On Hardware-Trojan-Assisted Power Budgeting System Attack Targeting Many Core Systems ☆

Yiming Zhao[a], Xiaohang Wang[a,*], Yingtao Jiang[b], Liang Wang[c], Mei Yang[b], Amit Kumar Singh[d], Terrence Mak[e]

[a]*School of Software Engineering, South China University of Technology, Guangzhou, China*
[b]*Department of Electrical and Computer Engineering, University of Nevada, Las Vegas, USA*
[c]*Institute of Microelectronics, Tsinghua University, Beijing, China*
[d]*School of Computer Science and Electronic Engineering, University of Essex, Colchester, UK*
[e]*School of Electronics and Computer Science, University of Southampton, Southampton, UK*

Abstract

In a modern many-core chip, as all the cores are constantly competing for their shares of power out of the maximum power available to the chip, a sound power budgeting scheme is needed to efficiently allocate power to achieve the highest possible overall system performance. When a core is poised to run some applications, it has to request its power budget by sending a series of data packets, routed typically through an on-chip communication network infrastructure, to a specific core designated as the global manager that makes its power allocation decision based on all the budget requests it receives and its assessment of each cores potential contribution to the overall system performance. This power budgeting scheme is shown, in this paper, to be extremely vulnerable to stealthy false-data attacks, which can cause catastrophic denial of service (DoS) effects. Essentially, when a power budget request packet is routed through a Trojan-infected network-on-chip node, such as a router, the power budget request can be secretly modified by the Trojan. The global manager then tends to make really bad power budget allocation decisions with all the tampered power requests it received. That is, legitimate applications will be victimized with lower power budgets than what they initially asked for, and thus, could suffer serious performance degradation; malicious applications, on the other hand, may be entitled to high power budgets and thus see performance boost that they do not deserve. Our study has revealed that this new type of DoS attack can be initiated and sustained by a simple hardware Trojan (HT) circuit that is extremely hard to be detected due to its low silicon footprint and short activation time. The effects of this new DoS attack are simulated using a network model, and all the major parameters and factors that impact the attack effects are identified and quantified. Should the HTs can be intelligently turned ON/OFF following a scheme based on Q-learning, we further demonstrate the attacks can undermine the best countermeasures against power budgeting system attack as suggested in this paper, which gives rise to a need for further research in this regard.

Keywords: Network-on-chip, hardware Trojan, power budgeting

1. Introduction

Security of servers, data centers, mobile computing and Internet of Things is largely dependent on the security of the many-core chips that these facilities/systems have adopted. Unfortunately, many-core chips are susceptible to be attacked by hardware Trojans (HT) which can be easily implanted to the chips at any stage from design to manufacturing, as chip designing and manufacturing is going global, and licensing of third party IP cores is becoming commonplace in many-core chip designs. In contrast to a modern many-core chip with billions of transistors and complex functionalities, an HT circuit has an extremely low transistor count, making it hardly visible, and thus difficult to be detected by most known offline HT detection methods [1, 2, 3, 4, 5, 6, 7]. If many-core devices infected with HTs finally get deployed, they can create catastrophic effects, including dangerous security breach and severe performance degradation [8, 9, 10, 11]. HTs have been designed to explore many different types of vulnerabilities of a many-core chip. In this paper, we show one such vulnerability that roots in the power budgeting

☆This research program is supported by the Natural Science Foundation of China No. 61376024 and 61306024, Natural Science Foundation of Guangdong Province 2015A030313743 and 2018A030313166, Special Program for Applied Research on Super Computation of the NSFC-Guangdong Joint Fund (the second phase), and the Science and Technology Research Grant of Guangdong Province No. 2016A010101011 and 2017A050501003, Pearl River S&T Nova Program of Guangzhou No. 201806010038, and Tip-top Scientific and Technical Innovative Youth Talents of Guangdong special support program (No. 2014TQ01X590).

*Corresponding author

Email addresses: `yimingzhao3@gmail.com` (Yiming Zhao), `xiaohangwang@scut.edu.cn` (Xiaohang Wang), `yingtao.jiang@unlv.edu` (Yingtao Jiang), `lwang@link.cuhk.edu.hk` (Liang Wang), `mei.yang@unlv.edu` (Mei Yang), `a.k.singh@essex.ac.uk` (Amit Kumar Singh), `tmak@ecs.soton.ac.uk` (Terrence Mak)

scheme adopted in the chip.

Power budgeting is necessary in many-core chips [12, 13], as the total power budget typically available to a chip is not sufficient to allow all the cores to run at their peak performance simultaneously, and a core designated as the global manager then has to allocate power among the cores that request power to run their applications [14, 15, 16]. To make a fair and optimized power budgeting decision, the global manager needs to solicit budget requests from all the threads/applications running at different cores. Each power budget request is packetized and routed to the global manager through the chip's networked communication infrastructure, known as the network-on-chip (NoC).

Provided the hacker's agents can gain access to the global manager through a simple hardware Trojan embedded in an NoC router, the power budgeting system can be then attacked where power requests initiated by various cores are deliberately modified for harm. That is, power requests from the malicious applications (legitimate applications) will be increased (or decreased) to be higher (or lower) value than what were actually requested. Upon receiving the manipulated power budget requests, the global manager, irrespective of the power budgeting algorithms [17, 18, 19, 20] it runs, always allocates power budgets to favor the malicious applications, but penalize the legitimate applications. As a result of such stealth (a.k.a. false-data) attack against the power-budgeting scheme, a new many-core chip may experience significant performance degradation and even complete malfunctioning.

In [21], we identified the major contributors to the attack effects and subsequently adopted a linear model to quantify the attack effect.

Our previous work [21] has been significantly extended by making the following new contributions:

1) A new variable *application's likelihood to be tampered* is introduced to the attack effect regression model to improve the model's accuracy.

2) The attack effect regression model is modified to be a non-linear regression to reduce the regression error.

3) The attack is enhanced by a reinforcement learning algorithm, such that the activation status of each HT (ON or OFF) can be controlled at run time. In this manner, the countermeasure has a high probability of failing to detect the attack. The experimental results confirm that with the enhanced attack, the attack effect is much improved even when a countermeasure is there.

The rest of this paper is organized as follows. Section 2 introduces the background information and surveys the relevant previous studies. Section 3 provides a design of hardware Trojan that enables a DoS attack that hackers tamper and steal power budgets from other threads. Section 4 defines the related system parameters of the attack model. A defense method against the proposed DoS attack is discussed in Section 5, and Section 6 shows an anti-detection method with reinforcement learning. Section 7 reports the simulation results under different attack scenarios using the proposed attack model. Finally, Section 8 concludes the paper.

2. Background and Previous Studies

In this section, we will first review the power budgeting schemes that have been applied to many core systems. We then categorize various hardware Trojan enabled DoS attacks seen in many-core systems. Detection and countermeasures against hardware Trojans are also surveyed in this section.

2.1. Power Budgeting in Many-Core Systems

We assume a many-core system of interest follows a tiled architecture where each tile consists of a core, an L1 cache, an L2 cache bank, and a router. Each core can operate at any of the preset frequencies, and a higher frequency leads to higher performance at a cost of higher power consumption. As the total power budget in a many-core system is lower than the power needs for all the cores to run at their peak performance simultaneously, power budgeting is necessary with an aim to optimize the overall performance by choosing an appropriate frequency for each core. The decisions on frequency selection, or equivalently power allocation, are made by a special core designated as the global manager [14, 15, 16, 22, 23]. To make an optimized power budgeting decision, the global manager needs to solicit budget requests from all the threads/applications running at different cores. Power budgeting systems [17, 18, 19, 20, 24, 21] are the target or victim of the proposed hardware Trojan. Then the power allocation problem can be solved by heuristics [17, 18], control theory [19] or dynamic programming [20, 24], etc.

2.2. DoS Attack

DoS attacks on a many-core chip can target different components of the chip, including the memory system [25], and the network-on-chip (NoC) [7, 26, 27]. When a DoS attack is launched, the malicious agents can either generate a large volume of traffic to saturate the target components (e.g., the memory controller as in [28], or a particular network node in [29]) or silently drop packets. Apparently, DoS attacks can cause many-core chips to malfunction, or completely fail.

Xiao et al. [30] studied the researches on HTs from the last decade and proposed an adversarial model taxonomy (e.g. untrusted 3PIP Trojan model, untrusted fab or fabless design house Trojan model, etc.) to classify various HTs and countermeasures. The offline pre-silicon HT detection methods (e.g. functional validation, code/structural analysis) are used to validate third-party IP cores. Compared to these offline HT detection methods, we propose an online detection method since HTs can bypass the offline detection schemes due to the reason that they can be conditionally-triggered. With the help of hardware Trojans (HTs) [8, 9, 10, 31, 32, 33, 34] DoS attacks can be classified as the following categories, 1) *flooding attack* [35, 36], where a large volume of useless packets floods a victim node and saturates it; 2) *packet drop attack*, where some packets are dropped or directed to the malicious nodes so that the victim node can never receive a single packet designated to it [36]; 3) *privilege escalation attack* [37], where an average user process is granted the privileges of a supervisor so that it can steal passwords; and 4) *routing loop attack* [4], where packets that pass the malicious node will be routed back to the source node, effectively blocking the source core from communicating with any other cores.

