Lightweight Mitigation of Hardware Trojan Attacks in NoC-based Manycore Computing

Venkata Yaswanth Raparti, Sudeep Pasricha
Department of Electrical and Computer Engineering
Colorado State University, Fort Collins, CO, U.S.A.
yaswanth@rams.colostate.edu, sudeep@colostate.edu

ABSTRACT

Data-snooping is a serious security threat in NoC fabrics that can lead to theft of sensitive information from applications executing on manycore processors. Hardware Trojans (HTs) covertly embedded in NoC components can carry out such snooping attacks. In this paper, we first describe a low-overhead snooping invalidation module (*SIM*) to prevent malicious data replication by HTs in NoCs. We then devise a snooping detection module (*THANOS*) to also detect malicious applications that utilize such HTs. Experimental analysis shows that unlike state-of-the-art mechanisms, *SIM* and *THANOS* not only mitigate snooping attacks but also improve NoC performance by 48.4% in the presence of these attacks, with a minimal ~2.15% area and ~5.5% power overhead.

1. INTRODUCTION

With the rise in number of processing cores and growing parallelism in applications, the communication traffic in a manycore processor has been increasing. Chip designers and manufacturers are moving towards network-on-chip (NoC) as their de-facto intra-chip communication fabric [1]-[2]. Typically, emerging manycore processors have tens to hundreds of components that are designed either by in-house engineers or obtained from third-party vendors (3PIP), and then finally integrated together in a single global facility. With the growing complexity in NoC design, designers are opting for third-party NoC IPs, e.g., [3], to connect the components in their processors. This global trend of distributed design, validation, and fabrication has led to major challenges in ensuring secure execution of applications on manycore platforms, in the presence of potentially untrusted hardware and software components.

Much work has been done to mitigate side-channel attacks on shared resources and to detect counterfeit ICs that compromise manycore chip performance [4], [6]. This work focuses on an orthogonal attack scenario where an adversary can insert a hardware Trojan (HT) into the RTL or the netlist of a manycore processor to disrupt or alter the integrity of its behavior without being detected at the post silicon verification stage. HTs can be inserted by an intellectual property (IP) vendor, untrusted CAD tool/designer, or at the foundry via reverse engineering [5]. We focus on one such attack called a data-snooping attack where a malicious software and an HT work together to steal information from applications executing on manycore processors.

NoCs are ideal candidates for such attacks as they have a complex design that can be used to hide an HT which cannot be easily detected via functional verification. HTs can be placed in NoC links, routers, or network interfaces (NIs) to secretly snoop on the data or corrupt data passing through them. Typically, in data-snooping attacks HTs create duplicate packets with modified headers and send them into the NoC for an accomplice thread to receive them [11]. Several works propose packet encoding/error correction mechanisms such as parity bits and ECC in NoC packets to detect faulty data packets at the receiver [7]-[8]. Other works such as [9]-[12] have also proposed data protection mechanisms in the presence of an HT in NoC components. However, there are three major shortcomings with the state-of-the-art: (1) these works assume the presence of HTs in NoC routers or links which can be detected by physical inspection or functional verification, without employing costly security mechanisms; (2) the mechanisms proposed in prior works protect application data from snooping attacks but do not detect the attack and mitigate future attacks; and (3) most of the security enhancement mechanisms are costly to implement and increase NoC latency and power consumption which worsens the overall performance. *It is important to design and deploy lightweight mechanisms that can detect the operation of malicious HTs embedded in NoCs and accomplice threads, and secure against their data-snooping attacks in emerging manycore processors.*

In this work we focus on security enhancement that do not notably increase performance and power overheads. We provide robust yet low-power mechanisms to detect the source of the attacks by utilizing controlled aging in circuits at runtime, which is not easy to obfuscate or tamper with in the design and fabrication process. Our novel contributions in this work are as follows:

- We first design and demonstrate a data-snooping attack using an HT in the NoC interface that duplicates packets and injects them into the NoC with a minimal area and power footprint, making it difficult to detect by traditional functional verification mechanisms;
- We then protect against such data-snooping attacks by proposing a novel snooping invalidation module (*SIM*) that uses an encoding-based duplicate packet detection mechanism;
- We further propose a novel data-snooping detection circuit called *THANOS* that uses threshold voltage degradation as a means to detect an on-going attack at runtime and blacklist the malicious software task that initiated the attack;
- Experimental analysis shows that *SIM* with *THANOS* provides security against HTs with minimal area and power overhead.

2. RELATED WORK

Significant research has been done to increase robustness against attacks by HTs in NoCs by assuming that an HT tampers or snoops data passing through it. In [9], bit shuffling and Hamming ECC are used to reduce the effectiveness of HTs that corrupt data. In [10], security zones managed by a centralized security manager are proposed to protect sensitive information from being accessed by malicious agents. In [11] data scrambling, packet authentication, and node obfuscation are proposed to prevent data stealing by a compromised NoC. Data scrambling, and packet-authentication mechanisms use a one-time pad XOR cipher that can be broken by the malicious tasks when enough encrypted packets are accumulated. In [12], CRC and algebraic manipulation detection (AMD) are used to encode packet headers to safeguard from faults and snooping attacks. In [13], a novel wave-based scheduling mechanism for NoCs is proposed that eliminates the need for TDMA-based NoC resource sharing, hence providing non-interference between different domains of applications. In [14], a process variation-

