SECTAR: Secure NoC using Trojan Aware Routing

Manju R[‡], Abhijit Das[‡], John Jose[‡] and Prabhat Mishra[*]
[‡]Indian Institute of Technology Guwahati, Assam, India
[*]University of Florida, Gainesville, Florida, USA
{manju18, abhijit.das, johnjose}@iitg.ac.in

***Abstract*—System-on-Chips (SoCs) are designed using different Intellectual Property (IP) blocks from multiple third-party vendors to reduce design cost while meeting aggressive time-to-market constraints. Designing trustworthy SoCs need to address the increasing concerns related to supply-chain security vulnerabilities. Malicious implants on IPs, such as Hardware Trojans (HTs) are one of the significant security threats in designing trustworthy SoCs. It is a major challenge to detect Trojans in complex multi-processor SoCs using conventional pre- and post-silicon validation methodologies. Packet-based Network-on-Chip (NoC) is a widely used solution for on-chip communication between IPs in complex SoCs. The focus of this paper is to enable trusted NoC communication in the presence of potentially untrusted IPs. This paper makes three key contributions. (1) We model an HT in NoC router that activates misrouting of the packets to initiate a denial of service, delay of service, and injection suppression. (2) We propose a dynamic shielding technique that isolates the identified HT infected IP. (3) We present a secure routing algorithm to bypass the HT infected NoC router. Experimental results on HT infected NoC demonstrate that the proposed method reduces effective average packet latency by 38% in real benchmarks and 48% in synthetic traffic patterns. Our method also increases throughput and reduces effective average deflected packet latency by 62% in real benchmarks and 97% in synthetic traffic patterns.**

***Index Terms*—Hardware Trojan, Network-on-Chip Security**

I. Introduction

With the widespread commercialization of safety-critical real-time systems, semiconductor industries have started paying more attention to robust hardware-based security. Due to time-to-market and cost considerations, many products still rely on the supply chain to perform various activities, including design automation of specific components as well as manufacturing of integrated circuits. Functional security of these devices can be compromised due to the involvement of potentially untrusted third-parties during the design cycle [1] [2]. While there are various forms of supply-chain vulnerabilities, malicious implants in circuits, also known as Hardware Trojans (HTs) [3] [4], is one of the major security threats in modern System-on-Chips (SoCs). These HTs can create security vulnerabilities as well as functional inconsistencies in the SoC [1]. Some of the HTs are hard to detect, subtle in their operation and are sophisticated to the extent that they can even bypass the root-of-trust techniques that secure device firmware [2] [5]. Given that SoCs are used in a wide variety of embedded and IoT devices, it is critical to enable trustworthy computing using potentially untrusted components in SoCs.

Packet-based Network-on-Chip (NoC) provides the on-chip communication infrastructure for modern Multi-Processor System-on-Chips (MPSoCs). NoC provides connectivity between a wide variety of components in an MPSoC such as processor cores, GPUs, memories, converters, controllers, I/O, etc. Today's NoCs provide more emphasis on performance, scalability and backward compatibility than security [6] [7]. Due to its positional advantage, NoC is a prime target for attackers to insert HTs. Consequently, NoC routers, being the communication backbone of MPSoCs, becomes most vulnerable to security threats. HT infected NoC routers can lead to denial of service [8], information leakage [9], high jacking [10], unauthorized memory access [11], etc. They directly or indirectly result in bandwidth depletion and performance degradation of the entire system. Detection and mitigation of HTs on NoCs impose unique challenges [12] [13]. One of the popular HTs that exists in an NoC router misroute packets to trigger a DoS attack [14]. A runtime detection algorithm that uses the incoming direction of the packets detects the location of such HT infected routers. However, it assumes that the operating system will provide shielding to ensure protection. Such a hardware-software solution can lead to unacceptable performance overhead, whereas a simple hardware-only approach will give a better action response. Motivated by the impact of such an HT infected NoC router and the limitations in the existing work, we propose an HT threat model with runtime detection as well as mitigation at the hardware level. To the best of our knowledge, there are no prior efforts that consider runtime detection, shielding and bypassing of NoC based Trojans at the same time.

In this paper, we model an HT on an NoC router that misroutes packets in the network and initiates DoS attack on a specific set of processing elements. Furthermore, misrouting packets at times also create injection suppression that propagates across various routers, taking the system to a near halt. To secure NoC from such misrouting HTs, we propose a technique called Trojan Aware Routing (TAR), which consists of three main phases. In the first phase, we deploy a runtime detection mechanism that tracks for routing violation and exposes the HT infected NoC router. After detection, the second phase employs a dynamic shielding mechanism that isolates the HT infected NoC router from the rest of the

This work was partially supported by the NSF grant SaTC-1936040.

