory structure, with each core having its own L1 cache and a shared L2 cache bank. A tile is connected to a local network interface and a router, and together they form one NoC node. In the following experiments, we select a 8×8 2D mesh with adaptive routing as the underline NoC architecture. Multi-threaded applications can have their threads run on different cores, and these threads communicate with each other through the NoC. Table 3 lists the simulation configuration.

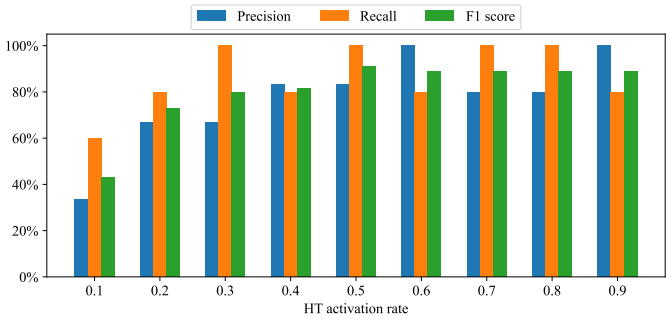


Fig. 10: The detection rate when packet injection rate changes.

TABLE 3: System Configurations for the Simulation

Number of processors	16/64/256 (Alpha ISA 64 compatible)
Fetch/Decode/Commit size	4/4/4
ROB size	64
L1 D cache(private)	16 KB, two-way, 32B line, two cycles, two ports, dual tags
L1 I cache(private)	32 KB, two-way, 64B line, two cycles
L2 cache(shared) MESI	64 KB slice/node, 64B line
protocol	six cycles, two ports
Main memory size	2 GB, latency 200 cycles
On-chip network parameters	
NoC flit size	64 bits
Data packet size	5 flits
Meta packet size	1 flit
NoC latency	Router: two cycles; link: one cycle
Number of NoC Virtual Channels (VC)	4
NoC buffer size	5 × 5 flits
Routing algorithm used in NoC	Adaptive Routing [46] /XY Routing
DICN flit size	16 bits
DICN latency	Router: two cycles; link: one cycle
Number of DICN Virtual Channels (VC)	1
DICN buffer size	5 × 2 flits
Routing algorithm used in DICN	XY Routing

TABLE 4: Benchmarks Used in the Simulation

Real benchmarks	
PARSEC	streamcluster, swaptions, ferret, fluidanimate, blackscholes, freqmine, dedup, canneal, vips
SPLASH-2	barnes, raytrace
Random benchmarks	
Packet Injection Rate (PIR)	[0, 50] packets/cycle
Distribution of destination nodes	Random (uniform); bit reverse; bit complement

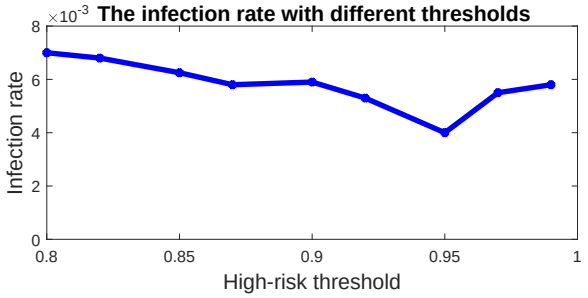


Fig. 11: The infection rates when adopting different thresholds.

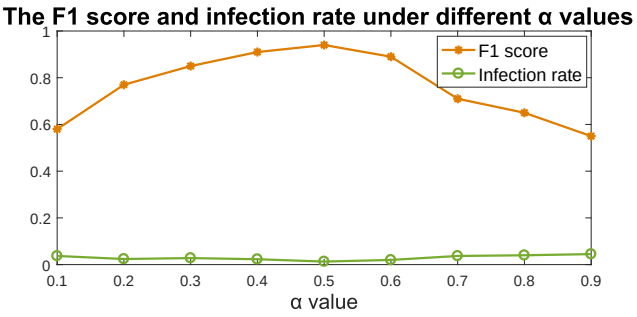


Fig. 12: The F1 scores and infection rates under different α values needed for PoM updating.

The E2E authentication unit adopts a tag-based detection method as in [9], and this method is taken as the baseline for comparison. The conditionally triggered HT is used to tamper the passing packets, and the HT is activated by an internal counter. In our experiments, every NoC is set to have 8% malicious nodes. As Fig. 11 shows, when the threshold is 0.95, the infection rate falls to the lowest level. Correspondingly, the threshold value is set to be 0.95 throughout all the experiments, and any node whose PoM exceeds this value will be marked as a high-risk node.

As Fig. 12 shows, when α, a parameter needed in updating the PoM (Eqn. 21), has a value of 0.5, the F1 score of the detection method is 0.94, and the infection rate with countermeasure applied is at its minimal of 0.015, which means the HT attacks are effectively defeated. We have tested this α value under different configurations, and in all cases, the infection rates are below 0.05. In what follows, α is fixed to be 0.5 for PoM updating.

Two figures of merit are adopted to quantify the detection effect.