2.3. Detection and Countermeasures against Hardware Trojans

Various circuit level HT detection techniques have been proposed, based on logic testing and side channel analysis during the post-silicon test/validation process [2, 3, 38]. Logic testing approaches attempt to develop directed test patterns to activate Trojans, should they ever exist, and then propagate their effects to the output ports for analysis. However, these approaches are likely to fail to activate Trojans consisting of a large number of trigger inputs, or those that are triggered by sophisticated conditions. In the side-channel analysis approaches, a system parameter (e.g., supply current or path delay), which can be affected due to unintended design modifications, needs to be measured. However, the effectiveness of side-channel analysis is limited by large intrinsic device parameter variations in modern nanometer technologies.

In the literatures, a number of countermeasures against HTs were proposed. In [6], a method concerning how to detour the malicious nodes was suggested. Another countermeasure to HT was through obfuscating the network states [35, 39]. In [40, 41], the countermeasure involved partitioning the network in a time-multiplexed manner.

3. HT-Enabled DoS Attacks Targeting the Power Budgeting System

In this section, we show that a DoS attack can be launched targeting the power budgeting scheme of a many-core chip. This new type of attack is made possible by implanting a hardware Trojan into the NoC's router. This section is thus dedicated to present a detailed design of such hardware Trojan.

3.1. Threat Model

The target system consists of trusted IP cores including processors, memories, network interfaces, and an untrusted third-party designed NoC IP core. The trusted processors communicate through the untrusted networks-on-chip. In this paper, the NoC topology is assumed to be 2D mesh with XY or adaptive routing algorithms (e.g., west-first routing, etc.)

The malicious IP vendors can implant various HTs in the NoC and run the hacker program on a processor to turn ON or OFF those implanted HTs. In the baseline attack, the HTs are always-on in the runtime, that is, the HTs are always active once the chip is powered on. In the enhanced attack, the HTs are activated by configuration packets, sent from the hacker program running on a compromised processor. The configuration packets have some special data fields in the packet payload. When they pass the routers infected by HTs, the HTs are activated by them. In this manner, the hacker program can control the ON/OFF status of the HTs.

3.2. Packet Frames

Generally speaking, a data packet arriving at a network node or a router has 4 fields: the source address (16 bits in this study), the destination address (16 bits), packet type (32 bits) and payload field (32 bits). Additional information, if needed, can be included in an optional field that is named so.

As shown in Fig. 1(a), a packet will be recognized as a power request packet if its packet type field is POWER_REQ. In this case, the payload of the packet is the power request value.

A packet will be recognized as a Trojan configuration packet if its packet type is CONFIG_CMD as shown in Fig. 1(b). For a configuration packet, its source address is actually the attacker's ID and the packet type field includes the CONFIG_CMD, global manager and activation signal. The attacker is a malicious program running on a compromised core. A configuration packet is meant to be sent to an implanted HT core by the attacker to set up the HTs.

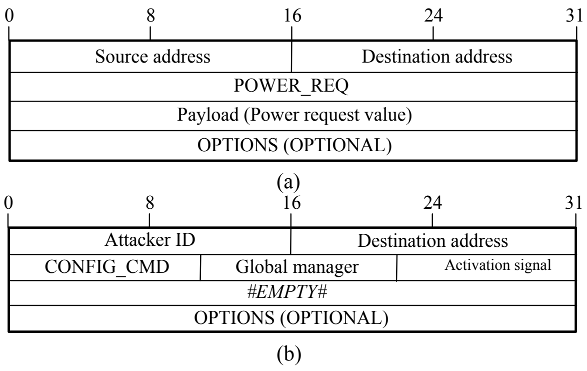


Figure 1: (a) Normal power request packet. (b) Attacker's configuration packet.

3.3. HT Attack Process

Fig. 2(a) shows a normal case where core B sends a power request of 1.2W to the global manager, core A, and this core decides to let core B have the requested amount

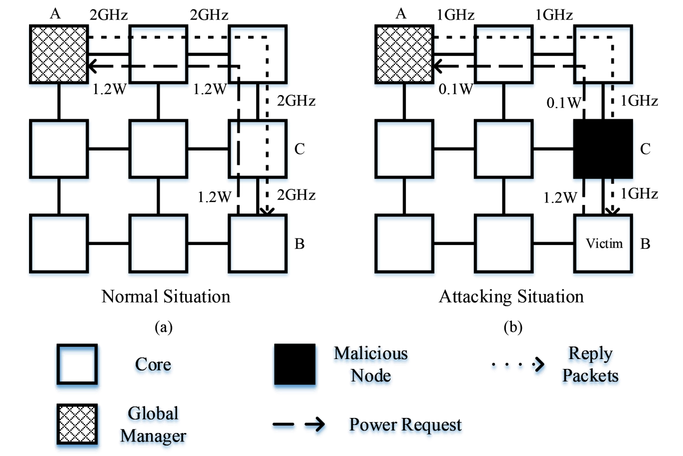


Figure 2: An example of the DoS attack targeting power budgeting system. (a) A many-core chip without HTs. (b) An attack scenario where an HT is implanted into the router of node C.

of power, and thus the global manager sets core B to operate at 2GHz. But if an HT implanted in node C gets activated, as shown in Fig. 2(b), node B's power request changes from 1.2W to 0.1W. As a result, the global manager wrongfully sets core B's frequency to 1GHz based on the power budget algorithm and the modified power budget it received. With a low power budget, core B is victimized as its performance will certainly degrade.

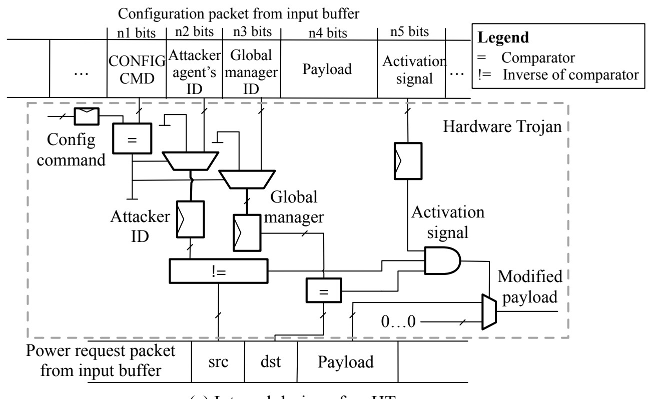
Before malicious nodes tampered the power requests, the attacker needs to locate the node which is implanted HT so that he can send the configuration packets to those HTs. When the cores communicated through MPI, the attacker program can parse the cache address to get specific node's id [42]. With knowledge of the node ID, topology and routing, the attacker can locate where the HTs are. Then the attacker program can be sneak in as a customer that has the privilege to run on a many-core server compromised by HTs. After finishing those preparations, the attacker can send configuration packets by MPI calls which specifying the target node or accessing a special memory address which translating the address to a cache bank ID, corresponding to the target node's ID [42]. The attacker program can also multicast (making multiple MPI calls or accessing memory addresses corresponding to multiple cache banks) configuration packets to set up the HTs.

When an attacker is about to attack, the configuration packet that includes the global manager's ID, its own core ID and the activation signal is sent to the malicious nodes. When the configuration packet arrives at the infected nodes, the HT stores the global manager's ID and the attacker's ID in its local registers, if it has not done so. The HT's activation state is set by the activation signal in the configuration packet. The hacker can launch different attack modes by setting the right activation signal, and the configuration packets may be sent out periodically for the control of attack strategies. For example, if the attacker agents want the HTs to be active in specific cycle time, a series of configuration packets can be sent with activation signals alternated to be ON and OFF.

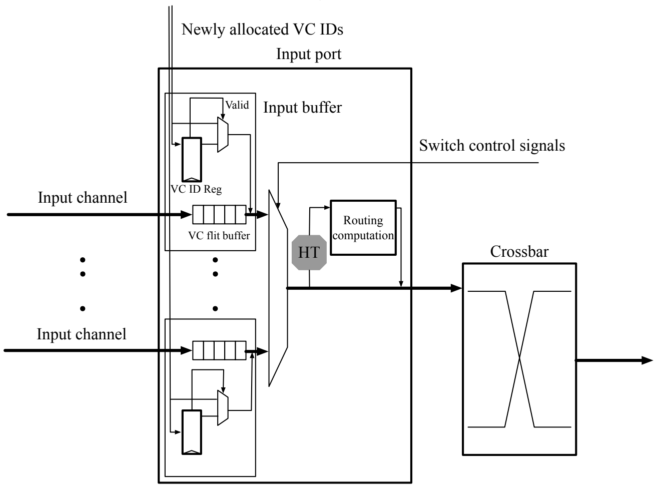
After configuration, when a victim's data packet passes through those routers with implanted HT, if the HT is not activated, the packet is forwarded normally (i.e., no modification to the packet will ever be made). Otherwise, the HT checks whether the data packet's destination is the global manager and the source is not hacker's agent. If so, the triggering module triggers the functional module to modify the value of the power request field.