978-1-4673-9030-9/20/$31.00 2020 IEEE

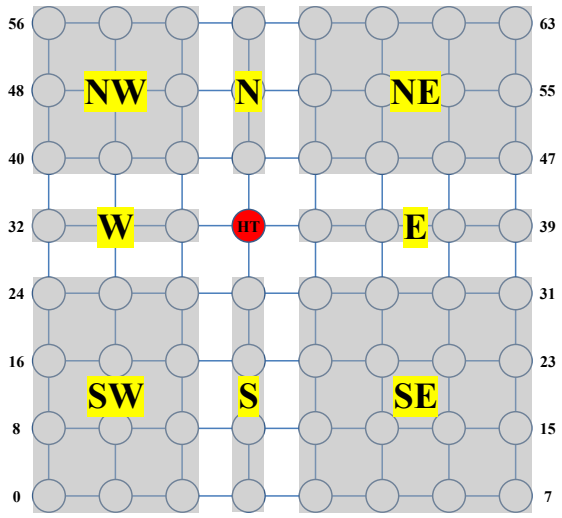


Fig. 1: 8×8 mesh NoC with an HT at router 35

network. With the shielding enabled, the third (final) phase uses a bypass algorithm that route packets in the network isolating the HT infected NoC router. In this paper, we make the following significant contributions:

- We implement packet misrouting on NoC to model an HT that leads to denial of service, delay of service, and injection suppression.
- Our Trojan-aware routing dynamically detects a misrouting HT, shields it and route packets bypassing it.
- We experimentally demonstrate that our approach effectively mitigates DoS and injection suppression.

II. Threat Model

In our threat model, we consider an HT that tampers the routing algorithm employed in NoC routers to enable misrouting. When triggered, the HT maliciously assigns a wrong output port to the head flit of a packet. As a result, all the flits of that packet also get misrouted due to wormhole routing. This can move the packet away from its destination and can cause either denial of service (DoS) or injection suppression or both. DoS is a scenario where a packet gets indefinitely delayed in the path and never reaches its destination. Injection suppression scenario is a by-product of DoS where new flits cannot be injected into the network due to unavailability of router input buffers. Sometimes the packet may reach the destination after few cycles of extra delay. Usually, NoC packets carry cache miss requests, cache miss replies, evicted cache blocks, and coherence messages. An infected NoC router with the proposed HT can misroute these packets and degrade the application-level performance of latency-critical applications. Such type of HTs can be added to an NoC IP at any of the phases of an IC life cycle, including specification phase, design phase, and fabrication phase [3] [15].

In this work, we assume that the proposed HT enters the NoC IP during the pre-silicon stage, either by an attacker having access to the system design or by an untrusted third party EDA tool. An adversary can activate any number of such HTs in the NoC. However, activating multiple HTs can create an unusual variation in energy and power consumption and hence may be easily noticed (detected). To make it hard to get detected, we model the NoC with HT deployed in a single NoC router. The detection is made even harder by assuming that the proposed HT is intermittently malicious and internally triggered [3] [16]. The proposed HT threat model is as follows: An NoC packet P can be represented as:

$$P = \{F^p_{head} \parallel F^p_{body1} \parallel F^p_{body2} \parallel ... \parallel F^p_{bodyn} \parallel F^p_{tail}\} \quad (1)$$

where F^p_i are the flits of packet P such that:

$$F^p_{head} = [\{SRC,\ DEST,\ CTRL\_MSG\}]$$
$$F^p_{body} = [\{CTRL\_MSG\},\ \{Data\}]$$
$$F^p_{tail} = [\{CTRL\_MSG\},\ \{Data\}]$$

Path of packet P from source to destination can be given as:

$$P = \{R_{src},\ \ldots\ R_{k-1},\ R_k,\ R_{k+1},\ \ldots\ R_{dest}\} \quad (2)$$

where R_i denotes router i on the NoC. Let RA_i denote the routing algorithm employed in router R_i. We can infer from Equation (1) and (2) that for a packet P,

$$RA_k(F^p_{head}) = R_{k+1} \quad (3)$$

where for packet P, the routing algorithm employed in router R_k will assign the next router as R_{k+1}.

Let HT denotes our proposed threat model such that

$$HT(RA_k) = RA^*_k \text{ and}$$
$$RA^*_k(F^p_{head}) = R^*_{k+1} \text{ where}$$
$$R^*_{k+1} \neq R_{k+1}$$

Consider an 8×8 mesh NoC shown in Fig. 1. Based on the location of HT (router 35, shown in *red*), we divide the NoC into eight regions: N, E, S, W, NE, SE, SW and NW. When triggered, the impact of HT varies based on the source and destination regions of packets. We categorize the behaviour and impact of the proposed HT threat model into two cases. We explain them using two specific examples.