1) F1 score: PBDS can be regarded as a binary classification method, and the proposed method allocates prediction results to each node, regardless whether a node is malicious or not. Let **HighRiskNodes** be the number of nodes that are identified by the DMU as malicious nodes, and TP be the number of true malicious nodes. F1 score is defined as,

$$\text{precision} = \frac{TP}{\|\mathbf{HighRiskNodes}\|}, \text{recall} = \frac{TP}{m}$$
$$F1 = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \quad (40)$$

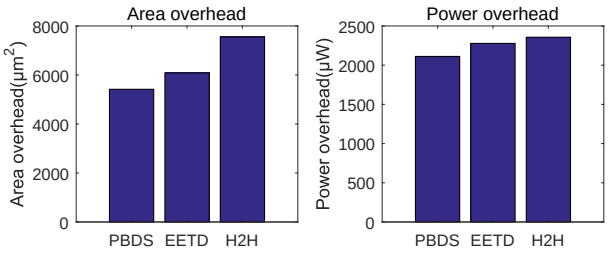


Fig. 13: The area, power, timing overheads and infection rates of the three methods.

2) Infection rate: It is the ratio of the number of infected packets to the total number of packets transmitted.

The infection rate is computed after each iteration, and the F1 score is computed at the end of each experiment.

5.2 Comparisons with the EETD and H2H Methods

We compare EETD [6], H2H [19], and PBDS in terms of the area overhead, power overhead, and the infection rate of data packets.

The area overhead of introducing PBDS at each node is $5416.4\mu m^2$, which includes the area overhead of the proposed 4 PWUs, the router (with flit width of 16 bits) in DICN [43], and the bypass channels (with flit width of 64 bits) [47], as reported by Synopsys Design Compiler under 45nm TSMC library. A router with 4 virtual channels and FIFO depth of 5 flits has a total area of $71,814\mu m^2$ obtained from DSENT. One can see that the per-node area of PBDS actually accounts for only 7.5% of a single router. The per-node area overhead of EETD with the DICN [43] and countermeasure (bypass channels [47]) is $6081.8\mu m^2$, and the per-node area overhead of H2H with its own countermeasure is $7551.4\mu m^2$. As Fig. 13 shows, the total area overhead of PBDS is about 11% lower than that of EETD and 28.3% lower than that of H2H.

The power overhead of introducing PBDS with the countermeasure at each node is $2111.8\mu W$, as reported by Synopsys Design Compiler for the same CMOS technology node. A router consumes a total power of $31,881\mu W$; that is, per-node power consumption of PBDS only accounts for about 6.6% of a single router. In comparison, the total power overhead of PBDS is about 7% lower than that of EETD and 10.4% lower than that of H2H.

Fig. 14 (a) shows the normalized timing overheads of PBDS, EETD, and H2H methods under different NoC sizes. The execution times of the three methods all increase with the size of the NoC. For an 8×8 NoC, the normalized run time overhead of PBDS is 22% higher than that of EETD but 67% lower than that of H2H. Regardless of the NoC size, the run time overhead of PBDS is always much lower than that of H2H, and just slightly higher than that of EETD. Since H2H method needs to inspect packet at each hop of the routing path, its time overhead is much higher. PBDS needs to run several iterations to accumulate the PoM of each node, while EETD does not need to; as a result, the timing overhead of PBDS is slightly higher than that of EETD. Fig. 14 (b) shows the timing overheads of the three methods with different traffic patterns. One can see that the traffic patterns have very limited impact on the timing overheads across all three methods.

Fig. 15 (a) shows the infection rates of the three methods in different NoC sizes. PBDS achieves the lowest infection rate (i.e., 0.013) in an 8×8 NoC. The infection rate of PBDS is much lower than that of EETD, and it is similar to that of H2H. EETD performs poorly in terms of infection rates, since it is unable to detect HTs if they go back to hibernation after a short period of activation. Fig. 15 (b) shows the infection rates of the three methods in the presence of different traffic patterns. Again, one can see that the traffic patterns have very limited impacts on the infection rates across all three methods.

5.3 Evaluating PBDS, PBDS-A1, PBDS-A2, and PBDS-A3

These four methods have different assumptions:

- PBDS is best suited for systems using deterministic routing algorithms, where complete packet routing paths are readily available. This method is appropriate when bandwidth and computation overhead are not limiting factors.
- PBDS-A1 is more suitable for systems using adaptive algorithms, where routing path might change dynamically. It uses probe packets to approximate routing paths, making it more practical for real-world deployment with manageable bandwidth costs.
- PBDS-A2 is preferred for scenarios where there is only one HT is expected along any packet's path. It simplifies the Bayesian update process, resulting in reduced computational complexity without compromising detection accuracy.
- PBDS-A3 is more effective when both bandwidth and computational resources are limited, and the single-HT assumption holds. It combines the use of probe packets with the simplified assumption of PBDS-A2 to offer a lightweight yet accurate detection method.

Fig. 16 (a) shows the infection rates when PBDS takes different HT activation rates (i.e., $0.1 \sim 0.9$). The infection rate is about 0.4663 when the NoC is under an attack that sees the HTs are switched ON and OFF but without applying any countermeasure. The infection rate drops by two orders of magnitude, to about $0.0031 \sim 0.0438$, when the proposed PBDS assumes different HT activation rates, which is about $90.61\% \sim 99.34\%$ lower, compared to the case without applying a countermeasure. Overall speaking, HT activation rate has a relatively minor effect on the infection rate reduction.