DAC '19, June 2–6, 2019, Las Vegas, NV, USA

ACM ISBN 978-1-4503-6725-7/19/06…$15.00
DOI: 10.1145/3316781.3317851

based packet encoding and decoding mechanism is proposed to prevent data-snooping in silicon photonic NoCs. *Most of these schemes that protect application data from NoC security attacks lack an efficient and low-power attack detection mechanism which makes them incomplete in providing security.*

A few works address HT detection in NoC components at design-time and runtime. At design time, techniques such as physical inspection [15], functional testing [16], and side channel analysis [17] have been proposed. But testing for HTs at design time is still in infancy, and the growing complexity of NoC components make this even more difficult. Hence, designers are now exploring runtime detection methods. A key logic built-in self-test (LBIST) was proposed in [18] that uses test vectors generated by programmable keys to detect Trojans. However, LBIST requires that the chip operation should be paused while testing at regular and frequent intervals, which is not suitable for NoCs that should function seamlessly. A few other works such as [19], [20] propose in-situ HT detection modules that rely on verification units placed in NoC components to detect HTs. There are two limitations with all of these works: (1) the verification units used to detect HTs can also be reverse-engineered and tampered, (2) these mechanisms are used to detect only HT induced data-corruption attacks. Data-snooping attacks unlike data-corruption attacks attempt to leak critical application data to malicious software tasks. *None of the prior works have addressed the problem of detecting the software task that initiates data-snooping attacks to blacklist and prevent future attacks.*

In [21], a run-time technique called NoCAlert is proposed to detect failures in the control logic of NoC components. This technique is further enhanced by [22] that proposes modules which alert the host system if the control logic in NoC routers detects invariance violations caused by HTs placed in its control-path, e.g., logic for route computation (RC) or virtual channel allocation (VCA). However, these techniques focus on NoC components that have substantial control logic, such as routers. They ignore the network interface (NI) which prevents easy placement of model checkers to detect packet duplication. In this paper we propose a novel snooping invalidation module (*SIM*) in the NI that can mitigate snooping attacks. We then propose low-overhead techniques to detect the source of data-snooping attacks in NoCs. *To the best of our knowledge, this is the first work that mitigates snooping attacks in NoCs with minimal performance and power overheads, while also detecting the source of snooping attack to protect against future attacks.*

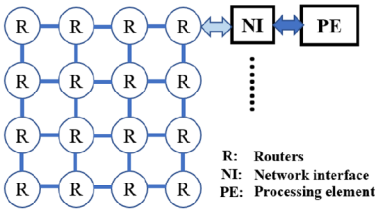

Figure 1: Baseline NoC architecture with example routers, a PE and an NI

3. BACKGROUND AND ATTACK MODEL

3.1 Background

In this section we discuss our assumed baseline NoC design. We consider a traditional 2D mesh based NoC with processing elements (PE) connected to the NoC via a network interface (NI). The packets entering the NoC are routed towards their destination by routers that use a hop-by-hop, turn-based distributed deadlock free XY routing algorithm. Figure 1 shows the schematic of the baseline 2D mesh NoC with an NI and PE connected to routers. We use traditional 3-stage {*buffer write, RC+VCA+SA, LT*} pipelined routers in the NoC with wormhole switching and 4-VC buffers at each input port. PEs communicate using messages that are passed to the NI which packetizes them before sending them to the NoC. The packets received by the NI from the NoC are de-packetized and sent to the connected PE. We consider ARM Cortex-A73 cores in our PEs that use the AXI interface for communication. Each PE has a private L1 cache and a shared distributed L2 cache that uses a scalable directory-based cache coherence protocol to send messages in the form of NoC packets.

3.2 Attack Model

Prior works [9]-[12] assumed data-snooping attacks to be carried out by HTs embedded in NoC routers or by compromised links that enable HTs to modify the packet headers. These HTs, once activated by a flit with a special activation sequence, make copies of packets passing through the router and transmit them to the PE that has a malicious accomplice task running on it. Once an HT is activated in a router, it generates new packets, or diverts an existing packet to the PE running the accomplice task. This type of HT, that has a high 4% area overhead [11], may be noticed by testers while conducting physical inspection or side channel analysis. Moreover, this type of attack can lead to illegal utilization of router resources such as buffers, VCs, and switch allocators, which cause control logic violations that can be detected by secure model checkers [22]. We thus focus on a harder-to-detect attack with an HT embedded in the NI where packets are generated, and hence packets can be duplicated with relatively simpler logic without interfering with the basic NI functionality.

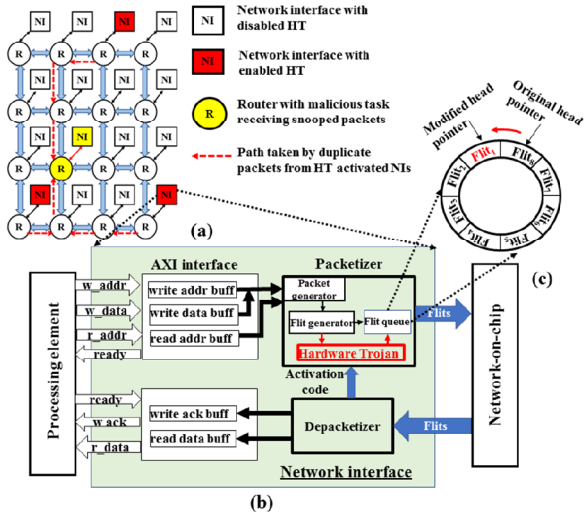

Figure 2: (a) Overview of attack model on a NoC with a malicious software task coordinating the data-snooping attack, (b) microarchitecture of network interface (NI) with a hardware Trojan embedded in packetizer module, (c) FIFO queue modification by hardware Trojan.