Case 1: Consider a packet $P1$ with source $S1$ on its way to destination $D1$ reaches router 35 as shown in Fig. 2. Instead of forwarding $P1$ to router 43 as per XY routing, the HT misroutes $P1$ to router 34. $P1$, upon reaching router 34, follows XY routing, and reaches back to router 35. Note that destination $D1$ is at router 59 which is on the same column as that of HT infected router 35. As per XY routing, $P1$ can reach destination $D1$ only through router 35, which is infected. Hence router 35 will always misroute and $P1$ can never reach destination $D1$, leading to a DoS like attack on NoC. From Fig. 1 we can see that source $S1$ is in region E and destination $D1$ is in region N. Thus, inter-region communication of type $E \rightarrow N$ will create a DoS like scenario here. To generalise, for all inter-region communication where the destination router is on the same column as that of HT infected router 35, a DoS attack like scenario will arise. A DoS attack like scenario arises when there is a packet movement between the following

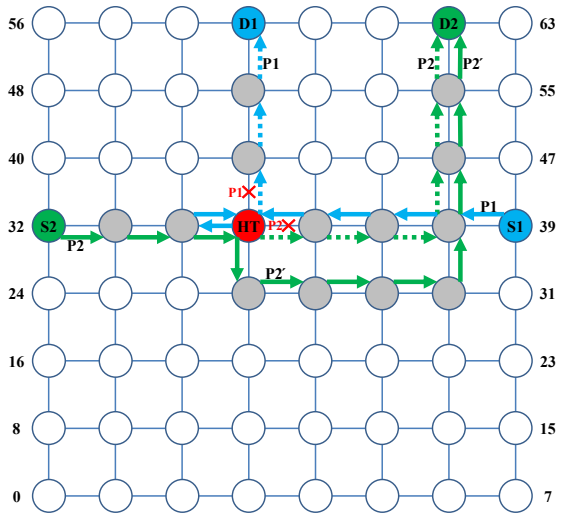


Fig. 2: Illustration of diverse HT impacts

regions: $E \rightarrow N$, $E \rightarrow S$, $W \rightarrow N$, $W \rightarrow S$ $NE \rightarrow S$, $NW \rightarrow S$, $SE \rightarrow N$, $SW \rightarrow N$.

In this illustrative example, packet $P1$ will be trapped in a ping-pong behaviour between router 35 and its neighbor routers 27, 34, and 36 since router 35 will never forward $P1$ to router 43. Packets are buffered in VCs of routers while taking part in routing and arbitration decisions. Prolonged ping-pong of $P1$ leads to VC unavailability in neighboring routers and propagates the effect to others by back-pressure. Eventually, a scenario of injection suppression arises in the entire system. When the traffic is high, unavailability of NoC resources due to the ping-pong effect also leads the system into a deadlock. Fig. 3 shows how the proposed HT threat model creates injection suppression in an 8×8 NoC while running the *uniform_random* synthetic traffic. As injection rate increases, the impact of the proposed HT escalates, and results in more injection suppression and eventually a deadlock.

Case 2: Consider another packet $P2$ in Fig. 2 with source $S2$ on its way to destination $D2$ reaches router 35. Instead of forwarding $P2$ to router 36 (as per XY routing), the activated Trojan at router 35 misroutes $P2$ to router 27. Following XY routing, router 27 now forwards packet $P2$ to router 28. Since the destination $D2$ is not in the same column as that of router 35, packet $P2$ can eventually reach the destination. However, getting misrouted by router 35 delays the arrival of packet $P2$ at destination $D2$. This is a scenario of delay of service attack. From Fig. 1 we can see that source $S2$ is in region W and destination $D2$ is in region NE. Thus inter-region communication of type $W \rightarrow NE$ creates a delay of service. To generalise, a delay of service attack like scenario will arise when there is a communication between the following regions: $E \rightarrow W$, $E \rightarrow NW$, $E \rightarrow SW$ $W \rightarrow E$, $W \rightarrow NE$, $W \rightarrow SE$.