Fig. 16 (b) shows the infection rates of the proposed methods against the HTs that are conditionally triggered. HT activation rate in this experiment is set to 0.5. (i.e., there is an equal probability that a node is tampered vs. not). The infection rate is about 0.47 when the NoC is under an attack by the conditionally triggered HTs. The infection rate decreases to about 0.0039 when PBDS is applied, which is about 99.2% lower than the case that does not involve any countermeasure. This infection rate is even lower than that reported in previous section, and this is due to the adaptive routing algorithm used in this experiment. When an adaptive routing algorithm is

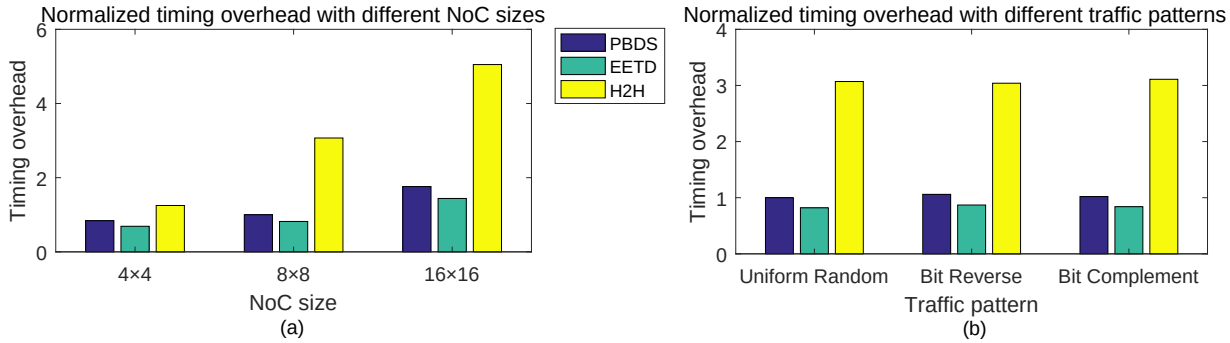


Fig. 14: The normalized timing overheads of the three methods with different NoC sizes.

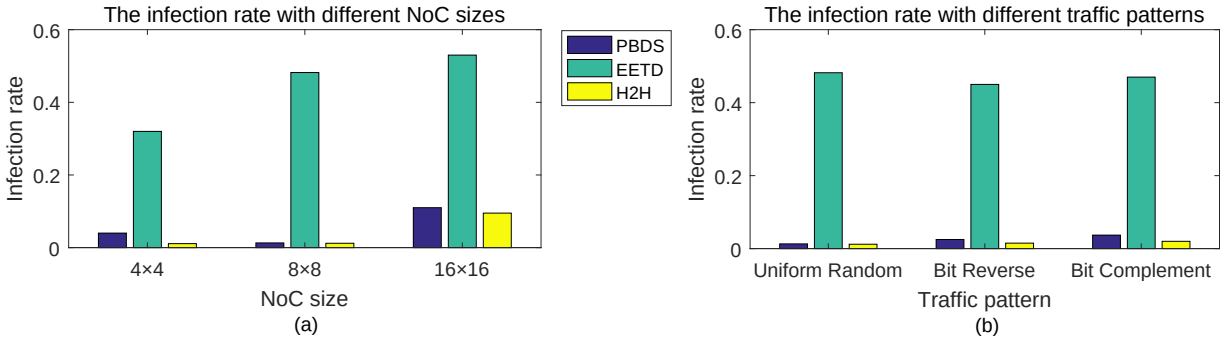


Fig. 15: The infection rates of the three methods with different traffic patterns.

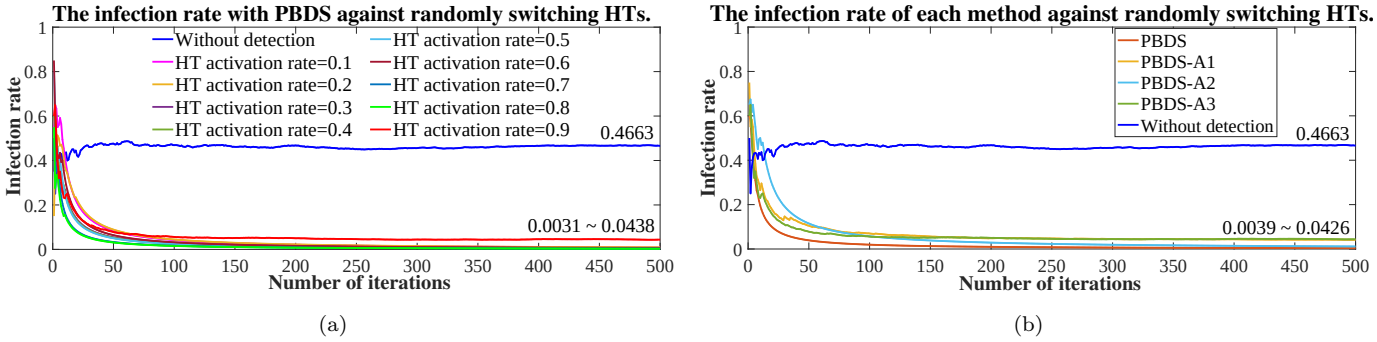


Fig. 16: (a) The infection rates with PBDS against those of conditionally triggered HTs. (b) The infection rates when different configurations of the proposed method are applied to fight against the conditionally triggered HTs.

adopted, the DMU can check more nodes in one detection cycle. The infection rates drop to about 0.0412, 0.0115, and 0.0426, when the PBDS-A1, PBDS-A2 and PBDS-A3 are respectively applied, which are about 91.4%, 97.7%, and 90.9% lower than the case without applying any countermeasure. The infection rates decrease at a modest rate when moving from PBDS-A1 to PBDS-A2, and finally to PBDS-A3. As the DMU uses approximate routing paths and formula to update the PoMs, the DMU cannot obtain accurate information for the detection. Correspondingly, the DMU has to spend more time on detection, making the infection rates decrease at a slower pace.