3.4. HT Circuits Implementation

To launch a DoS attack that is enabled by stealing power budget from victim threads, an HT circuit includes some registers to store the configuration parameters from the hackers' agents. The triggering module scans data packets, and if a packet is found to match the configuration parameters stored in the HT's local register, the functional module is then enabled if HT is activated. The functional module basically manipulates the payload of the packet that the power request is changed to a smaller value. As Fig. 3 shows, an HT has 3 comparators and 2 registers that sit between the router's input buffer and the routing computation module.



(a) Internal design of an HT.



(b) The location of the HT inside a router.

Figure 3: Hardware Trojan Design

4. Evaluating the DoS Attack Effect

A DoS attack targeting the power budgeting scheme as described in Section 3 tends to cause victim applications/threads to suffer from serious performance degradation while the attacker's malicious application may see a considerable performance gain. The effectiveness of the proposed DoS attack can be attributed to a number of factors listed below:

- The number of hardware Trojans in the NoC.
- Locations of the hardware Trojans and the global manager. When an HT is close to the global manager, since more budget request packets are likely to pass the HT before reaching the global manager, this HT can make more packet modifications, and thus can have higher impacts on the overall system performance.
- Application's sensitivity to its power budget. Performance impacts due to power budget changes can be application-specific. For example, the performance of an instruction-bounded application is typically hit harder than that of memory-bounded applications.
- Application's likelihood to be tampered. We maintain an assumption that the HTs are evenly distributed. For a power request packet with longer length from source to the global manager has a higher probability of being tampered. Different applications' locations on the chip lead to different probabilities to be attacked.

Table 1 shows the mathematical notations used in the DoS attack.

4.1. Measures of Attack Effect

Definition 1. *Application k's performance θ_k is defined by*

$$\theta_k = \sum_{j \in C_k} IPC(j, k, f_j) \cdot f_j \tag{1}$$

where θ_k is application k's instruction per clock (IPC) value, C_k is the set of cores running application k's threads, and $IPC(j, k, f_j)$ is core j's IPC.

Definition 2. *Application k's performance change Θ_k is defined as*

$$\Theta_k = \frac{\theta_k}{\Lambda_k} \tag{2}$$

where θ_k is the application k's IPC value with HTs and Λ_k is the application k's IPC without HTs.

Table 1: Notations in the DoS Attack

$IPC(i, z, f)$	The core i's IPC value when running application z and it is frequency is f.
C_k	The set of cores running application k's threads.
θ_k	The application k's IPC value.
Λ_k	The application k's IPC value without HTs in the chip.
Θ_k	The application k's change in performance.
V	The number of victims in the many-core chip.
A	The number of attackers in the many-core chip.
$\phi(j, z)$	The core j's sensitivity to power budget when it is running application z.
Φ_k	The application k's power budget sensitivity.
Z_k	The application k's likelihood to be tampered.
ζ_j	The Manhattan distance from core j to the global manager.
m	The number of malicious nodes.
(ω_X, ω_Y)	The HTs' virtual center.
ρ	The Manhattan distance from the global manager to the HTs' virtual center.
η	The HT's distribution density.
M_{HT}	The maximum number of HTs on a chip.

Definition 3. *The attack effect of the proposed DoS attack $Q(\Delta, \Gamma)$ is defined as*

$$Q(\Delta, \Gamma) = \frac{V \cdot \sum_{a \in \Delta} \Theta_a}{A \cdot \sum_{v \in \Gamma} \Theta_v} \tag{3}$$

where V and A are the numbers of victims and attackers respectively. Δ and Γ are the sets of attacker applications and victim applications respectively.

If the attacker's performance improves, or victim's performance degrades, $Q(\Delta, \Gamma)$ will show a higher value. In a simple word, the larger $Q(\Delta, \Gamma)$ value has, the stronger an attack is.

4.2. Factors Impacting Attack Effect

There are a number of factors that can impact the attack effects.

1. Application's sensitivity to power budget: Applications have different performance changes when their V/F values vary.

   Definition 4. *Core j's sensitivity to power budget while running application z, denoted as $\phi(j, z)$ is de-*

fined by

$$\phi(j,z) = \sum_{i=1}^{s-1} \left| \frac{IPC(j,z,\tau_i) - IPC(j,z,\tau_{i+1})}{\tau_i - \tau_{i+1}} \right| \quad (4)$$
$$\tau_1 < \tau_2 < \cdots < \tau_s$$

where τ_i and τ_{i+1} are the available frequency levels, $IPC(j,z,\tau_i)$ is core j's IPC when it is running application z's thread with a frequency level of τ_i.

Definition 5. *Application k's sensitivity to the power budget is defined by*

$$\Phi_k = \frac{\sum_{i \in C_k} \phi(i,k)}{|C_k|} \quad (5)$$

where C_k is the set of cores running application k's threads.

2. Application's likelihood to be tampered: under the assumption that HTs are evenly distributed over the chip, a packet with a longer path length from the source to the destination implies a higher probability of being tampered. The likelihood of an application's power request being tampered is defined as the average length of the packets' paths that are sent from the cores running the application to the global manager.

Definition 6. *Denote ζ_j as the Manhattan distance from core j to the global manager. Application k's likelihood to be tampered Z_k is defined by*

$$Z_k = \frac{\sum_{j \in C_k} \zeta_j}{|C_k|} \quad (6)$$

3. Parameters characterizing system architecture: The distribution of HTs has performance implications on both attackers and victims.

Definition 7. *Assume there are a total of m malicious nodes. The coordinates of the m HTs' virtual center is defined by*

$$\omega_X = \frac{\sum_{i=1}^{m} X_{M_i}}{m} \quad (7)$$

$$\omega_Y = \frac{\sum_{i=1}^{m} Y_{M_i}}{m} \quad (8)$$

where X_{M_i} and Y_{M_i} are x and y coordinates of the malicious nodes' respectively.

Definition 8. *Distance between the global manager and the virtual center of those HTs denoted as ρ is defined by*

$$\rho = \mathbf{MD}(O, \Omega) \quad (9)$$

where O is the global manager's location, and Ω is the HTs' virtual center.

If the virtual center of HTs is far away from the global manager, measured as a long distance between the two, fewer packets with power requests actually pass through the nodes infected with HTs. In other words, the performance of the system has a lower probability to be affected.

Definition 9. *HTs' density denoted as η is the average Manhattan distance between HTs' virtual center and each malicious nodes. It measures the variance of the HT distribution, which is defined as*

$$\eta = \frac{\sum_{i=1}^{m} \mathbf{MD}(\Omega, M_i)}{m} \quad (10)$$

where m is the number of malicious nodes.

A higher density indicates a large number of malicious nodes are around the virtual center, and more packets may be intercepted and modified, leading to a higher infection rate.

4.3. Modeling Attack Effects

We use three regression models, the linear, the quadratic and the cubic models, to quantify the relationship between the aforementioned parameters. The dependent variable is the Q value ($Q(\Delta,\Gamma)$). The independent variables include distance (ρ), density (η), the number of HTs (m), all applications' sensitivity to power budget (Φ_k) and all applications' likelihood to be tampered (Z_k).