III. Trojan Aware Routing (TAR)

Our proposed TAR technique is employed in every router on NoC. TAR involves three phases: Detection, Shielding, and Bypass. The working of these phases is described as follows:

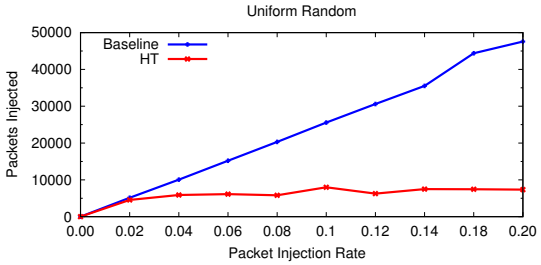


Fig. 3: HT triggered injection suppression

A. Phase 1: Trojan Detection

We use XY routing where a packet travels along the X direction and reach the same column as that of destination. Then, the packet travels along the Y direction to reach the destination. Let P be a packet with source $S(x1, y1)$ and destination $D(x2, y2)$. As per XY routing, when P reaches an intermediate router $R(x, y)$, it will be forwarded along X direction until $(x < x2)$. When P reaches a router where $(x == x2)$, it changes the direction and starts travelling along Y direction until $(y < y2)$. When P reaches a router where $(y == y2)$, it reaches destination $D(x2, y2)$. The XY routing algorithm decides the output port for a packet based on the position of destination router with respect to the current router. The routing algorithm does not consider the input port of the packet and its previous router for its routing decisions. Our proposed HT threat model exploits this feature of the routing algorithm and enables misrouting. Now, even if a packet is misrouted and reaches a router where it should not have reached as per XY routing, the employed routing algorithm will never be able to detect it. The packet will be forwarded to destination without knowing the misrouting that lead the packet to this router.

To identify packet misrouting and HT infected router, we add a detection module, a 1-bit *alert_flag* and a 3-bit *alert_dir* at every NoC router. *alert_flag* is set only if the neighbor is identified as an HT infected router and reset otherwise. *alert_dir* either denotes no direction or the direction where the HT is detected; north, east, south, or west. In the illustrative example shown in Fig. 2, packet $P1$ is forwarded to router 34 because of the misrouting at router 35. With the detection module in place, router 34 knows that $P1$ has entered through east input port from router 35. Analyzing the position of destination $D1$ at router 59 with respect to router 35, the detection module concludes that XY routing is violated and $P1$ is misrouted. Router 34 sets its *alert_flag* and updates *alert_dir* as east since router 35 misrouted packet $P1$ and hence must be an HT infected router. *alert_flag* and *alert_dir* are also used in the subsequent phases of shielding and bypass routing.

B. Phase 2: Dynamic Shielding

Once the HT is detected by one of its neighbors (27, 34, 36, or 43), a dynamic shielding protocol is activated. The router that detects the HT, generates a special *alert flit* to be sent to its neighbors about the detection of the HT. We call such routers as generators. Neighbors upon receiving the *alert flit*

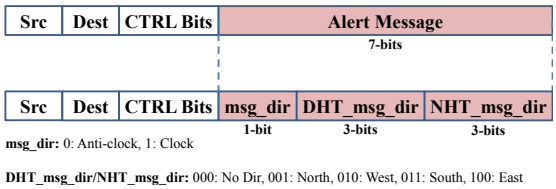


Fig. 4: Structure of an alert_flit

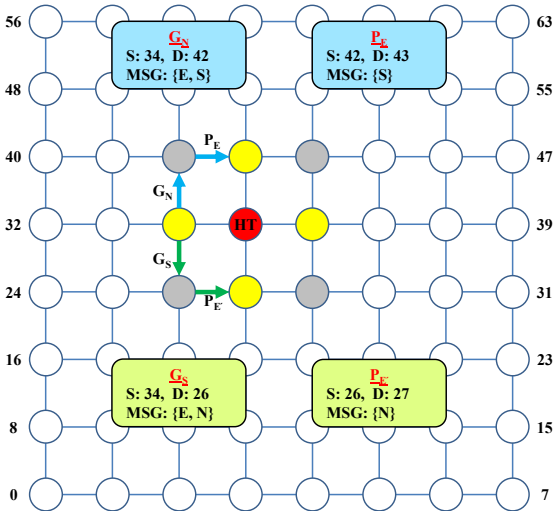


Fig. 5: Working of dynamic shielding in TAR

propagates the message further by creating a *propagation flit*. We call such routers as propagators. The structure of these special flits is very similar to normal flits, as shown in Fig. 4. *Alert flit* contains a 1-bit *msg_dir* indicating the direction an *alert flit* needs to be forwarded by generators. A 3-bit *DHT_alert_dir* indicates the direction an *alert flit* needs to be forwarded by propagators. The alert message also contains a 3-bit *NHT_alert_dir* which indicates the direction where the HT is detected. Fig. 4 presents all the possible values for different fields of the *alert flit*. When the message of HT detection is propagated among all the neighboring routers using *alert* and *propagation flits*, each router accordingly updates its *alert_flag* and *alert_dir*. This results in a shield creation around the HT that successfully isolates the HT infected router from the rest of the network. The third and final phase of TAR uses this shielding to route packets by bypassing the isolated HT infected router.