Figure 2(a) shows an overview of an on-going data-snooping attack taking place in a 2D NoC based manycore processor with multiple HTs activated in the NoC NI modules shown in red, and a malicious task running on a PE connected to the yellow router and NI. The HTs make duplicate copies of packets in NIs that are sent to the malicious task.

3.3 Design Details: Network Interface with a Hardware Trojan

Figure 2(b) shows the microarchitecture of an NI with an embedded HT in the packetizer module. The NI receives messages from the PE via the AXI interface that are then stored in its buffers. The messages usually are read/write commands with address and data fields. The packetizer module appends source ID, destination ID, and virtual channel ID information to the commands and creates packets. A packet is further divided into flits, with the header flit containing the NoC routing related information. The packet flits are then injected into the circular flit queue that is accessed via head and tail pointers. After the packetizer injects a flit, the tail pointer of the queue is incremented. After a flit is transmitted to a router, the head pointer is incremented to transmit the next flit.

An HT can potentially tamper with the pointer values to re-send duplicate packets intelligently. Once a flit has been transmitted from NI to the router, it stays in the cyclic queue until a new flit is overwritten on that location. The HT can keep track of these locations to read a header

flit that has already been transmitted to the router, append it with a duplicate destination ID of the malicious node, and *update the head-pointer*. Figure 2(c) shows how the HT modifies the head pointer. By moving the head pointer at regular intervals, the HT can send duplicate packet flits without having to store them externally. The duplicate packet is re-sent to the router for transmission. If the flit queue is full (head pointer = tail pointer), both the HT and packetizer do not inject new flits into the queue, and do not accept any more incoming data from the PE until the outstanding flits are transmitted. The HT does not interfere with the control logic which is mostly present in the AXI interface, and an attacker can snoop on data using this HT in NIs between two PEs, or between a PE and a memory controller that is connected to main memory channels.

We now perform an overhead analysis of this HT. The proposed HT requires an internal memory to save the head flit to modify its destination ID, save the header pointer of the queue, and save the current state of the HT (~72 bits). We designed the NI shown in figure 2(b) by modifying the CONNECT open-source NoC model [24] and used Xilinx's Vivado HLS [25] tool to analyze the overheads. Table 1 shows the clock cycle period, number of flipflops and LUTs used for an FPGA implementation of the packetizer. The optimized design indicates that the NI with an HT requires an additional ~5% FFs and ~1% LUTs (1.3% area overhead) without incurring additional timing latency. This low overhead HT can be inserted at the RTL level, or by reverse engineering and changing the netlist at the place and routing stage [5], [15]. The small size of the HT makes it hard to detect by physical inspection or by side-channel analysis. Also, the run-time secure model checkers from [21], [22] are not able to check the validity of flits in the NI as it does not interfere with the control logic. *Hence, there is a need to design a low-overhead flit validation module in NIs to check flit validity before injecting them into the NoC.*

Table 1. FPGA Implementation of NI packetizer with and without Hardware Trojan (HT)

	Timing (ns)	Number of FFs	Number of LUTs
NI without HT	3.45	258	535
NI with HT	3.45	273	549

4. MITIGATION OF NOC SNOOPING ATTACKS

We propose a novel framework that integrates two mechanisms to mitigate data snooping attacks from taking place, as well as to detect the source of an on-going attack, and protect against future snooping attacks. We rule-out data corruption attacks as they can be detected and corrected using ECC codes such as in [12]. Our proposed framework consists of two security mechanisms, *(1) a snooping invalidator module at the NI output queue to discard duplicate packets, (2) detection of data-snooping attacks at the PE where an accomplice thread is executing.* This comprehensive protection framework ensures that we proactively mitigate future attacks and safeguard the application data for the entire lifetime of the processor. We have designed our security mechanism to be hard to be tampered by adversaries that use reverse engineering techniques to insert HTs in the netlist. Our approach also works irrespective of the HT triggering process to start snooping attacks such as special flit data, circuit aging, or temperature [23]. The following sections discuss our two security mechanisms.

4.1 Security enhanced NI: preventing data-snooping attack

The first security enhancement mechanism is to prevent a snooping attack with the help of a snooping invalidator module (*SIM*) at the NI. Using *SIM*, we aim to discard packets with invalid header flits from being injected into the NoC. Unlike traditional ECC-based security enhancement mechanisms, *SIM* incurs low-power and low latency overheads because of its lightweight computations that are designed solely to mitigate snooping attacks. Figure 3 (a) shows an overview of the security enhancement in NI using *SIM*.

We divide the implementation of *SIM* across the PE and NI to prohibit 3PIP NoC designers/testers to reverse engineer or tamper with the secret encoding/decoding information at runtime. The PE and NI communicate using the standard AXI hand-shake protocol (ready, valid, and valid ready signals). A typical NI receives messages from the PE to be packetized and sent to the NoC and vice-versa. In the security enhanced NI, additional encoding information (*key*) is attached with the data received from the PE to validate the uniqueness of data packets. The numbered sequence of steps shown in Figure 3 describe how a packet is validated using *SIM*. These steps are discussed next.