And in Section 7, the quadratic model shows the lowest prediction error, the model is as follows.

$$Q(\Delta,\Gamma) = \alpha_1 \times \rho + \beta_1 \times \eta + \sigma_1 \times m + \sum_{s=1}^{V} a_{s,1} \times \Phi_{\gamma_s} + \sum_{t=1}^{A} b_{t,1} \times \Phi_{\delta_t} + \sum_{s=1}^{V} c_{s,1} \times Z_{\gamma_s} + g_1 \quad (11)$$

$$Q(\Delta,\Gamma) = \sum_{p=1}^{2} \alpha_p \times \rho^p + \sum_{p=1}^{2} \beta_p \times \eta^p + \sum_{p=1}^{2} \sigma_p \times m^p + \sum_{p=1}^{2}\sum_{s=1}^{V} a_{s,p} \times \Phi_{\gamma_s}^p + \sum_{p=1}^{2}\sum_{t=1}^{A} b_{t,p} \times \Phi_{\delta_t}^p + \sum_{p=1}^{2}\sum_{s=1}^{V} c_{s,p} \times Z_{\gamma_s}^p + g_2 \quad (12)$$

$$Q(\Delta,\Gamma)=\sum_{p=1}^{3}\alpha_p\times\rho^p+\sum_{p=1}^{3}\beta_p\times\eta^p+\sum_{p=1}^{3}\sigma_p\times m^p$$
$$+\sum_{p=1}^{3}\sum_{s=1}^{V}a_{s,p}\times\Phi^p_{\gamma_s}+\sum_{p=1}^{3}\sum_{t=1}^{A}b_{t,p}\times\Phi^p_{\delta_t}$$
$$+\sum_{p=1}^{3}\sum_{s=1}^{V}c_{s,p}\times Z^p_{\gamma_s}+g_3 \qquad (13)$$

where $\alpha_p, \beta_p, \sigma_p, a_{s,p}, b_{t,p}, c_{s,p}, 1 \leqslant s \leqslant V, 1 \leqslant t \leqslant A, g_i, i = 1, 2, 3$ are the regression coefficients, γ_s and δ_t are sth and tth victim/attacker application, ρ is the HT's distance between HTs' virtual center and the global manager, η is the HT's distribution density, m is the number of malicious nodes, Φ_{γ_s} and Φ_{δ_t} are the victim and the attacker applications' sensitivities to power budget, respectively. Z_{γ_s} is likelihood of being tampered for application γ_s. The Eqns. 11 12 13 is linear, quadratic, and cubic regression models respectively.

Following the attack effect model, we formulate a problem to maximize the attack effect by selecting the proper distance and density of HTs. The constraints are the area of the HT circuits, or equivalently, the number of HTs that will be selected by the attacker. This attack effect optimization problem can be formulated as follows.

$$\max_{\rho,\eta,m} \quad Q(\Delta,\Gamma) \qquad (14)$$

$$\text{subject to} \quad m \leqslant M_{HT} \qquad (15)$$

where M_{HT} is the maximum number of malicious nodes selected by the hackers.

The attacker solves the above problem to maximize the attack effect before loading the application into the chip. To solve the problem, one can exhaustively enumerate all possible values for above mentioned three metrics: 1) number of HTs, 2) distance between the global manager and the virtual center of HTs, 3) HTs distribution density.

5. Possible Detection and Countermeasure Against the Proposed Attack

A possible detection method works as follows.

1) The global manager collects each core's power budget request and computes the average value.

2) The difference between the average value and a core's power request is defined as the deviation of this core's power request. The global manager calculates the deviations from all cores' power requests and notes the cores into an observation list whose deviations are larger than a threshold $\lambda\times$ average value.

3) When a core appears in the observation list consecutively n times, the global manager marks this core is under attacked (i.e., victims).

4) A core is marked not being attacked if the deviation of this core's power request is no longer larger than the threshold.

To determine the values of λ and n, a set of experiments are performed. In Fig. 4(a), the value of n varies from 1 to 8 with the threshold $\lambda = 0.4$. The results show that when $n = 4$, the attack effect is minimized. In Fig. 4(b), the threshold λ varies from 0.05 to 0.95 with $n = 4$. The results show that when $\lambda = 0.4$, the attack effect is minimized. Therefore, λ is set to be 0.4 and n is set to be 4.

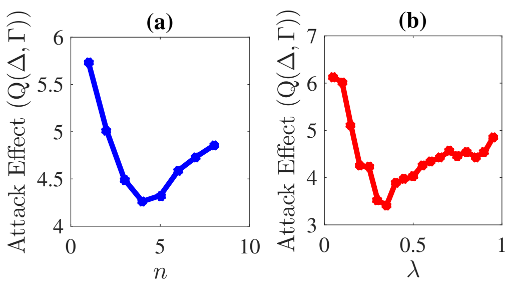


Figure 4: The attack effect over different n and λ values. (a) n ranges from 1 to 8 with $\lambda = 0.4$. (b) λ ranges from 0.05 to 0.95 with $n = 4$.

When the global manager marks the victim cores, it defends as follows.

1) The global manager computes the average value of the power requests from cores that are not being attacked.

2) Before running the power budgeting algorithm, the global manager resets those victims' power budget requests to the average value in step 1).

3) The global manager runs the power budgeting algorithm with the new power requests.

Fig. 5 shows an example of the detection and countermeasure against the proposed attack. If the power requests of four cores A, B, C and D are 0.1W, 1.2W, 1.3W, 1.4W in consecutive 4 times. The average power request is 1.0W. The deviations of all cores' power requests are 0.9W, -0.2W, -0.3W, -0.4W. The deviation of core A (0.9W) is larger than the threshold (0.4 × 1.0W = 0.4W), then the global manager marked core A is under attacked. Core A's power request is recalculated as the average power requests of the other three cores. Then the power budgeting algorithm is applied with 1.3W (core A's new power request), 1.2W (core B), 1.3W (core C), 1.4W (core D).

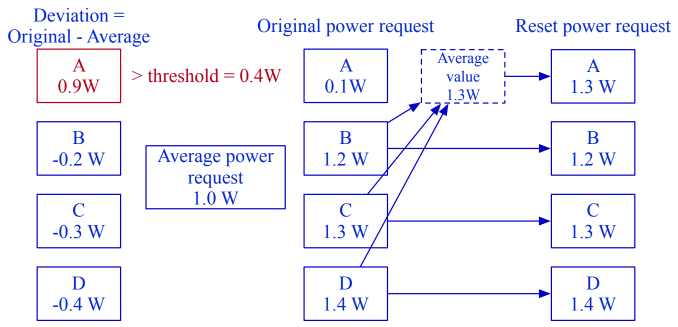


Figure 5: An example of the detection and countermeasure against the proposed attack.

6. The Enhanced Attack

To avoid being detected by the method in Section 5, the HTs need to hibernate and attack intermittently, by controlling the ON/OFF status of the HTs. There are two ways of controlling the ON/OFF status of an HT.

1) The HTs are turned ON or OFF randomly so that the detection scheme fails to locate the HTs.

2) The HTs are activated by configuration packets sent by the hacker program running on a compromised core.

For the first type of HTs (referred as randomly-activated HTs), each HT in the routers flips its ON/OFF status after a random time period. The HT uses a mod-k counter such that when the counter output equals to 0, the HT is OFF and when the counter is 1 otherwise. When the victim packets are passing through the HT-infected routers, they might be tampered, depending on the ON/OFF status of the HTs. As a result, the detector fails to locate HTs.

For the second type of HTs (referred as hacker-controlled HTs), as introduced in Section 3.3, the attacker can use activation signals to turn each hardware Trojan ON or OFF by a Q-learning algorithm [43]. The hacker program running on a compromised core configures the HTs by searching the Q-table to select an action which is to turn ON or OFF the HTs. In the next iteration, the attacker records its performance improvement as the reward and updates the Q-table. The advantage of this type of attack is that the attack effect can be optimized. It is different from a worm or a software malware in that, 1) the worm or malware can be detected and neutralized by anti-virus software while the HTs cannot be detected by it, and 2) HTs can tamper the data packets directly, which cannot be directly accessed by software level operations. The notations are listed in Table 2.

In what follows, the state space and actions are first defined, followed by the Q-learning based algorithm which runs by the hacker program.

Table 2: Notations in the enhanced attack

$Q(s,a)$	Q-table.
ϵ	The greedy rate in Q-learning.
α	The learning rate in Q-learning.
γ	The discount rate in Q-learning.
$\mathbf{s_t}$	HTs' state at time t.
$\mathbf{a_t}$	The attacker's action at time t.
$\mathbf{r_t}$	The reward at time t, equal to the malicious application's performance change (Θ in the Definition 2) at time t.
$\oplus$	Exclusive or (XOR) operation.

6.1. States and Actions

Definition 10. *At time t, the attacker's state* $\mathbf{s_t} = (s_t^1, s_t^2, ..., s_t^m)$ *is defined as*

$$s_t^i = \begin{cases} 1, \text{if the } i\text{th HT is ON}, \\ 0, \text{if the } i\text{th HT is OFF}, \end{cases} \quad (16)$$
$$i = 1, 2, ..., m$$

For example, assume a chip has 3 HTs. The first one is ON while the others are OFF. The current state vector is $\mathbf{s_t} = (1, 0, 0)$.

Definition 11. *For the i-th HT, the flipping operation is defined as reversing the current state* ($s_{t+1}^i = 1 - s_t^i$). *At time t, the attacker's action* $\mathbf{a_t} = (a_t^1, a_t^2, ..., a_t^m)$ *is defined as*

$$a_t^i = \begin{cases} 1, \text{if the } i\text{th HT's state is flipped}, \\ 0, \text{if the } i\text{th HT's state is NOT flipped}, \end{cases} \quad (17)$$
$$i = 1, 2, ..., m$$

The action space is $\Pi = \{\tau_1, \tau_2, ..., \tau_{2^m}\}$, where $\mathbf{a_t} \in \Pi$. In the above example, the action space is $\Pi = \{(0,0,0), (0,0,1), (0,1,0), (0,1,1), (1,0,0), (1,0,1), (1,1,0), (1,1,1)\}$. If the action is $\mathbf{a_t} = (0,1,0)$, the 2nd HT's state is to be flipped. The next state $\mathbf{s_{t+1}}$ is $(1,1,0)$.