With an illustrative example shown in Fig. 5, we explain the working of our dynamic shielding phase. From the previous phase of HT detection, let us assume that router 34 has identified router 35 as an HT infected router. *alert_flag* in router 34 is now set to 1 and *alert_dir* as 100 (East). As shown in Fig. 5, router 34 generates two alert flits, G_N and G_S. With an alert message {*msg_dir* = 0, *DHT_alert_dir* = 100, *NHT_alert_dir* = 011}, alert flit G_N is forwarded from router 34 to router 42, where *msg_dir* = 0 indicates G_N to be forwarded in clockwise direction. *DHT_alert_dir* = 100 (East) in G_N indicates that upon reaching router 42, the message needs to be propagated in East direction. Router 42 generates a propagation flit P_E with an alert message {*msg_dir* = 0, *DHT_alert_dir* = 000, *NHT_alert_dir* = 011} to be forwarded

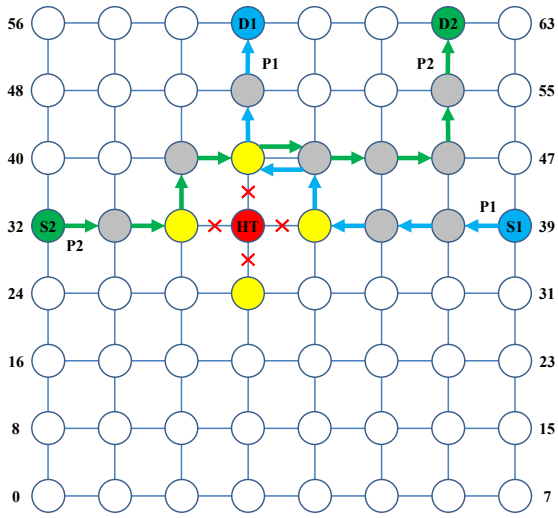


Fig. 6: Working of bypass algorithm in TAR

to router 43. When P_E reaches router 43, *NHT_alert_dir* = 011 (South) indicates that the HT is detected in South direction of router 43; which is router 35. *alert_flag* and *alert_dir* are updated as 1 and south respectively in router 43 which can be a generator for other neighbors. Similarly, G_S and $P_{E'}$ also propagates the message of HT detection to other neighbors. Here, 27, 34, 43, and 36 are generator routers and 26, 42, 44, and 28 are propagation routers. The message propagation continues from both sides until a logical shield is created around the HT infected router. In this example, the shield is completed when *alert_dir* is set for router 27 as north, router 34 as east, router 43 as south, and router 36 as west. After the end of dynamic shielding, the detected HT infected router is isolated from rest of the network.

C. Phase 3: Trojan Bypass

The final phase of TAR implements a bypass routing mechanism, as presented in Algorithm 1. When a packet arrives at a router, bypass mechanism checks the *alert_flag* and *alert_dir* of that router. Only if the *alert_flag* is set and *alert_dir* matches with the desired output port direction of the packet, bypass routing is activated. In all other cases, a packet follows normal XY routing to reach its destination.

We explain the working of Trojan bypass algorithm with an illustration, as shown in Fig. 6. We consider the same example as that in Fig. 2 for the sake of simplicity and continuity. A packet $P1$ with source $S1$ on its way to destination $D1$ reaches router 36. After the completion of shielding in the previous phase, router 36 has its *alert_flag* set and *alert_dir* as west. As per XY routing, the desired output port of packet $P1$ at router 36 is west which matches with the *alert_dir* of router 36. Now the Trojan bypass algorithm initiates and reroutes packet $P1$ away from the HT infected router 35 as presented in Part I of Algorithm 1. Packet $P1$ is rerouted from router 36 to router 44, and Part II of Algorithm 1 is initiated since 44 is a propagation router. Now, packet $P1$ is forwarded from router 44 to router 43, and from there it directly reaches destination $D1$ at router 59.