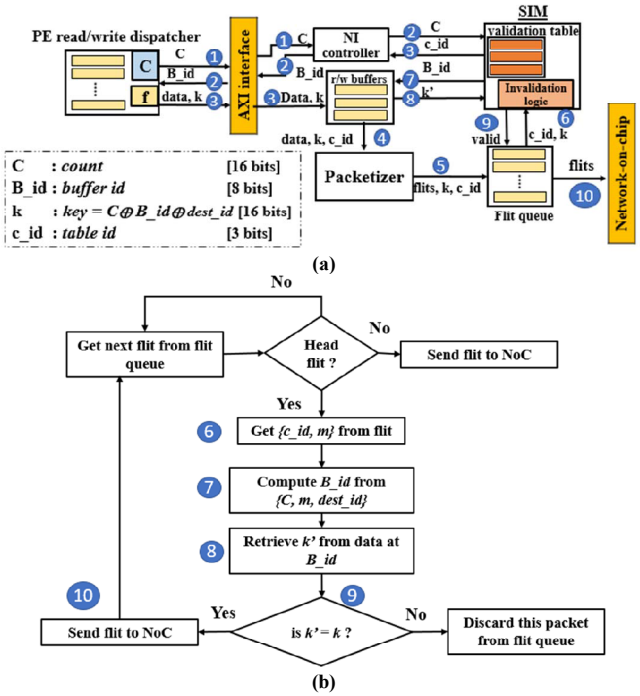

Figure 3: (a) Security enhanced NI using SIM (b) Flowchart of snooping invalidation mechanism in NI

In step 1, the PE data dispatcher sends a *count* (*C*: increments with each outgoing data) value to the NI controller along with the AXI ready signal. In step 2, the NI controller sends a buffer id (*B_id*) that is reserved to store the incoming data along with the AXI valid signal to the PE data dispatcher. The NI controller simultaneously sends *C* to *SIM* that stores it in a "validation table". The PE uses an XOR function *f* to generate an encoded key *k* as a function of *C*, *B_id*, and destination ID (*dest_id*) of the packet, as shown in eq. (1) below. In step 3, the PE sends a message *{data, k}* combination to the data buffers and toggles the valid ready signal to high. At the same time, the validation table sends a *c_id* (location where *count* is stored in the table) to the NI controller. This is stored with the message sent by the PE. The *{data, k, c_id}* combination is stored as a unit in read/write buffers till the packet is sent out of the NI. While the size of payload *data* varies from 8B to 128B depending on message type, the values of *k, C* and *c_id* require only few bits of storage (see legend of figure 3(a)).

$$k = C \oplus B\_id \oplus dest\_id \quad (1)$$

$$B\_id = C \oplus k \oplus dest\_id \quad (2)$$

In step 4, the *{data, k, c_id}* combination is sent to the packetizer to generate packets. In step 5, flits are generated with *k* and *c_id* copied into the header flit. We save *k* and *c_id* in the 24-bits reserved in the header flit to store destinations of source-routing path [26] which are unused as we adopt distributed routing for our NoC. The flits are then saved in the output flit queue. *Steps 6-9 are part of snooping invalidation flow explained in more detail in figure 3(b). SIM* tries to retrieve the encoded key *k* from the *C* entry in the validation table as a part of packet validation. In step 6, *SIM* reads *dest_id, k, and c_id* bits of the header flit, and performs a decoding operation shown in eq. (2) to obtain *B_id* of the

buffers that stored the corresponding packet data, and *k* sent by the PE (step 7). In steps 8 and 9, *SIM* retrieves the value of *k'* stored in the buffer located at *B_id* and compares with *k* that is read from the header flit. If *k=k', SIM* sends a valid signal that the header flit is valid, and it is injected into the NoC. If *SIM* sends an invalid signal, the flit queue discards all the flits corresponding to the duplicate packet. *SIM* efficiently detects duplicate packets because, if the value of *dest_id* is modified by the HT, eq. (2) leads to an incorrect value of *B_id* that does not retrieve the *k* value corresponding to the data of packet sent in step 3. Note that for broadcast/multicast packets, multiple keys are generated for each *dest_id* value and key verification steps 8 and 9 are performed on each of them separately. After a packet is sent out, the corresponding read/write data buffer and validation table entries are reused for new data. *This low-overhead SIM module with minor modifications can also be used to curb potential data duplication at router-link interfaces or within a router.*

4.1.1 Overhead analysis

Several steps in the *SIM* module can be performed in parallel. The existing communication data channel between the PE and the NI that is established by AXI interface is used to communicate both packet data and *SIM* metadata (*C, k, B_id* in steps 1 and 2). Hence, no additional wires are needed to transmit *SIM* metadata. Steps 1 and 2 are performed in parallel with AXI interface's ready and valid signal exchange to minimize the latency overhead. Also, there is no additional overhead involved in steps 3 to 5. Steps 6 to 9 take one cycle in a NoC that is clocked at 1GHz frequency which was verified via FPGA synthesis of the modified NI [25]. This increases the number of pipeline stages of the NI microarchitecture. *SIM* takes additional memory to maintain a validation table, and additional logic to perform XOR and comparison operations. *SIM* incurs ~5.5% more power and ~2.15% more area overhead compared to the baseline NI with a buffer capacity of 16 packets, at the 22nm technology node.

4.2 Detecting the source of a data-snooping attack

Using our security enhanced NI with the integrated *SIM*, we can curb packet duplication at the NI. However, the malicious task that is the source of the attack is still not detected that could initiate attacks from compromised routers or links [12]. In this section, we propose a module called *THANOS,* a novel threshold activated snooping attack detector that is implemented at the interface between an NI and a PE, as shown in figure 4(a) to detect the source of the snooping attack.