Based on the above settings, the next state equals to bit-wise exclusive ORing (XORing) the current state and the action, i.e., $\mathbf{s_{t+1}} = \mathbf{s_t} \oplus \mathbf{a_t}$.

6.2. Q-Learning Based Enhanced Attack with Improved Stealthiness

Definition 12. *The Q-table* $Q(\mathbf{s_t}, \mathbf{a_t})$ *is a 2-dimension table where the index of the row is* $\mathbf{s_t}$ *and the index of column is* $\mathbf{a_t}$.

Definition 13. *The attacker's reward* $\mathbf{r_t}$ *is the malicious application's performance change* (Θ *in Definition 2*) *at time t.*

The attacker is enhanced by a Q-learning algorithm to improve the stealthiness of the HTs given the HT detection method in Section 5. By doing so, the detection method fails to identify those victim cores. The reason is as follows. The detection method only marks those cores as being attacked that send n low power requests consecutively. However, the attacker switches the HTs ON and OFF to break the continuity of victims' low power requests. The pseudocode of the Q-learning based method is shown in Algorithm 1.

The attacker's initial state $\mathbf{s_0}$ is set to be (0, 0, ... ,0). In each iteration, the attacker selects the action $\mathbf{a_t}$ by the ϵ-greedy strategy as follows.

Definition 14. *At time t, the attacker selects an action randomly with a probability ϵ, or the action which maximizes the Q value given the current state.*

$$\mathbf{a_t} = \begin{cases} \tau_k, \tau_k \in \Pi, k \in \{1, 2, ..., 2^m\}, \text{if } \mathbf{random(0,1)} \leqslant \epsilon \\ \\ \arg\max_{a \in \Pi} Q(\mathbf{s_t}, a), \text{otherwise} \end{cases} \tag{18}$$

where the function $\mathbf{random(0,1)}$ generates a random number ranging from 0 to 1.

Next, the attacker transfers to the next state ($\mathbf{s_{t+1}}$), records the reward and updates the Q-table by [43]

$$\begin{aligned} Q(\mathbf{s_t}, \mathbf{a_t}) = (1-\alpha) \times Q(\mathbf{s_t}, \mathbf{a_t}) + \\ \alpha \times [\mathbf{r_t} + \gamma \times \max_{a \in \Pi} Q(\mathbf{s_{t+1}}, a)] \end{aligned} \tag{19}$$

The entry $Q(\mathbf{s_t}, \mathbf{a_t})$ is updated by a weighted average of two terms [43]. The two terms are: 1) the entry's previous value $Q(\mathbf{s_t}, \mathbf{a_t})$, 2) future expected return after discounting plus the reward $\mathbf{r_t}$. The future expected return after discounting is the product of discount rate γ and the maximum value under the next state in Q-table $\max_{a \in \Pi} Q(\mathbf{s_{t+1}}, a)$. These steps iterate until reaching the preset maximal iteration number (i.e. T in Algorithm 1). When all the HTs are at the ON state at the time when power is turned on, the detector shall be able to detect some hardware Trojans, and as a result, the infection rate gradually decreases. When all the HTs are at the OFF state at the time when power is turned on, the detector will not be able to detect all the HTs. As a result, the infection rate gradually increases. When the greedy probability ϵ is small, the attacker tends to choose an action that tends to maximize the Q value in the current state. When the greedy probability ϵ is large, the attacker tends to choose its action in a random fashion. A larger greedy probability leads to better exploration in the reinforcement learning, which increases the probability of HTs being detected, and thus leading to a lower infection rate.

The Q-learning based hacker program has a lower overhead on the overall system. The experimental results show the enhanced attack program accounts for about 2%-4% running time compared to a normal program running on the compromised core.

Algorithm 1: The Enhanced Attack

```
Input: $\epsilon$: The probability that the attacker chooses
       an action randomly.
   $\alpha$: The learning rate in Q-learning.
   $\gamma$: The discount rate in Q-learning.
   $T$: The maximal iteration number.
   $\mathbf{s_0}$: The initial state of the attacker in Q-learning.
1  $t = 0$ ;
2  while $t < T$ do
       /* The greedy strategy in Q-learning.
          */
3      if random(0,1) $\leqslant \epsilon$ then
4          Choose $\mathbf{a_t}$ randomly;
5      end
6      else
7          $\mathbf{a_t} = \arg\max_{a \in \Pi} Q(\mathbf{s_t}, a)$;
8      end
9      Execute action $\mathbf{a_t}$, $\mathbf{s_{t+1}} = \mathbf{s_t} \oplus \mathbf{a_t}$ ;
10     Calculate reward $\mathbf{r_t}$ ;
       /* Updating the Q-table.                    */
11     $Q(\mathbf{s_t}, \mathbf{a_t}) = (1-\alpha) \times Q(\mathbf{s_t}, \mathbf{a_t}) + \alpha \times [\mathbf{r_t} + \gamma \times$
        $\max_{a \in \Pi} Q(\mathbf{s_{t+1}}, a)]$ ;
12     $t = t + 1$ ;
13 end
```

7. Experimental Results

7.1. Experimental Setup

Experiments are performed to evaluate the attack effect with the number of HTs and HTs' distribution. The experiments are using an event-driven many-core simulator in C++ [44]. 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 node of the NoC. Table 3 lists the simulator configuration. Multi-threaded applications can have their threads running on different cores. Communications between threads take place through the NoC. The messages in the network are generated by memory transactions including read/write access requirements. Table 4 lists the benchmarks used for performance evaluation, they are selected from PARSEC and SPLASH-2. In the following experiments, we select a 16 × 16 2D mesh as the underline NoC architecture.

7.2. Evaluating the Infection Rate

Fig. 6 (a) compares the infection rates when the global manager is at different locations and the system size is 64. One can see that along with the increase of the number of HTs, the infection rate increases as well. If the global manager is placed at the corner of the chip, the corresponding infection rate is more than 20% higher than that of the case where the global manager is placed at the center, assuming there are more than 10 HTs. When the system

Table 3: Configuration used in the simulation

Number of processors	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 protocol	64 KB slice/node, 64B line six cycles, two ports
Main memory size	2 GB, latency 200 cycles
On-chip network parameters	
NoC flit size	72-bit
Data packet size	5 flits
Meta packet size	1 flit
NoC latency	router two cycles, link one cycle
NoC Virtual Channel (VC) number	4
NoC buffer	5 × 5 flits
Routing algorithm	XY Routing or Adaptive Routing

Table 4: Benchmarks used in the simulation

PARSEC	streamcluster, swaptions, ferret, uidanimate, blackscholes, freqmine, dedup, canneal, vips
SPLASH-2	barnes, raytrace

size is 512, a similar trend is observed as shown in Fig. 6 (b). The reason is that a packet loaded with a power budget request has to travel a long distance to reach the global manager that is at the corner of the chip. Longer routing distance increases the chance that such a packet is intercepted and modified by an HT node.

Fig. 7 compares the infection rates of 3 different cases: (i) HTs are all placed close to the center of the chip, (ii) HTs are randomly distributed, and (iii) HTs are placed to a concentrated area near one corner. Here, the global manager is assumed to be at the center of the chip.

From Fig. 7, one can see that the infection rates of the cases that the HTs are near the center location of the chip are higher than those of the other two cases. For example, in Fig. 7 (a), when the system size is 256, the infection rate of the case that the HTs are close to the center is 1.59× and 9.85× more than those of the other two cases, respectively. The reason is that, in the case HTs are around the center, more packets with power requests are likely to be intercepted and modified. In the case when HTs are randomly distributed, fewer packets with power requests will be attacked. In the case when HTs are clustered around one corner of the chip, some packets with power requests will never be attacked.

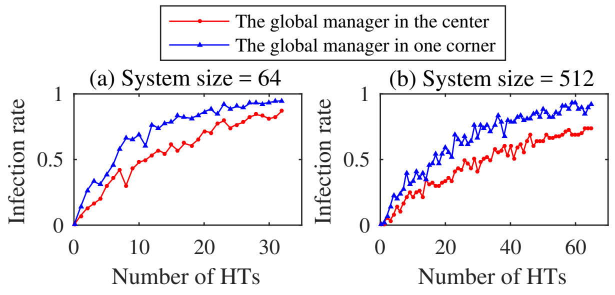


Figure 6: Infection rate comparison with different HT numbers when the system size is (a) 64, and (b) 512.