Algorithm 1: Trojan bypass

Input : Packet header
Output: Output port direction of a flit
Terminology
x_{diff}: x difference between destination & current router.
y_{diff}: y difference between destination & current router.
in_dir: Input port direction of a flit.
out_dir: Output port direction of a flit.
$maxCredit(out\_dir_1, out\_dir_2)$: returns out_dir with more VCs.

```
/*Part I: Mitigation by generator routers */
if alert_flag is SET then
    if x_diff ≠ 0 && y_diff ≠ 0 then
        if alert_dir ≠ EAST then
            if x_diff > 0 && in_dir ≠ EAST then
                out_dir = EAST
        else if alert_dir ≠ WEST then
            if x_diff < 0 && in_dir ≠ WEST then
                out_dir = WEST
        else if alert_dir == EAST || WEST then
            if y_diff < 0 then
                out_dir = SOUTH
            else
                out_dir = NORTH
    else if x_diff == 0 then
        if (y_diff > 0 && alert_dir == NORTH) ||
        (y_diff < 0 && alert_dir == SOUTH) then
            out_dir = maxCredit(EAST, WEST)
    else if y_diff == 0 then
        if (x_diff > 0 && alert_dir == EAST) ||
        (x_diff < 0 && alert_dir == WEST) then
            out_dir = maxCredit(NORTH, SOUTH)
    else if alert_dir ≠ NORTH then
        if y_diff > 0 && in_dir ≠ NORTH then
            out_dir = NORTH
    else if alert_dir ≠ SOUTH then
        if y_diff < 0 && in_dir ≠ SOUTH then
            out_dir = SOUTH
/*Part II: Mitigation by propagation routers */
else if alert_flag is RESET then
    if (x_diff < 0 && in_dir == WEST) ||
    (x_diff > 0 && in_dir == EAST) then
        if y_diff < 0 then
            out_dir = SOUTH
        else
            out_dir = NORTH
    else if x_diff < 0 && in_dir == SOUTH then
        out_dir = WEST
    else if x_diff > 0 && in_dir == NORTH then
        out_dir = EAST
```

Since destination $D1$ is in the same column as that of HT infected router 35, it becomes impossible for $P1$ to reach $D1$ using the conventional approach and resulted in a DoS like scenario. With the Trojan bypass algorithm in place, now $P1$ can reach its destination, thus mitigating the impact of DoS. Since packet like $P1$ is not trapped in the network anymore, our bypass routing also diminishes the possibility of injection suppression. Similarly, packet $P2$ with source $S2$ on its way to destination $D2$ reaches router 34. Instead of forwarding to router 35 which is HT infected, router 34 reroutes $P2$ towards router 42. The Trojan bypass algorithm rerouted packet $P2$ in such a way that it reaches destination $D2$ without any additional delay. Hence, the delay of service scenario created by the proposed HT threat model is mitigated by intelligent bypassing. Please note that router 35 misroutes only those packets that are passing through it. Hence, even after bypassing is activated, the packets whose source/destination is router 35 will continue to come out of/go into router 35, thus not hampering the application executing in the infected core. Due to the nature of runtime detection, when an HT is detected, it might have already misrouted first few flits of some packets while rest of the flits are on the way. Intuitively, it seems that the bypassing algorithm will not allow the rest of the flits to travel to the HT infected router in order to avoid misrouting. However, this situation will not arise since only the head flit takes part in routing and arbitration. Hence, if a head flit is already misrouted before HT detection, all the following flits will go through the same route. After HT detection, when such a misrouted head flit comes out of the HT infected router due to the ping-pong effect, it will never enter the HT again due to the employed bypassing. Hence, even misrouted flits will eventually reach their respective destination.

Rerouting packets using the bypass algorithm violates normal XY routing and creates a possibility for network deadlock. To ensure deadlock prevention, we employ the concept of intermediate destination [17]. When packet $P2$ is rerouted from router 34 to router 42, it starts travelling in the Y direction. However, when it travels from router 42 to router 43, $P2$ violates XY routing, since turning X from Y direction is prohibited. Using the concept of intermediate destination [17], router 42 is made the new destination for packet $P2$. Now, after getting rerouted from router 34, packet $P2$ reaches router 42 and gets ejected into its local output port, since 42 is the new destination. Only after router 42 finds out that $P2$ is meant for destination $D2$ at router 62, it re-injects $P2$ as a new packet destined for $D2$. Packet $P2$ now follows normal XY routing like any other packet to reach the destination. The ejection of packet $P2$ and re-injection as a new packet from the intermediate destination 42 makes sure that XY routing is not violated thus eliminating deadlock.