Fig. 17 shows the precisions, recalls, and F1 scores of these proposed methods under different HT activation rates. For PBDS, the recall is higher than 0.9 in most cases, even when the values of HT activation rate are small. The reason is that these values are generated from many iterations. The F1 score is high when HT activation rate ranges from 0.1 to 0.9. Note that the precision is less than 0.5 when HT activation rate is 0.1, but it soars when HT activation rate is 0.2. With the increase of HT activation rate, the HTs are more easily detected. The PBDS can maintain a good performance when HT activation rate fluctuates greatly. For PBDS-A1 and PBDS-A3, the recalls are higher than 0.9 in most cases and the precisions are lower compared to that of PBDS. For PBDS-A2, the precision and recall are higher than 0.95 in most cases. The PBDS-A1, PBDS-A2, and PBDS-A3 all maintain a good result when HT activation rate fluctuates greatly.

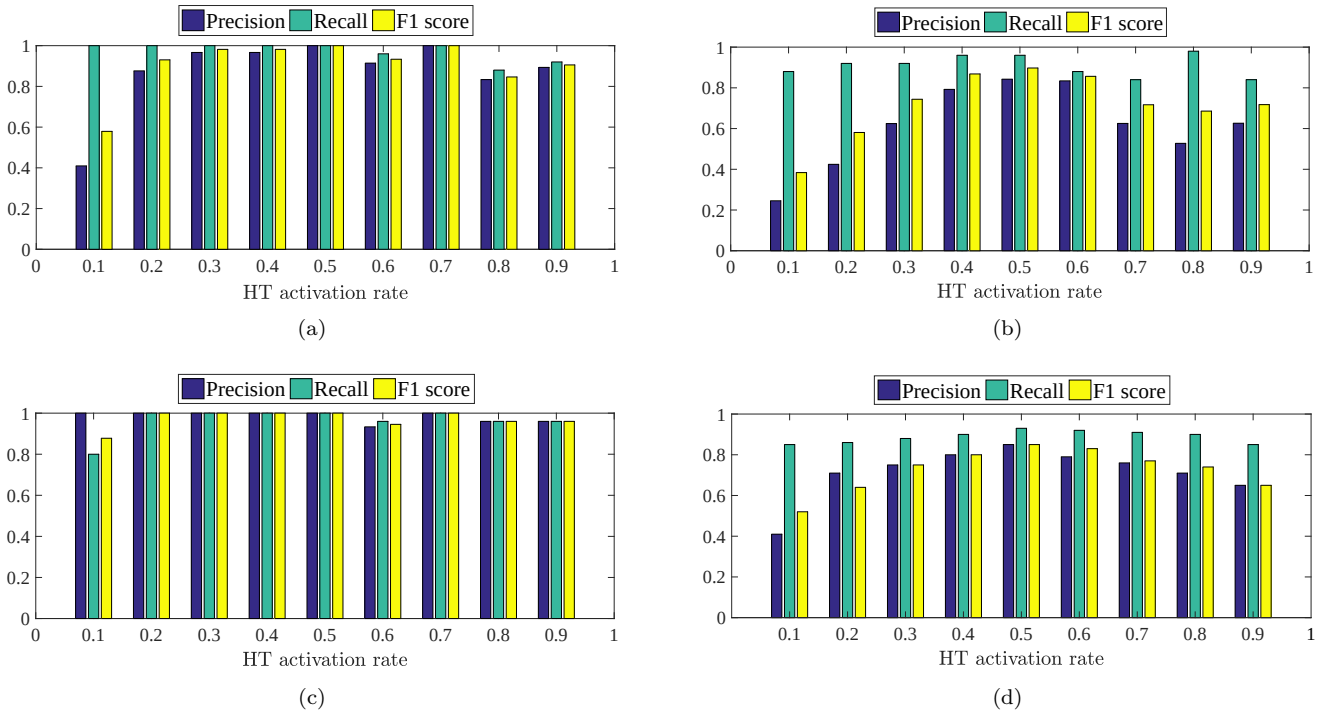


Fig. 17: The precisions, recalls and F1 scores with PBDS (a), PBDS-A1 (b), PBDS-A2 (c), and PBDS-A3 (d) against those of conditionally triggered HTs under different HT activation rates.