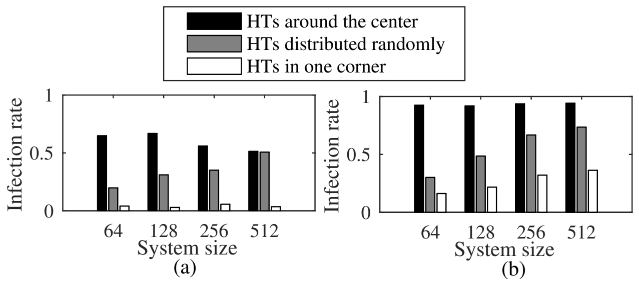


Figure 7: Infection rate comparison with different HT distributions, when the number of HTs is (a) $\frac{1}{16}$, (b) $\frac{1}{8}$ of the system size.

7.3. Evaluating Attack Effect

We use the mixes of a few benchmarks to test the attack effects of the proposed DoS attack. The numbers of attackers and victims are set to be 1, 2, and 3, and they can be mixed to obtain four combinations, as tabulated in Table 5.

Table 5: Benchmark combinations used in the experiments

Combination	**Attackers**	**Victims**
Mix-1	barnes, canneal	blackscholes, raytrace
Mix-2	freqmine, swaptions	raytrace, vips
Mix-3	canneal	barnes, vips, dedup
Mix-4	barnes, streamcluster, freqmine	raytrace

Fig. 8 shows each mix's Q value with respect to infection rate. Each application is set to have 64 threads and run on a chip with 256 cores. Overall speaking, a higher infection rate leads to a larger Q value. In the case of Mix-4 which has three attackers, the Q value reaches its peak (i.e., 6.89) at the infection rate of 0.9. In Fig. 9 (a), one can see that when the infection rate is 0.5, the performance of the attackers is improved by as much as 1.2×, and the victim's performance drops by 0.6×. In Fig. 9 (c), when there are 3 victims, and the infection rate is 0.5, the performance improvement of the attacker is as much as 1.35×. In Fig. 9 (d), when there are 3 attackers, and the infection rate remains at 0.5, the victims' performance degrades by as much as 0.8×.

Next, the attack effect of an NoC system with HTs optimally placed (select number of HTs, distance between the global manager and virtual center of HTs, HTs distribution density which solves Eqns. 14-15) is compared with that of a system with randomly placed HTs. In the case that there are 16 HTs in the chip and the global manager is in the center of the chip, the attack effect of the NoC with optimal HT distribution is about 30% higher than that of NoC but with a random HT distribution for the mixes of 1, 2 and 3. More significant improvement (by as much as 110%) in attack effect is seen in the case of mix-4. In a simple word, solving the attack effect maximization problem in Eqns. 14-15 can indeed improve the attack effects substantially.

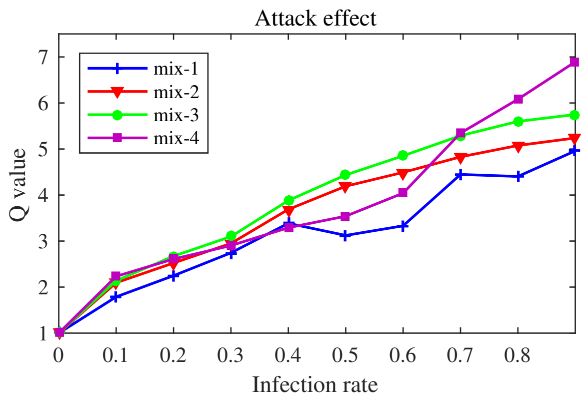


Figure 8: Attack effect comparison with different infection rates.

7.4. HT's Area and Power

Each HT has an area of $12.1716\mu m^2$ and consumes $0.55018\mu W$ power, as reported by Synopsys Design Compiler under 45nm TSMC library. As a comparison, a router with 4 virtual channels and 5-flit depth First-Input-First-Output has a total area of 71814 μm^2 and consumes a total power of $31881\mu W$, obtained from DSENT. In a simple term, an HT's area and power is about 0.017% and 0.0017% of a single router, the delay for a packet passing through a router with HT's circuits is $0.05ns$ more than passing a router without HT, making such an HT hard to be detected.

7.5. Comparison of Different Attack Effect Models

As discussed in Section 4, a new variable that quantifies the likelihood that power request may be tampered (Z_k) is added in the attack effect model. To validate the new variable, we used mixes in Table 5 to compute the average regression error under different system size. Fig. 10 and Fig. 11 shows the regression error after including applications' likelihoods to be tampered (Z_k) in the linear and quadratic models respectively. In Fig. 10, as the size of the system grows, the linear model's regression error decreases from about 8%-9% to 6% before Z_k is added in the model and decreases from about 7% to 5% after Z_k is added in the model. In the Fig. 11, as the size of the system grows, the quadratic model's regression error decreases from about 7% to 6% before Z_k is added in the model and decreases from about 6% to 4% after Z_k is added in the model. After including Z_k both of the two models' regression errors decrease by 1-2%.

Next, the attack effect models (Eqn. 11, 12, 13) with different polynomial orders are compared. Fig. 12 shows the regression error with three different polynomial orders, i.e., linear, quadratic and cubic. As the size of the system grows, the regression errors of all three models decreased from about 7%-9% to 5%. From Fig. 12, the quadratic model has a minimal error and is thus used in the paper.

7.6. Evaluating the Enhanced Attack

With mix-1 in Table 5, Fig. 13 shows the infection rate in four scenarios. 1) The infection rate is about 0.72 when the many-core chip is under the proposed baseline attack in Section 3 (without enhancement). 2) The infection rate decreases to about 0.14 when the global manager uses the countermeasure in Section 5 to monitor all cores' power requests, which is about 80.56% lower compared to the case without the countermeasure. 3) The infection rate increases to about 0.38 when the randomly-activated HTs are deployed against the countermeasure in Section 5, which is about 171.43% higher compared to the case without the enhancement. 4) The infection rate increases to about 0.56 when the hacker-controlled HTs are applied, which is about 298.34% higher compared to the case without the enhancement and 47.37% higher compared to the case with the randomly-activated HTs. The proposed countermeasure can decrease the infection rate compared to the baseline attack. Comparing to the baseline attack, both of the proposed enhanced attacks increase the infection rate significantly even the countermeasure is there. The hacker-controlled HTs have more infection rate improvement than the randomly-activated HTs.

8. Conclusion

In this paper, we proposed a new type of Hardware-Trojan-assisted DoS attack that can severely penalize the performance of a many-core chip by tampering the power budget requests within the chip. Specifically, when a power request packet from a victim application/thread traverses through a malicious router infected with an HT, the power request is modified. As a result, this victim application

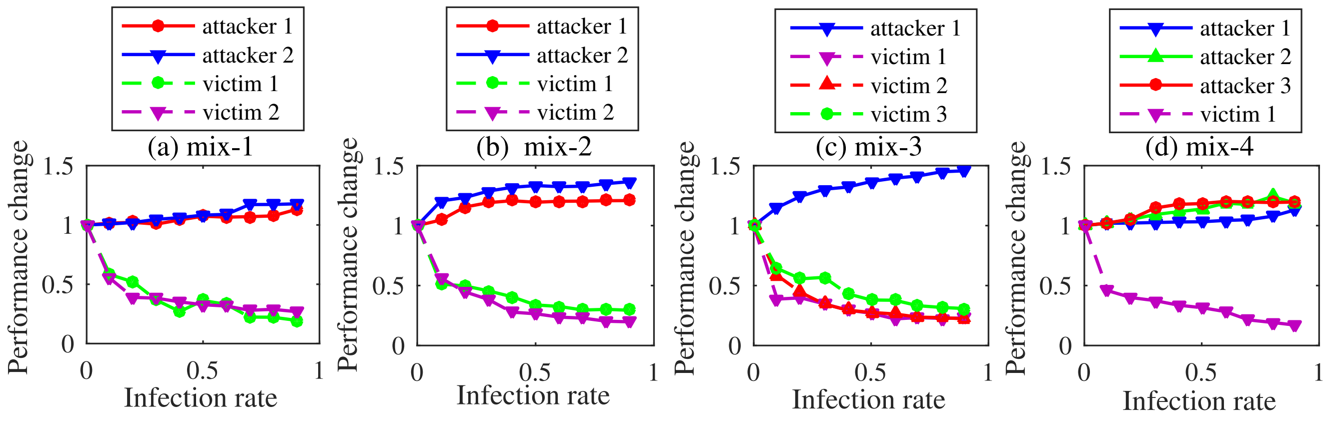


Figure 9: Applications' performance changes for each mix.

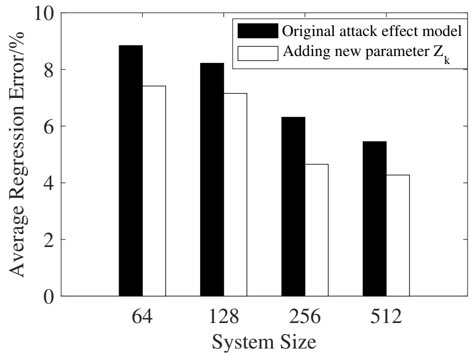


Figure 10: The regression error comparison over different network sizes, when the attack effect model is a linear regression model (Eqn. 11).