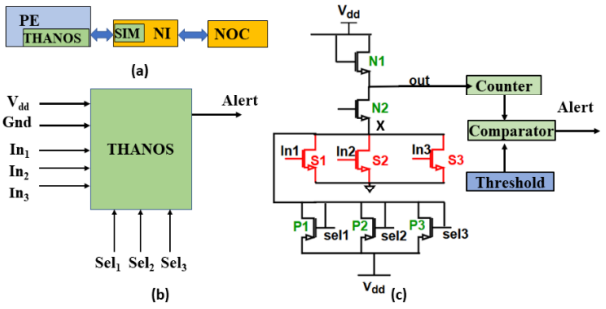

Figure 4: (a) Overview of THANOS (b) block diagram of THANOS showing inputs and outputs (c) snooping detecting circuit used in THANOS

A PE sends and receives various types of messages into the NoC that can be broadly classified into two types: (1) direct messages between cores for inter-core communication, and (2) cache-coherence messages between a PE and directory table. Figure 5 (blue bars) shows the average incoming-outgoing message ratio sent over 64 cores in a NoC by different PARSECv2.1 [27] benchmark applications with 64 tasks each. The error-bars in figure 5 represent variance across NoC nodes. Figure 5 shows that the ratio is less than 1 over all the benchmarks with each node receiving a smaller number of messages than the messages it sends out (number of "packets" in a "message" can vary based on message type).

Another important observation from figure 5 is that the incoming-outgoing message ratio is much greater than 1 (red points) when a data-snooping attack takes place, because a PE receives significantly higher number of messages (and packets) than it sends out. This phenomenon can be easily detected in the short term by placing a counter in the NI and observing the number of incoming and outgoing messages over an epoch of time. However, observing messages in the short term can lead to false positives, e.g., due to periodic bursts of messages from a task that requires higher volumes of input data. Also, a message counter is not the most secure way to detect a snooping attack, given the reverse engineering techniques available to tamper digital logic [15].

In *THANOS* we devise a mechanism that observes the ratio of incoming and outgoing messages over a period of few hours and identifies the source of a snooping attack. *THANOS* is designed using a combination of analog and digital logic to detect if a PE is snooping on messages over a duration of time. As *THANOS* is not entirely a digital logic implementation, it is hard to reverse engineer or tamper with, and can be used as a final frontier to mitigate data-snooping attacks. *THANOS* receives inputs from the PE and sends a security alert signal to the PE as shown in figure 4(b). The PE then identifies the source of data-snooping attacks and takes preventive steps to mitigate future attacks. *THANOS* is designed as a standalone module that can also be used with prior data protection schemes [9]-[12] to detect the source of attack.

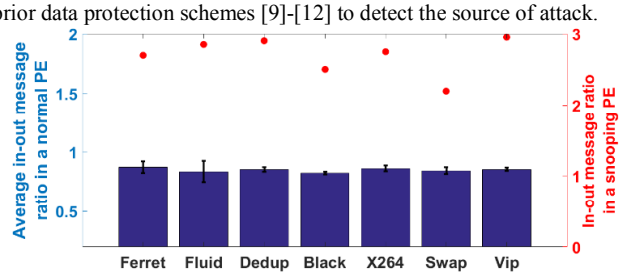

Figure 5: Average incoming-outgoing message ratio at normal PE (left), and at snooping PE (right) across different applications

4.2.1 Overview of snooping detection circuit

We take inspiration from a controlled aging module [6] that uses threshold degradation of NMOS transistors due to aging phenomenon such as bias temperature instability (BTI) and hot carrier injection (HCI) to detect chip usage, which helps identifying counterfeit ICs. In *THANOS* we use NMOS threshold voltage degradation to detect a PE that is receiving duplicate packets injected by multiple HT activated NIs in the NoC. NMOS transistors undergo stress-recovery periods in their ON and OFF operations that leads to threshold voltage (V_{th}) degradation [28]. Figure 6 shows the V_{th} degradation observed across different ratios of stress and recovery in an NMOS transistor at 22nm using the long-term aging model proposed in [28]. At 100% stress (no recovery) the V_{th} of a transistor increases by ~ 100mV in about two hours duration. We use this phenomenon to detect snooping attacks.

4.2.2 Operation of snooping detection circuit

In our snooping detecting circuit shown in figure 4(c), there are 2 transistors; N_1 that acts as a diode connected load and N_2 that acts as gate-source voltage (V_{gs}) sensor. P_1, P_2, P_3 are diode-connected PMOS transistors that pull the drain voltages of S_1, S_2, S_3 to high. Transistors S_1, S_2, S_3 are driven using low over-drive voltages In_1, In_2, In_3 that barely switch them ON. *We artificially induce stress in a selected transistor among $S_1/S_2/S_3$ when a message is received and induce recovery when a message is sent out. Hence, we call them stress-transistors.* At any point only one of S_1,S_2,S_3 are connected to the circuit (using *In* and *sel* signals). When $S_1/S_2/S_3$ is turned ON, the source (V_x) of N_2 is pulled low, which turns ON N_2, leading to a "low" *out* state. But, when a stressed transistor ($S_1/S_2/S_3$) undergoes V_{th} degradation, its over-drive voltage ($In = V_{gs}-V_{th}$) is not high enough to turn ON the stress-transistor and hence drives it into the triode region. When $S_1/S_2/S_3$ is in the triode-region, the source voltage (V_x) of N_1 is not pulled low and the *out* signal is set to "high".