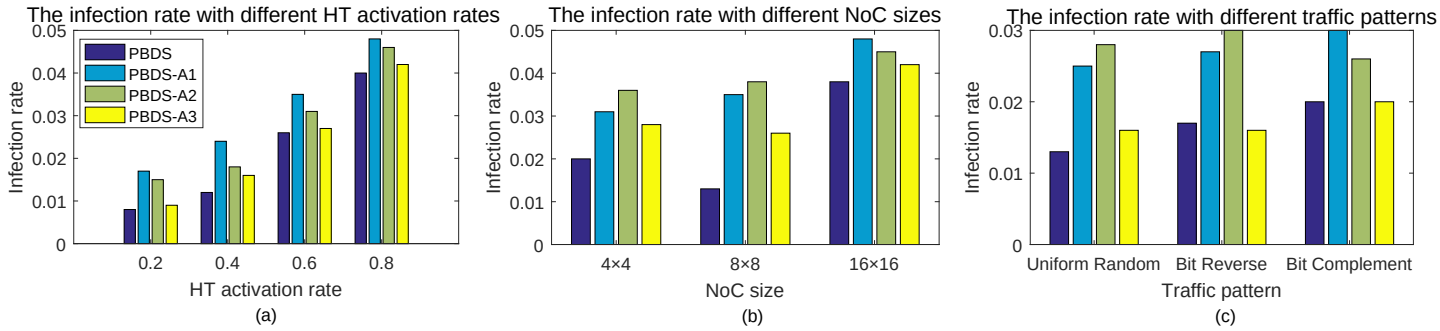


Fig. 18: The infection rates of PBDS, PBDS-A1, PBDS-A2, and PBDS-A3 with different HT activation rates (a), different NoC sizes (b), and different traffic patterns (c).

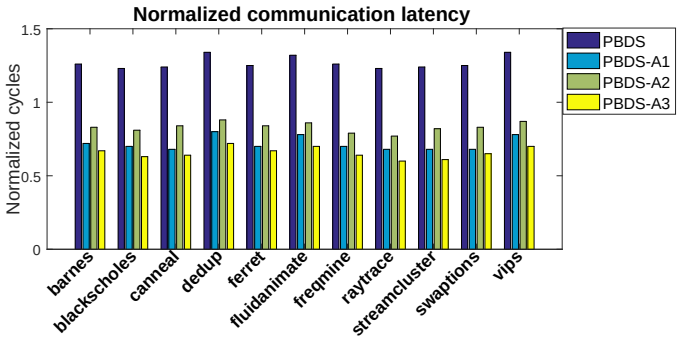


Fig. 19: The normalized communication latencies when running real benchmarks.

Fig. 18 (a) shows the infection rates of PBDS, PBDS-A1, PBDS-A2, and PBDS-A3 with respect to various HT activation rates. Of all four methods, their infection rates are found to increase with the increase of HT activation rates. All four methods record low infection rates, with PBDS showing the lowest. Overall speaking, the performance of PBDS-A3 is very close to that of PBDS. Fig. 18 (b) shows the infection rates of the four methods under different NoC sizes. For an 8×8 NoC , the infection rates of the four methods are the lowest. As the NoC size jumps to 16×16, there is a slight increase of the infection rates for the four methods. This is due to the fact that as the NoC size scales up, the detection process needs extra amount of time to detect HTs. Fig. 18 (c) shows the infection rates of the four methods with respect to different traffic patterns. The results confirm that the traffic patterns have a very limited impact on the infection rates

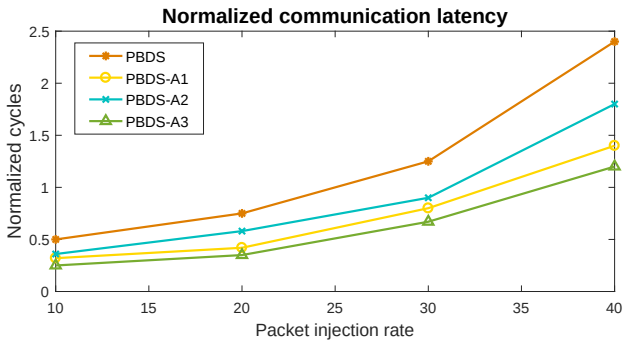


Fig. 20: The normalized communication latencies for random benchmarks respect to different packet injection rates (the number of injected packets every 10 cycles).

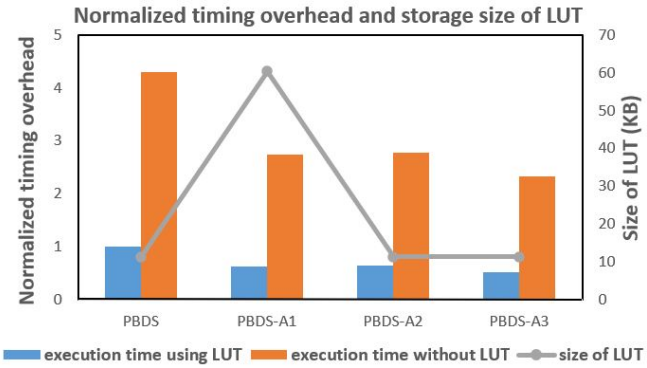
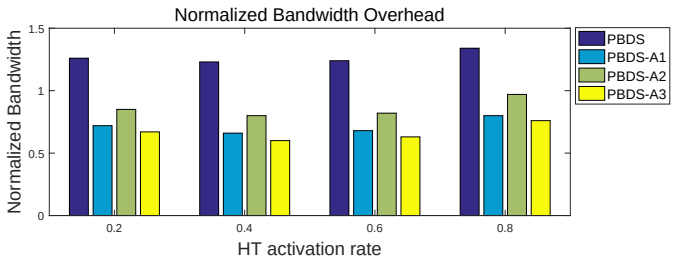
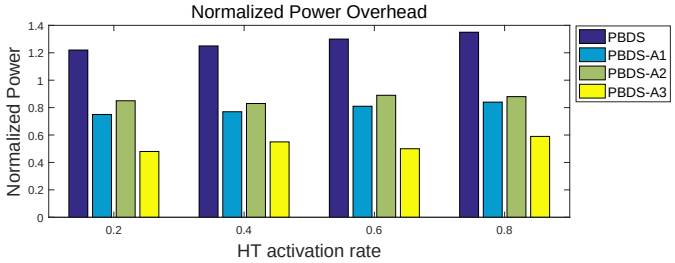