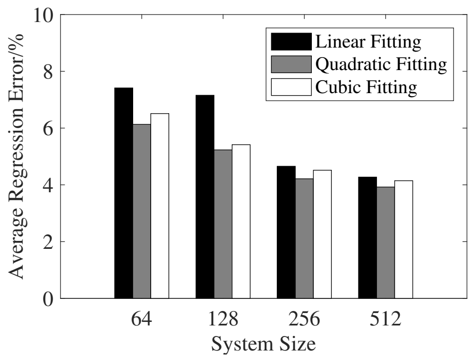


Figure 12: The regression error comparison over different network sizes with different models (Eqn. 11, 12, 13).

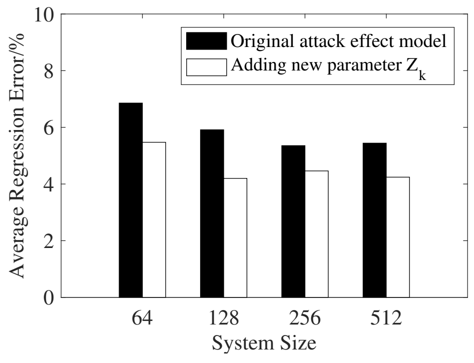


Figure 11: The regression error comparison over different network sizes, when the attack effect model is a quadratic regression model (Eqn. 12).

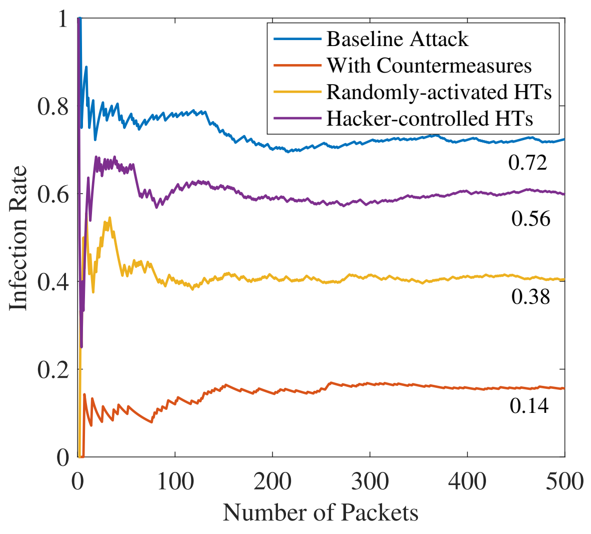


Figure 13: The infection rate comparison of 1) the baseline attack (without any countermeasures) in Section 3, 2) system with countermeasures in Section 5, 3) system under the randomly-attacked HTs and 4) system under the hacker-controlled HTs in Section 6

.

suffers from poorer performance due to its granted power budget is lower than its power needs to sustain its performance. In contrast, malicious applications/threads can receive excessive power budgets so that they experience performance boost at the cost of performance degradation of legitimate applications/threads. Even worse, we experimentally demonstrated that with a randomly-activated HT method and a Q-learning based attack method as described in this paper, all the known countermeasures known today would fail when combating this new type of DoS attack. As so, rigorous research on developing detection and protection methods against this new DoS attack is warranted.

References

[1] Y. L. Gwon, H. T. Kung, D. Vlah, Distroy: detecting integrated circuit trojans with compressive measurements, in: Proc. USENIX Conf. Hot Topics in Security, HotSec'11, 2011, pp. 3–9.

[2] M. Beaumont, B. Hopkins, T. Newby, Hardware Trojans-prevention, detection, countermeasures (a literature review), Tech. rep., DTIC Document (2011).

[3] K. Chrysanthou, P. Englezakis, A. Prodromou, A. Panteli, C. Nicopoulos, Y. Sazeides, G. Dimitrakopoulos, An online and real-time fault detection and localization mechanism for network-on-chip architectures, ACM Trans. Architecture and Code Optimization 13 (2) (2016) 22:1–22:26.

[4] A. Kulkarni, Y. Pino, T. Mohsenin, Svm-based real-time hardware trojan detection for many-core platform, in: Int'l Symp. Quality Electronic Design (ISQED), 2016, pp. 362–367.

[5] A. Kulkarni, Y. Pino, M. French, T. Mohsenin, Real-time anomaly detection framework for many-core router through machine-learning techniques, ACM J. Emerging Technologies in Computing Systems (JETC) 13 (1) (2016) 10:1–10:22.

[6] R. JS, D. M. Ancajas, K. Chakraborty, S. Roy, Runtime detection of a bandwidth denial attack from a rogue network-on-chip, in: Proc. Int'l Symp. Networks-on-Chip, 2015, pp. 8–16.

[7] J. Frey, Q. Yu, A hardened network-on-chip design using runtime hardware trojan mitigation methods, Integration, the VLSI Journal 56 (2017) 15–31.

[8] H. Li, Q. Liu, J. Zhang, A survey of hardware trojan threat and defense, Integration, the VLSI Journal 55 (2016) 426–437.

[9] S. K. Haider, C. Jin, M. van Dijk, Advancing the state-of-the-art in hardware trojans design, arXiv preprint arXiv:1605.08413.

[10] W. Burleson, O. Mutlu, M. Tiwari, Invited-who is the major threat to tomorrows security?: You, the hardware designer, in: Proc. Design Automation Conf., 2016, pp. 1–5.

[11] J. Dofe, Q. Yu, H. Wang, E. Salman, Hardware security threats and potential countermeasures in emerging 3d ics, in: Proc. Int'l Symp. VLSI, 2016, pp. 69–74.

[12] S. Pagani, J.-J. Chen, M. Shafique, J. Henkel, Thermal-aware power budgeting for dark silicon chips, in: Proc. Int'l Conf. Green Computing and Sustainable Computing Conf. (IGSC), 2015, pp. 1–6.

[13] A. Rezaei, D. Zhao, M. Daneshtalab, H. Wu, Shift sprinting: fine-grained temperature-aware noc-based mcsoc architecture in dark silicon age, in: Proc. Design Automation Conf., 2016, pp. 155:1–155:6.

[14] X. Wang, J. F. Martínez, Rebudget: trading off efficiency vs. fairness in market-based multicore resource allocation via runtime budget reassignment, ACM SIGPLAN Notices 51 (4) (2016) 19–32.

[15] A. Sharifi, A. K. Mishra, S. Srikantaiah, M. Kandemir, C. R. Das, PEPON: performance-aware hierarchical power budgeting for NoC based multicores, in: Proc. Int'l Conf. Parallel Architectures and Compilation Techniques, 2012, pp. 65–74.

[16] S. M. Zahedi, B. C. Lee, Ref: resource elasticity fairness with sharing incentives for multiprocessors, ACM SIGARCH Computer Architecture News 42 (1) (2014) 145–160.

[17] C. Isci, A. Buyuktosunoglu, C.-Y. Cher, P. Bose, M. Martonosi, An analysis of efficient multi-core global power management policies: maximizing performance for a given power budget, in: Proc. IEEE/ACM Int'l Symp. microarchitecture, 2006, pp. 347–358.

[18] X. Li, G. Yan, Y. Han, X. Li, Smartcap: user experience-oriented power adaptation for smartphone's application processor, in: Proc. Conf. Design, Automation and Test in Europe, 2013, pp. 57–60.

[19] K. Ma, X. Wang, Pgcapping: exploiting power gating for power capping and core lifetime balancing in cmps, in: Proc. Int'l Conf. Parallel Architectures and Compilation Techniques, 2012, pp. 13–22.

[20] X. Wang, B. Zhao, T. Mak, M. Yang, Y. Jiang, M. Daneshtalab, On fine-grained runtime power budgeting for networks-on-chip systems, IEEE Trans. Computers 65 (9) (2016) 2780–2793.

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

[22] A.-M. Rahmani, M.-H. Haghbayan, A. Kanduri, A. Y. Weldezion, P. Liljeberg, J. Plosila, A. Jantsch, H. Tenhunen, Dynamic power management for many-core platforms in the dark silicon era: A multi-objective control approach, in: IEEE/ACM Int'l Symp. Low Power Electronics and Design (ISLPED), IEEE, 2015, pp. 219–224.

[23] A. M. Rahmani, M.-H. Haghbayan, A. Miele, P. Liljeberg, A. Jantsch, H. Tenhunen, Reliability-aware runtime power management for many-core systems in the dark silicon era, IEEE Trans. Very Large Scale Integration (VLSI) Systems 25 (2) (2017) 427–440.