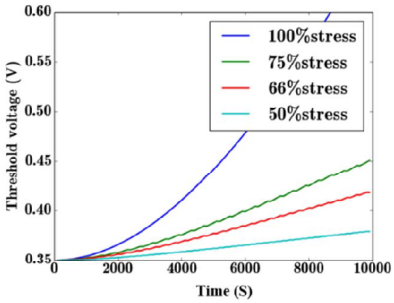

Figure 6: Threshold voltage degradation observed across different stress-recovery ratios in a NMOS transistor at 22nm technology node

Table 2: State transition of snooping detection circuit

Stress-transistors ($S_1/S_2/S_3$)	V_x	N_2	out
Saturated	Low	ON	Low
Triode	High	OFF	High

Table 2 gives the states of different transistors and the corresponding changes in *out* signal state. When a PE is not receiving snooped packets, its incoming-outgoing message ratio is less than 1 as shown in figure 5. Hence, for normal NoC traffic the stress-recovery ratio of stress-transistors ($S_1/S_2/S_3$) is less than 40%. Generally, BTI and HCI are slow wear-out phenomenon in logic circuits. But, we input low over-drive (V_{gs}-V_{th}) voltage of ~100mV to the stress-transistors through input signals $In_1/In_2/In_3$. Hence, the circuit would set the *out* signal to high state in a duration of 2-3 days. However, when a malicious task on a PE is snooping with up to four HT activated in NIs, its incoming-outgoing message ratio is 3× the average ratio (shown in figure 5). As a result, the stress-transistors in *THANOS* undergo 80-90% more stress than recovery when there is a snooping attack. From figure 6, when a stress-transistor receives ~90% stress, its threshold voltage increases over a shorter duration (~3-4 hours). Hence the snooping detection circuit toggles the *out* signal to "high" state quicker when PE receives snooped packets.

In *THANOS*, we use a counter to track the time taken for the *out* signal to change its state and compare it with a threshold time that is configured by a trusted PE firmware, as shown in figure 4(c). *THANOS* sends an ALERT signal when the time taken by the *out* signal to switch the state is less than the threshold. *Overall, THANOS sends a notification about a potential malicious task anywhere from ~2 hours to ~2 days based on the number of HTs that are active.* The trusted PE firmware then alerts the OS about the malicious application task executing on the PE, so that preventive measures can be taken.

The snooping detection circuit should last for the lifetime of the processor to detect snooping attacks. However, due to artificially induced stress and recovery cycles, the stress-transistors (S_1, S_2, S_3) wear-out much more rapidly than the rest of the chip. To increase the lifetime of *THANOS* we take two measures: (1) We input low over-drive voltage (In-V_{th} ≈ 100mV) and high V_{dd} using separate power lines for stress-transistors; after every state change of the *out* signal, we increment the *In* signal by ~100mV until we satisfy the MOS saturation condition (In-V_{th} < V_{dd}); (2) The stress-transistors are over-provisioned; we use only one stress-transistor at any time to detect an attack and when an *In* voltage of a stress-transistor can no longer be incremented without violating the saturation condition, *THANOS* switches to the next stress-transistor using the *sel* signal. Using three stress-transistors and V_{dd} = 3V, *THANOS* can seamlessly detect snooping attacks for up to 1.5 years. The number of stress-transistors in *THANOS* is hence left to the decision of the designer. The overhead of *THANOS* is negligible in power (~50 μW) and area (~0.9 μm^2) compared to the PE (~1W, ~318mm^2) at 22nm technology node as it requires just 8 MOSFETs, a counter, a comparator, and a simple control logic block to send input signals.

5. EXPERIMENTS

We target a 64-core manycore chip with low power ARM cortex-A73 cores and a 2D mesh NoC with 8×8 dimension to test the performance, latency, energy, and area overheads of the proposed lightweight snooping invalidation module *(SIM)* and snooping detection circuit *(THANOS)* compared to the state-of-the-art. For simulations, we modeled the behavior of *SIM* and *THANOS* as part of the cycle-accurate NoC simulator Noxim [29]. We obtained the power and area overheads of *SIM* and *THANOS* modules from post-synthesis vectorless estimation in Vivado [25], and Cadence Virtuoso [31], at 22nm. We integrate the latency and energy overheads of *SIM* and *THANOS* with Noxim for our simulations. We tested our framework using PARSECv2.1 benchmark NoC traces generated by netrace [30] to capture the request-response dependencies to accurately simulate parallel application performance.

We compare our work with a baseline NoC (with a configuration that is described in section 3.1) with no security mechanism employed, and with two prior works, FortNoCs [11], and P-Sec [12]. In [11], only data obfuscation and data scrambling techniques are implemented for a fair comparison. In [12] end-to-end algebraic manipulation detection (AMD) and cyclic redundancy codes (CRC) are appended to the header flit for reliability against faults and HT attacks. We set the threshold time in the snooping detection circuit of *THANOS* as ~2.5 days to get a security violation alert. We first present results of application performance, NoC latency and NoC energy consumption for 4 actively snooping HTs that are randomly placed in NoC. Subsequently we present results for scenarios with 1 and 2 HTs operating in the NoC.