Fig. 21: The normalized timing overheads and lookup table storage overheads when PBDS, PBDS-A1, PBDS-A2, and PBDS-A3 are adopted.



(a)



(b)

Fig. 22: The normalized bandwidth overheads (a) and power overheads (b) of PBDS, PBDS-A1, PBDS-A2, and PBDS-A3.

across all four methods.

Fig. 19 shows the normalized communication latencies for running different real benchmarks when various approaches are applied. The communication latencies of PBDS-A1, PBDS-A2, and PBDS-A3 are respectively about 42.9%, 34.1%, and 46.8% lower than that of PBDS on average. Fig. 20 shows the normalized communication latencies of the NoC while running the random benchmarks and injecting the packets at different rates. With the increase of packet injection rate, the latency of the NoC increases gradually. As a direct result of adopting the approximate designs, PBDS-A1, PBDS-A2, and PBDS-A3 all have lower latencies than PBDS.

Fig. 21 shows the timing and lookup table storage overheads of the proposed strategies. The normalized execution time overheads of PBDS, PBDS-A1, PBDS-A2, and PBDS-A3 using LUTs are 1, 0.62, 0.64 and 0.52, respectively, which are about 76.7%, 77.3%, 76.9%, and 77.8% lower than that of the case without LUTs. The sizes of LUTs employed for PBDS, PBDS-A2, and PBDS-A3 are all about 11.3KB, while the LUT for PBDS-A1 has a size of 60.4KB. The reason that the PBDS-A1 needs a larger LUT is because PBDS-A1 requires more parameters to be stored than the other two schemes.

Fig. 22 shows the normalized bandwidth (measured by the injection rate of the probe packets) and power overheads of PBDS, PBDS-A1, PBDS-A2, and PBDS-A3 under different HT activation rates. The bandwidth overheads of PBDS-A1, PBDS-A2, and PBDS-A3 are respectively about 45.2%, 25.8%, and 49.2% lower than that of PBDS. The power overheads of PBDS-A1, PBDS-A2, and PBDS-A3 are about 37.7%, 31.5%, and 61.5% lower than that of PBDS, respectively.

The detection capabilities of PBDS-A1, PBDS-A2, and PBDS-A3 are very close to that of PBDS. However, with much lower bandwidth and power overheads, the proposed approximate methods are more practical in real implementations.

5.4 Evaluating the Defense Hierarchy

Experiments have been run to evaluate the impacts of the hierarchical defense strategy as described in Section 3.5.

For a $s \times s$ NoC, the system can be partitioned into clusters with sizes of 2×2, 4×4, ..., $q \times q$ ($q = 2^z$,$q \leq \lceil s/2 \rceil$). In the experiments, different partitioning methods are tested, and the partition method (cluster size) with the lowest number of detection iterations is selected. For example, when the NoC size is 16×16, the experimental results show the system with the cluster size of 8×8 has the minimum detection iterations; thus, the 16×16 NoC system is partitioned into 4 clusters with each cluster being of size 8×8. In each of the 8×8 cluster, one of the nodes can be designated as the local defense manager. After extensive experiment evaluations, one of the nodes can be the local defense manager without difference in detection iterations.

The results for a 16×16 network are tabulated in Table 5. The number of detection iterations of the original system (no clustering) is found to be $1.53\times$ of the hierarchical approach, even though the infection rates of both methods are fairly

close. Such experiment clearly indicates that the hierarchical approach can help speed up the detection.

TABLE 5: Experimental Results of the Hierarchical Detection Approach

	Original system	The hierarchical approach
Normalized detection iterations	1.53	1
Infection rates	0.042	0.037

6 Conclusion

In this paper, a new lightweight online HT detection strategy based on Bayesian analysis was proposed to fight against packet tampering attacks in the NoC. Essentially, a low-cost end-to-end (E2E) authentication method was adopted to check the integrity of any packet flowing through a router, and each node is measured by a probability of being a malicious node (PoM). The DMU iteratively updates the PoMs based on every packet's routing path and its authentication result. The nodes whose PoMs exceeding a threshold are marked as high-risk nodes, and all the subsequent packets will bypass these identified high-risk nodes. Experiments confirmed that the proposed probability-based method, with a very low area and power consumption overheads, can accurately locate the malicious nodes, and the infection rates dropped to a level that is comparable to the super expensive hop-to-hop (H2H) authentication method.