IV. Experiments

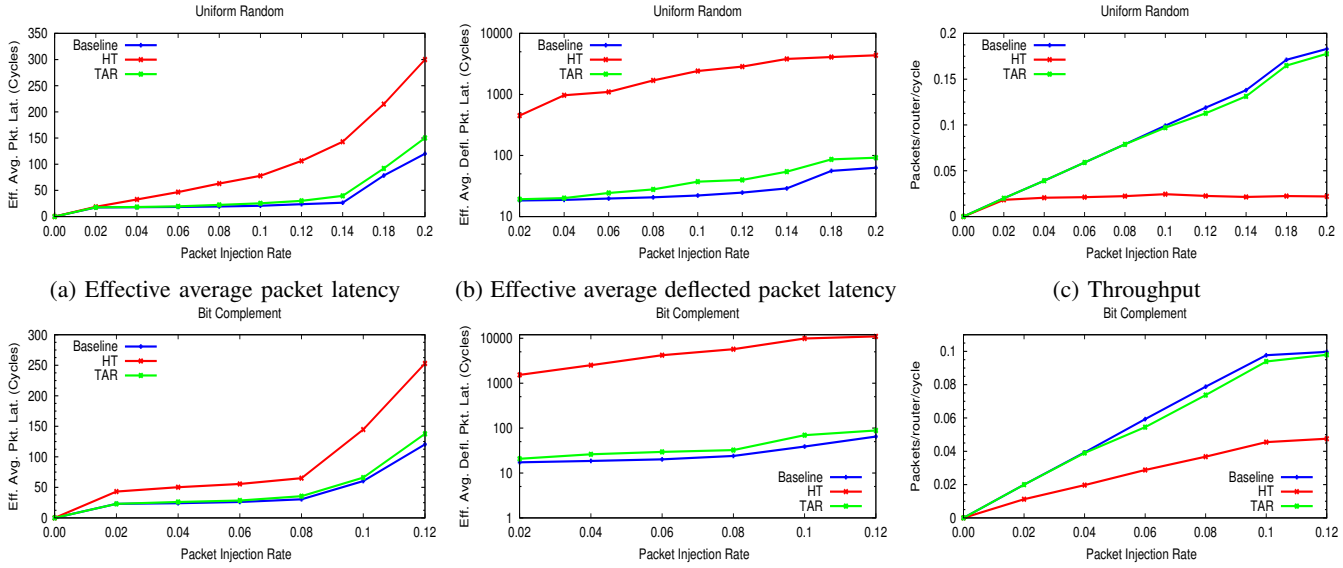
We evaluate the performance of TAR using effective average packet latency, effective average deflected packet latency, throughput, and injection suppression avoidance.

A. Experimental Setup and Workloads

We implement the baseline system (normal NoC without any HT), NoC with an HT infected router, as well as the proposed TAR using the event-driven simulator, gem5 [19]. We use the garnet framework in gem5's ruby memory model for implementing the NoC. Our baseline system is a traditional 8×8 2D mesh NoC with 5 VCs per input port and uses a 128-bit flit channel for inter-router communication using XY routing. To model the Trojan, we modify the routing module such that there exists a single HT router in the NoC at any given point in time. The shielding approach and bypass

TABLE I: Workload categorization using SPEC CPU 2006 benchmark mixes.

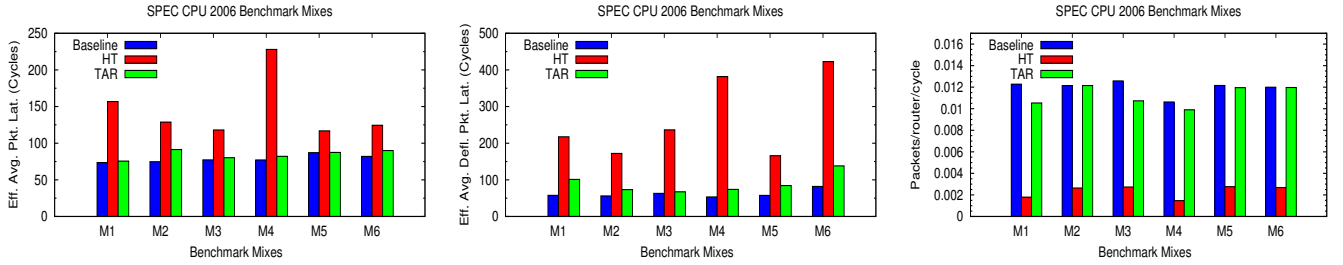
Workload	Workload Pattern: name of benchmark (number of instances)							Workload Characteristics
M1	leslie3d (16)		lbm (16)	GemsFDTD (16)		mcf (16)		100% High MPKI
M2	sjeng (16)		bzip2 (16)	omnetpp (16)		sphnix (16)		100% Low MPKI
M3	soplex (32)			astar (32)				100% Medium MPKI
M4	leslie3d (8)	bzip2 (8)	omnetpp (16)	sjeng(8)	GemsFDTD(8)	lbm (8)	mcf (8)	50% High MPKI, 50% Low MPKI
M5	sjeng (8)	bzip2 (8)	sphnix (16)	soplex (16)		astar (16)		50% Low MPKI, 50% Medium MPKI
M6	leslie3d (8)	bzip2 (8)	omnetpp (16)	sjeng(8)	soplex (8)	astar (16)		50% High MPKI, 50% Low MPKI



(a) Effective average packet latency (b) Effective average deflected packet latency (c) Throughput

(d) Effective average packet latency (e) Effective average deflected packet latency (f) Throughput

Fig. 7: Performance analysis with synthetic traffic patterns. For latency plots given in (a), (b), (d), & (e), lower the line better and for throughput plots in (c) & (f), higher the line the better.