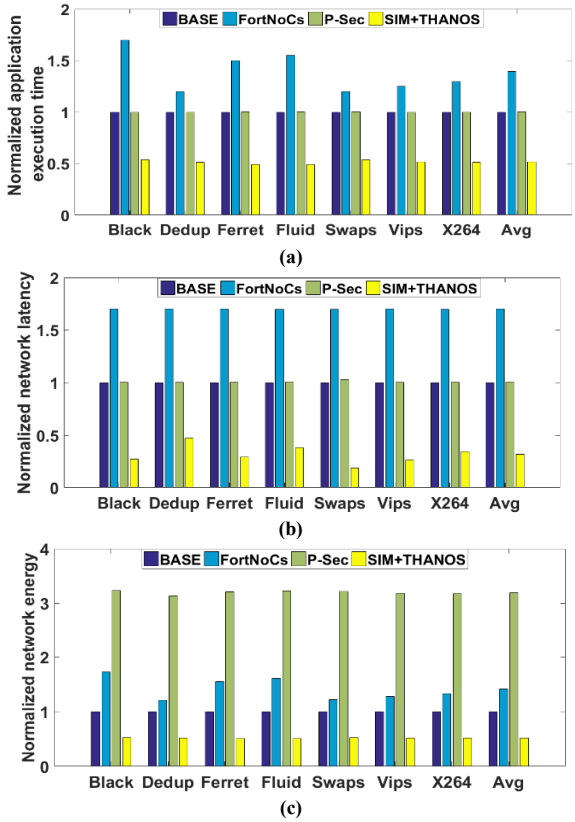

Figure 7: (a) Normalized application execution time, (b) normalized network latency, (c) normalized NoC energy consumption, across NoCs with different security mechanisms in the presence of 4 active HTs attempting to inject duplicate packets to an accomplice thread.

Figure 7(a) shows the comparison of application execution time across different NoC security mechanisms. P-Sec and FortNoCs cannot prevent the injection of duplicate packets at the NI, and only discard faulty packets at the receiver, which leads to higher NoC traffic. Moreover, P-Sec takes two extra cycles for CRC+AMD encoding/decoding, and FortNoCs takes at least four extra cycles at the NI for node obfuscation and data scrambling techniques on the entire packet. This leads to poor

application performance with P-Sec and FortNoCs compared to the baseline. *SIM* mitigates duplicate data packets near the source, resulting in less NoC, thereby actually achieving 48.4% average improvement in application execution time, compared to the baseline.

A similar trend is observed for network latency, shown in figure 7(b). The network latency of FortNoCs is higher due to the packet scrambling mechanism that encrypts/decrypts the entire packet using XOR operation, which is time consuming for packets with high payload size. FortNoCs incurs additional overhead due to packet authentication as an additional security mechanism. *SIM*, in comparison, takes one cycle only at the sending NI to detect duplicate packets, and *THANOS* has no latency overhead. In the absence of duplicate packets in the NoC, *SIM* has the lowest NoC latency, and achieves an average of 67.8%, 77.3% and 68.1% latency reduction compared to the baseline, FortNoCs, and P-Sec in the presence of active data-snooping HTs.

Next, we analyze NoC energy consumption. Although *SIM+THANOS* consumes ~5.5% additional NI power, its energy consumption is 47.8% lower compared to baseline on average due to the lower application execution time as shown in figure 7(c). FortNoCs consumes around 41.8% additional energy compared to the baseline due to increased execution time and the overheads incurred to employ XOR encryption/ decryption logic in the NI. P-Sec consumes up to 200% more energy compared to the baseline due to its costly AMD, and CRC codec engines present in NIs and NoC routers. P-Sec is thus much more expensive, although it provides combined safety against faults and snooping attacks.

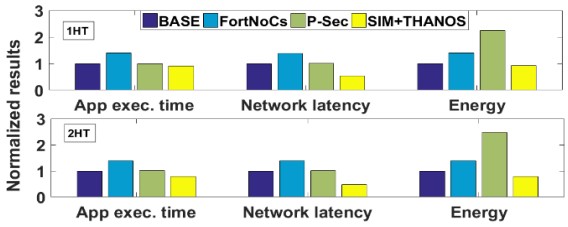

Figure 8: Normalized average values of application execution time, network latency, and NoC energy consumption across different security mechanisms with 1 HT (top), 2 HTs (bottom).

We observe similar trends in application execution time, NoC energy, and latency even when fewer number of HTs are active as shown in figure 8, with *SIM+THANOS* performing better than the baseline unlike FortNoCs and P-Sec. *This shows that our proposed snooping invalidation and snooping detection mechanisms, SIM+THANOS, does not trade-off NoC performance and NoC energy consumption to provide security.* Lastly, we compare area footprint of *SIM+THANOS* with other schemes. As shown in Table 3, *SIM+THANOS* has the lowest area footprint amongst the three security mechanisms. *SIM+THANOS* mechanism consumes only 2.15% additional area in the NI to implement the packet validation mechanism.

Table 3. Area footprint of different NoC security enhancement mechanisms

SIM+THANOS	FortNoCs	P-Sec
2.2 μm^2	4.9 μm^2	500μm^2

6. CONCLUSIONS

In this paper we proposed a low-overhead mechanism called *SIM* to prevent data-snooping attacks that are initiated by HTs embedded in NoC network interfaces. We also proposed a lightweight standalone snooping-attack detection mechanism called *THANOS* that uses controlled circuit aging to detect the source of attacks that can help processors take preventive steps to mitigate future attacks. In FortNoCs and P-Sec it is impossible to detect the source of the attack, which can be addressed by using *SIM+THANOS*. Experimental results show that *SIM+THANOS* reduces application execution time by 62.9% and 48.3% and energy consumption by 63.5% and 83.7% compared to FortNoCs and P-Sec. *SIM+THANOS* incurs a minimal additional 5.5% power and 2.15% area overhead, compared to the baseline, much lower than the overhead for FortNoCs and P-Sec. Thus *SIM+THANOS* represents a promising solution to enhance NoC security in manycore processors.