References

[1] K. Chrysanthou, P. Englezakis, A. Prodromou, A. Panteli, C. Nicopoulos, Y. Sazeides, and G. Dimitrakopoulos, "An online and real-time fault detection and localization mechanism for network-on-chip architectures,"ACM Trans. Architecture and Code Optimization, vol. 13, no. 2, pp. 22:1–22:26, 2016.

[2] A. Kulkarni, Y. Pino, and T. Mohsenin, "SVM-based real-time hardware Trojan detection for many-core platform," in Int'l Symp. Quality Electronic Design, 2016, pp. 362–367.

[3] H. Li, Q. Liu, and J. Zhang, "A survey of hardware Trojan threat and defense," Integration, the VLSI Journal, vol. 55, pp. 426–437, 2016.

[4] W. Burleson, O. Mutlu, and M. Tiwari, "Who is the major threat to tomorrow's security? You, the hardware designer," in Design Automation Conf., 2016, pp. 1–5.

[5] Y. Zhao, X. Wang, Y. Jiang, M. Yang, A. K. Singh, and T. Mak, "On a new hardware Trojan attack on power budgeting of many core systems," in IEEE Int'l Conf. System-on-Chip, 2018, pp. 58–64.

[6] M. Hussain, A. Malekpour, H. Guo, and S. Parameswaran, "EETD: an energy efficient design for runtime hardware trojan detection in untrusted network-on-chip," in Int'l Symp. VLSI, 2018, pp. 345–350.

[7] T. Boraten and A. Kodi, "Mitigation of hardware Trojan based denial-of-service attack for secure NoCs," J. Parallel and Distributed Computing, vol. 111, pp. 24–38, 2018.

[8] T. Boraten and A. Kodi, "Packet security with path sensitization for NoCs," in Design, Automation & Test in Europe Conf. & Exhibition, 2016, pp. 1136–1139.

[9] M. Hussain and H. Guo, "Packet leak detection on hardware Trojan infected NoCs for MPSoC systems," in Int'l Conf. Cryptography, Security and Privacy, 2017, pp. 85–90.

[10] S. K. Haider, C. Jin, and M. van Dijk, "Advancing the state-of-the art in hardware Trojans design," in Int'l Midwest Symp. Circuits and Systems, 2017, pp. 823-826.

[11] M. Tehranipoor and F. Koushanfar, "A survey of hardware Trojan taxonomy and detection," IEEE Design & Test of Computers, vol. 27, no. 1, pp. 10–25, 2010.

[12] S. Bhunia, M. S. Hsiao, M. Banga, and S. Narasimhan, "Hardware Trojan attacks: threat analysis and countermeasures," Proc. IEEE, vol. 102, no. 8, pp. 1229–1247, 2014.

[13] A. Ganguly, M. Y. Ahmed, and A. Vidapalapati, "A denial-of-service resilient wireless NoC architecture," in ACM Great Lakes Symp. on VLSI, 2012, pp. 259–262.

[14] S. R. Hasan, S. F. Mossa, C. Perez, and F. Awwad, "Hardware Trojans in asynchronous FIFO-buffers: from clock domain crossing perspective," in IEEE Int'l Midwest Symp. Circuits and Systems, 2015, pp. 1–4.

[15] T. Boraten and A. K. Kodi, "Mitigation of denial of service attack with hardware Trojans in NoC architectures," in IEEE Int'l Symp. Parallel and Distributed Processing, 2016, pp. 1091–1100.

[16] C. G´omez, M. E. G´omez, P. L´opez, and J. Duato, "Reducing packet dropping in a bufferless NoC," in European Conf. Parallel Processing, 2008, pp. 899–909.

[17] S. T. King, J. Tucek, A. Cozzie, C. Grier, W. Jiang, and Y. Zhou, "Designing and implementing malicious hardware." USENIX Workshop on Large-Scale Exploits and Emergent Threats, vol. 8, pp. 1–8, 2008.

[18] K. Yang, M. Hicks, Q. Dong, T. Austin, and D. Sylvester, "A2: analog malicious hardware," in IEEE Symp. security and privacy, 2016, pp. 18–37.

[19] Q. Yu and J. Frey, "Exploiting error control approaches for hardware Trojans on network-on-chip links," in Int'l Symp. Defect and Fault Tolerance in VLSI and Nanotechnology Systems, 2013, pp. 266–271.

[20] A. Waksman and S. Sethumadhavan, "Silencing hardware backdoors," in IEEE Symp. Security and Privacy, 2011, pp. 49–63.

[21] M. K. JYV, A. K. Swain, S. Kumar, S. R. Sahoo, and K. Mahapatra, "Run time mitigation of performance degradation hardware Trojan attacks in network on chip," in IEEE Symp. VLSI, 2018, pp. 738–743.

[22] M. Beaumont, B. Hopkins, and T. Newby, "Hardware Trojans prevention, detection, countermeasures (a literature review)," DTIC Document, Tech. Rep., 2011.

[23] J. Frey and Q. Yu, "A hardened network-on-chip design using runtime hardware Trojan mitigation methods," Integration, the VLSI Journal, vol. 56, pp. 15–31, 2017.

[24] J. Khichar and S. Choudhary, "Fault aware adaptive routing algorithm for mesh based NoCs," in Int'l Conf. Inventive Computing and Informatics, 2017, pp. 584-589.