[24] Wang, Xiaohang and Zhao, Baoxin and Wang, Ling and Mak, Terrence and Yang, Mei and Jiang, Yingtao and Daneshtalab, Masoud, A pareto-optimal runtime power budgeting scheme for many-core systems, Microprocessors and Microsystems 46 (2016) 136–148.

[25] T. Moscibroda, O. Mutlu, Memory performance attacks: denial of memory service in multi-core systems, in: Proc. Symp. USENIX Security, 2007, pp. 18:1–18:18.

[26] C. Karlof, D. Wagner, Secure routing in wireless sensor networks: attacks and countermeasures, Ad hoc networks 1 (2) (2003) 293–315.

[27] L. S. Indrusiak, J. Harbin, M. J. Sepulveda, Side-channel attack resilience through route randomisation in secure real-time networks-on-chip, in: Int'l Symp. Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2017, pp. 1–8.

[28] J. Lee, T. Kim, J. Huh, Reducing the memory bandwidth overheads of hardware security support for multi-core processors, IEEE Trans. Computers 65 (11) (2016) 3384–3397.

[29] A. K. Biswas, S. Nandy, R. Narayan, Network-on-chip router attacks and their prevention in mp-socs with multiple trusted execution environments, in: Proc. IEEE Int'l Conf. Electronics, Computing and Communication Technologies (CONECCT), 2015, pp. 1–6.

[30] Xiao, Kan and Forte, Domenic and Jin, Yier and Karri, Ramesh and Bhunia, Swarup and Tehranipoor, Mohammad, Hardware trojans: Lessons learned after one decade of research, ACM Trans. on Design Automation of Electronic Systems (TODAES) 22 (1) (2016) 6–28.

[31] M. Tehranipoor, F. Koushanfar, A survey of hardware trojan taxonomy and detection, IEEE Trans. Design Test of Computers 27 (1) (2010) 10–25.

[32] S. Bhunia, M. S. Hsiao, M. Banga, S. Narasimhan, Hardware trojan attacks: threat analysis and countermeasures, Proc. IEEE 102 (8) (2014) 1229–1247.

[33] A. Ganguly, M. Y. Ahmed, A. Vidapalapati, A denial-of-service resilient wireless noc architecture, in: Proc. Great Lakes Symp.

VLSI, 2012, pp. 259–262.

[34] S. R. Hasan, S. F. Mossa, C. Perez, F. Awwad, Hardware trojans in asynchronous fifo-buffers: from clock domain crossing perspective, in: Proc. IEEE Int'l Midwest Symp. Circuits and Systems (MWSCAS), 2015, pp. 1–4.

[35] T. Boraten, A. K. Kodi, Mitigation of denial of service attack with hardware trojans in noc architectures, in: Proc. IEEE Int'l Symp. Parallel and Distributed Processing, 2016, pp. 1091–1100.

[36] C. Gómez, M. E. Gómez, P. López, J. Duato, Reducing packet dropping in a bufferless noc, in: Proc. European Conf. Parallel Processing, 2008, pp. 899–909.

[37] S. T. King, J. Tucek, A. Cozzie, C. Grier, W. Jiang, Y. Zhou, Designing and implementing malicious hardware., Large-Scale Exploits and Emergent Threats (LEET) 8 (2008) 1–8.

[38] M. Hussain, H. Guo, Packet leak detection on hardware-trojan infected nocs for mpsoc systems, in: Proc. Int'l Conf. Cryptography, Security and Privacy, 2017, pp. 85–90.

[39] J. Frey, Q. Yu, Exploiting state obfuscation to detect hardware trojans in noc network interfaces, in: Proc. IEEE Int'l Midwest Symp. Circuits and Systems (MWSCAS), 2015, pp. 1–4.

[40] Y. Wang, G. E. Suh, Efficient timing channel protection for on-chip networks, in: Proc. IEEE/ACM Int'l Symp. Networks on Chip (NoCS), 2012, pp. 142–151.

[41] H. M. Wassel, Y. Gao, J. K. Oberg, T. Huffmire, R. Kastner, F. T. Chong, T. Sherwood, Surfnoc: a low latency and provably non-interfering approach to secure networks-on-chip, in: ACM SIGARCH Computer Architecture News, Vol. 41, 2013, pp. 583–594.

[42] M. Kayaalp, N. Abu-Ghazaleh, D. Ponomarev, A. Jaleel, A high-resolution side-channel attack on last-level cache, in: Proc. Design Automation Conference (DAC), ACM, 2016, pp. 72:1–72:6.

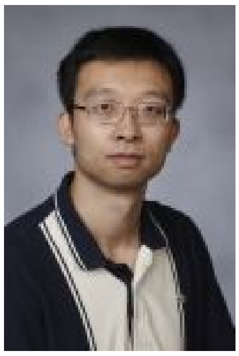
[43] C. J. Watkins, P. Dayan, Q-learning, Machine learning 8 (3-4) (1992) 279–292.

[44] X. Wang, M. Yang, Y. Jiang, P. Liu, M. Daneshtalab, M. Palesi, T. Mak, On self-tuning networks-on-chip for dynamic network-flow dominance adaptation, ACM Trans. Embedded Computing Systems (TECS) 13 (2s) (2014) 73:1–73:21.

Biography

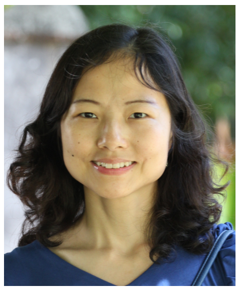
Yiming Zhao is an undergraduate in School of Software Engineering, South China University of Technology. His research interests include networks-on-chip architecture and hardware security.

Xiaohang Wang received the B.Eng. and Ph.D degree in communication and electronic engineering from Zhejiang University, in 2006 and 2011. He is currently an associate professor at South China University of Technology. He was the receipt of PDP 2015 and VLSI-SoC 2014 Best Paper Awards. His research interests include many-core architecture, power efficient architectures, optimal control, and NoC-based systems.



Yingtao Jiang joined the Department of Electrical and Computer Engineering, University of Nevada, Las Vegas in Aug. 2001, upon obtaining his Ph.D degree in Computer Science from the University of Texas at Dallas. He has been a full professor since July 2013 at the same university, and now assumes the role of the Department Chair. His research interests include algorithms, computer architectures, VLSI, networking, nano-technologies, etc.

Liang Wang received the B.Eng. and MSc degree in electronics engineering from Harbin Institute of Technology, Harbin, China, in 2011 and 2013 respectively, and the Ph.D degree in Computer Science and Engineering from The Chinese University of Hong Kong, Hong Kong, China, in 2017. He is currently a postdoctoral research fellow at Institute of Microelectronics, Tsinghua University, China. His research interests include power-efficient and reliability-aware design for network-on-chip and many-core system. He was a recipient of the VLSI-SoC 2014 Best Paper Awards.



Mei Yang received her Ph.D in Computer Science from the University of Texas at Dallas in Aug. 2003. She has been a full professor in the Department of Electrical and Computer Engineering, University of Nevada, Las Vegas since 2016. Her research interests include computer architectures, networking, and embedded systems.

Amit Kumar Singh received the B.Tech. degree in Electronics Engineering from Indian Institute of Technology (Indian School of Mines), Dhanbad, India, in 2006, and the Ph.D degree from the School of Computer Engineering, Nanyang Technological University(NTU), Singapore, in 2013. He was with HCL Technologies, India for year and half before starting his PhD at NTU, Singapore, in 2008. He worked as a post-doctoral researcher at National University of Singapore (NUS) from 2012 to 2014 and at University of York, UK from 2014 to 2016. Currently, he is working as senior research fellow at University of Southampton, UK. His current research interests include system level design-time and run-time optimizations of 2D and 3D multi-core systems with focus on performance, energy, temperature, and reliability. He has published over 45 papers in the above areas in leading international journals/conferences.

Terrence Mak is an Associate Professor at Electronics and Computer Science, University of Southampton. Supported by the Royal Society, he was a Visiting Scientist at Massachusetts Institute of Technology during 2010, and also, affiliated with the Chinese Academy of Sciences as a Visiting Professor since 2013. Previously, He worked with Turing Award holder Prof. Ivan Sutherland, at Sun Lab in California and has awarded Croucher Foundation scholar. His newly proposed approaches, using runtime optimization and adaptation, strengthened network reliability, reduced power dissipations and significantly improved overall on-chip communication performances. Throughout a spectrum of novel methodologies, including regulating traffic dynamics using network-on-chips, enabling unprecedented MTBF and to provide better on-chip efficiencies, and proposed a novel garbage collections methods, defragmentation, together led to three prestigious best paper awards at DATE 2011, IEEE/ACM VLSI-SoC 2014 and IEEE PDP 2015, respectively. More recently, his newly published journal based on 3D adaptation and deadlock-free routing has awarded the prestigious 2015 IET Computers & Digital Techniques Premium Award. He has published more than 100 papers in both conferences and journals and jointly published 4 books.