ACKNOWLEDGEMENT

This material is based upon the work supported by the National Science Foundation under grant CCF-1813370.

REFERENCES

[1] V.Y.Raparti, S. Pasricha "RAPID: Memory-Aware NoC for Latency Optimized GPGPU Architectures" IEEE Transactions on Multi-Scale Computing Systems, vol: 4, no: 4, pp. 874-887, Oct-Dec 2018.

[2] V.Y.Raparti, S. Pasricha "DAPPER: Data Aware Approximate NoC for GPGPU Architectures." Proc. IEEE/ACM NOCS 2018.

[3] Arteris, http://www.arteris.com/

[4] Y. Yarom et al., "FLUSH+ RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack." Proc. USENIX Security Sym. 2014.

[5] S. Bhunia, et al., "Hardware Trojan attacks: Threat analysis and countermeasures." Proc. IEEE, Vol: 102, no:8, pp :1229–1247, 2014.

[6] N.E.C. Akkaya, et al. "Secure chip odometers using intentional controlled aging." Proc. IEEE HOST 2018.

[7] D. Park et al. "Exploring fault-tolerant network-on-chip architectures." Proc. of the Intl. Conf. on Dependable Systems and Networks (DSN), 2006.

[8] S. Shamshiri, et al. "End-to-end error correction and online diagnosis for on-chip networks." Proc. of the International Test Conference (ITC), 2011.

[9] JYV Manoj Kumar, et al. "Run Time Mitigation of Performance Degradation Hardware Trojan Attacks in Network on Chip." Proc. IEEE Computer Society Annual Symposium on VLSI (ISVLSI), 2018.

[10] J. Sepúlveda, et al. "Reconfigurable security architecture for disrupted protection zones in NoC-based MPSoCs." Proc. IEEE ReCoSoC, 2015.

[11] D. Ancajas, et al. "Fort-nocs: Mitigating the threat of a compromised noc." Proc. ACM Design Automation Conference, 2014.

[12] T. Boraten, et al. "Packet security with path sensitization for NoCs." Proc. IEEE DATE, 2016.

[13] H.M. Wassel, et al. "SurfNoC: a low latency and provably non-interfering approach to secure networks-on-chip." Proc. ACM SIGARCH Computer Architecture News (Vol. 41, No. 3, pp. 583-594).

[14] S.V.R Chittamuru, I. Thakkar, S. Pasricha "SOTERIA: exploiting process variations to enhance hardware security with photonic NoC architectures." Proc. ACM//IEEE Design Automation Conference (DAC) 2018.

[15] S. Skorobogatov, "Physical attacks and tamper resistance," in Introduction to Hardware Security and Trust, Springer Berlin/Heidelberg, 2011.

[16] E. Dubrova, et al. "Secure and efficient LBIST for feedback shift register-based cryptographic systems," Proc. International Test Conf. (ITC), 2014.

[17] P. Kocher, et al., "Differential power analysis." Springer- Verlag, 1999.

[18] E. Dubrova, et al. "Keyed logic BIST for Trojan detection in SoC." Proc. IEEE International Symposium on System-on-Chip (SoC), 2014

[19] M. Oya, "In-situ Trojan authentication for invalidating hardware-Trojan functions." Proc. IEEE ISQED, 2016.

[20] M. Hussain, et al. "EETD: An Energy Efficient Design for Runtime Hardware Trojan Detection in Untrusted Network-on-Chip." Proc IEEE Computer Society Annual Symposium on VLSI (ISVLSI) 2018.

[21] A. Prodromou, et al. "Nocalert: An on-line and real-time fault detection mechanism for network-on-chip architectures." Proc. MICRO 2012.

[22] T. Boraten, et al. "Secure model checkers for Network-on-Chip (NoC) architectures." Proc. IEEE Great Lakes Symposium on VLSI, 2016.

[23] S.F. Mossa, et al. "Self-triggering hardware Trojan: Due to NBTI related aging in 3-D ICs." Integration, the VLSI Journal, Vol: 58, pp:116-124, 2016.

[24] M.K. Papamichael, et al. "CONNECT: re-examining conventional wisdom for designing nocs in the context of FPGAs." Proc. FPGA, 2012.

[25] Vivado HLS tool, Xilinx. https://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html.

[26] S.R. Vangal, et al. "An 80-tile sub-100-w teraflops processor in 65-nm cmos." IEEE Journal of Solid-State Circuits, vol: 43, no:1, pp.29-41, 2008

[27] M. Gebhart, et al. "Running PARSEC 2.1 on M5, Technical Report TR-09-32", UT Austin, Department of Computer Science, October 2009

[28] S. Bhardwaj, et al. "Predictive modeling of the NBTI effect for reliable design." Proc. IEEE Custom Integrated Circuits Conference, 2006.

[29] V. Catania, et al. "Noxim: An open, extensible and cycle-accurate network on chip simulator." Proc. IEEE ASAP, 2015.

[30] J. Hestness, et al. "Netrace: dependency-driven trace-based network-on-chip simulation." Proc. ACM NocArch, 2010.

[31] Cadence, https://www.cadence.com/content/cadence-www/tools/custom-ic-analog-rf-design/layout-design/virtuoso-layout-suite.html