[25] M. Kayaalp, N. Abu-Ghazaleh, D. Ponomarev, and A. Jaleel, "A high-resolution side-channel attack on last-level cache," in Design Automation Conf., 2016, pp. 1-6.

[26] X. Wang, M. Yang, Y. Jiang, P. Liu, M. Daneshtalab, M. Palesi, and T. Mak, "On self-tuning networks-on-chip for dynamic network flow dominance adaptation," ACM Trans. Embedded Computing Systems, vol. 13, no. 2s, pp. 73:1–73:21, 2014.

[27] L. Daoud and N. Rafla, "Routing aware and runtime detection for infected network-on-chip routers," in Int'l Midwest Symp. Circuits and Systems, 2018, pp. 775-778.

[28] K. Wang, H. Zheng, and A. Louri, "TSA-NoC: learning-based threat detection and mitigation for secure network-on-chip architecture," IEEE Micro, vol. 40, no. 5, pp. 56-63, 2020.

[29] D. Tamir, "Delta-Huffman coding of unbounded integers," in Data Compression Conf., 2018, pp. 428-428.

[30] V. Y. Raparti and S. Pasricha, "Lightweight mitigation of hardware Trojan attacks in NoC-based manycore computing," in Design Automation Conf., 2019, pp. 1-6.

[31] A. P. Deb Nath, S. Boddupalli, S. Bhunia and S. Ray, "Resilient system-on-chip designs with NoC fabrics," IEEE Trans. Information Forensics and Security, vol. 15, pp. 2808-2823, 2020.

[32] C. Sun, C.H. Chen, G. Jurian, L. Wei. J. Miller, A. Agarwal, L.-S. Peh, V. Stojanovic, "DSENT - A tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling," in Int'l Symp. Networks-on-Chip, 2012, pp. 201-210.

[33] X. Chen, L. Wang, Y. Wang, Y. Liu and H. Yang, "A general framework for hardware Trojan detection in digital circuits by statistical learning algorithms," IEEE Trans. Computer-Aided

Design of Integrated Circuits and Systems, vol. 36, no. 10, pp. 1633-1646, 2017.
[34] R. Gayatri, Y. Gayatri, C. Mitra, S. Mekala and M. Priyatharishini, "System level hardware Trojan detection using side-channel power analysis and machine learning," in Int'l Conf. Communication and Electronics Systems, 2020, pp. 650-654.
[35] S. Charles and P. Mishra, "A survey of network-on-chip security attacks and countermeasures," ACM Computing Surveys, vol. 54, no. 5, pp. 101:1-101:36, 2021.
[36] Weize Yu, O. A. Uzun and S. Köse, "Leveraging on-chip voltage regulators as a countermeasure against side-channel attacks," in Design Automation Conf., 2015, pp. 1-6.
[37] S. Charles, Y. Lyu and P. Mishra, "Real-time detection and localization of DoS attacks in NoC based SoCs," in Design, Automation & Test in Europe Conf. & Exhibition, 2019, pp. 1160-1165.
[38] R. Elnaggar, K. Chakrabarty, "Machine learning for hardware security: opportunities and risks," J. Electron Test, vol. 34, pp. 183–201, 2018.
[39] F. K. Lodhi, S. R. Hasan, O. Hasan and F. Awwadl, "Power profiling of microcontroller's instruction set for runtime hardware Trojans detection without golden circuit models," in Design, Automation & Test in Europe Conf. & Exhibition, 2017, pp. 294-297.
[40] Y. Zhao, X. Wang, Y. Jiang, L. Wang, M. Yang, A. K. Singh and T. Mak, "On hardware-trojan-assisted power budgeting system attack targeting many core systems," in J. System Architectures, 2020, vol. 109, pp. 101757.
[41] L. Zhang, X. Wang, Y. Jiang, M. Yang, T. Mak, A. K. Singh, "Effectiveness of HT-assisted sinkhole and blackhole denial of service attacks targeting mesh networks-on-chip," in J. System Architectures, 2018, vol. 89, pp. 84-94.
[42] X. Meng, R. Hassan, S. M. P. Dinakrrao and K. Basu, "Can overclocking detect hardware Trojans?," in IEEE Int'l Symp. Circuits and Systems, 2021, pp. 1-5.
[43] A. P. Deb Nath, S. Boddupalli, S. Bhunia and S. Ray, "Resilient system-on-chip designs with NoC fabrics," EEE Trans. Information Forensics and Security, vol. 15, pp. 2808-2823, 2020.
[44] J. Sepúlveda, A. Zankl, D. Flórez, G. Sigl, "Towards protected MPSoC communication for information protection against a malicious NoC," in Procedia Computer Science, vol. 108, pp. 1103-1112, 2017.
[45] M. K. Puthal, V. Singh, M. S. Gaur and V. Laxmi, "C-Routing: an adaptive hierarchical NoC routing methodology," in Int'l Conf. VLSI and System-on-Chip, 2011, pp. 392-397.
[46] A. Menon, L. Zeng, X. Jiang and T. Watanabe, "Adaptive look ahead algorithm for 2-D mesh NoC," IEEE Int'l Advance Computing Conference, 2015, pp. 299-302.
[47] P. Darbani and H. R. Zarandi, "A reconfigurable network-on-chip architecture to improve overall performance and throughput," Iranian Conf. Electrical Engineering, 2014, pp. 943-948.