(a) Effective average packet latency (b) Effective deflected packet latency (c) Throughput

Fig. 8: Performance analysis with SPEC CPU 2006 benchmark mixes. TAR reduces effective average packet latency and deflected packet latency over HT, and maintains a comparable throughput as that of baseline. For latency plots (a) & (b), lower the bar the better and for throughput plot (c), higher the bar the better.

algorithm is done in garnet with all micro-architectural and functional specifications, as discussed in Section 3.

To evaluate the performance and NoC-specific parameters, we run standard synthetic traffic patterns *uniform_random,* and *bit_complement* by varying the injection rate. We also analyze the proposed system using real application workloads consisting of SPEC CPU 2006 benchmarks. We model a 64-tile TCMP, each with a simple CPU core and a 32 KB, 4-way set associative, 64-byte block, private L1 cache. Each tile has a 256 KB, 16-way associative, 64-byte block, shared L2 cache. L2 cache sets are mapped to various tiles using the SNUCA technique. We assign a SPEC CPU 2006 benchmark application to each of the 64 core to model a TCMP simulation framework. L1 cache misses trigger NoC packets, which get routed from the source tile to the destination tile to which the corresponding L2 cache sets are mapped. Similarly, the reply packets also travel through the NoC. We use a 1-flit request packet and 5-flit reply packets.

We study the performance of the NoC under different network loads by grouping the SPEC CPU 2006 benchmarks based on their Misses Per Kilo Instructions (MPKIs). We classify the benchmarks into High MPKI (greater than 40), Medium MPKI (less than 40 but greater than 20), and Low MPKI (less than 20). Here we use *leslie3d*, *lbm*, *GemsFDTD*, and *mcf* under High MPKI, *soplex* and *astar* under Medium MPKI, and *sjeng*, *bzip2*, *omnetpp*, and *sphinx* under Low

MPKI. With the help of this classification, we form six categories of workloads; M1, M2, M3, M4, M5, and M6, each having 64 benchmark instances, as given in Table I.

B. Effective Average Packet Latency

To analyse the effect in packet latency with HT triggering and mitigation, we use *average packet latency* (APL), which is defined as the number of cycles required for a packet to reach its destination. As the average packet latency on an HT infected NoC shows inconsistent values at higher injection rates due to packet loss and injection suppression, we apply a more realistic metric *effective average packet latency* (EAPL) [16] which is defined as follows:

$$EAPL = APL * \frac{Packets\_Ejected_{withoutHT}}{Packets\_Ejected_{withHT}} \quad (4)$$

Fig. 7a and 7d shows the effective average packet latency using *uniform_random* and *bit_complement* traffic patterns. As expected, when the injection rate increases, packet latency also increases in the baseline, HT infected NoC, and TAR. But we observe that the rate of latency increase in the case of HT infected NoC is significantly higher than the other two. This is due to the deflection of packets by HT router and subsequent DoS as well as the delay of service scenario. However, TAR reduces effective average packet latency significantly compared to HT infected NoC. Since TAR uses HT bypassing to secure communication, the majority of packets that are supposed to travel through the HT have to take an extra few hops to reach the destination. Thus, we note an increase in effective average packet latency for synthetic traffic patterns by 16% compared to baseline. We also analyze the effective average packet latency using real workloads as shown in Fig. 8a. Across all benchmark mixes, HT triggering increases packet latency by an average of 87% over the baseline. TAR exhibits a reduction in the effective average packet latency by 38% with respect to HT infected NoC, but a minor 7% increase with respect to baseline due to bypass routing.

C. Effective Average Deflected Packet Latency

Average deflected packet latency (ADPL) is defined as the average packet latency of those packets which are meant to travel through the HT infected router. Consider a router R that is going to be HT infected. To calculate the ADPL in the baseline, we consider the packets that are passing through R. In the case of an HT infected NoC, ADPL is calculated for only those packets that suffer Trojan-induced deflection at R. For calculation of ADPL in TAR, we consider the packets that are deflected by the neighbors of R while applying the bypass algorithm. Similar to *effective average packet latency*, to get meaningful latency values, we use *effective average deflected packet latency* (EADPL) which is defined as follows:

$$EADPL = ADPL * \frac{Deflected\_Packets\_Ejected_{withoutHT}}{Deflected\_Packets\_Ejected_{withHT}} \quad (5)$$

Fig. 7b and Fig. 7e shows the effective average deflected packet latency using *uniform_random* and *bit_complement* traffic patterns, respectively. We observe that as the injection rate increases, effective average deflected packet latency increases significantly on the HT infected NoC. With HT triggering, few packets deflected by HT enter into a ping-pong state between its neighbors. Eventually, some of these packets move out of this state and reach the destination. This leads to an increase in the deflected packet latency. As the injection rate increases, this ping-pong effect reduces the available router buffers, which, in turn, increases the deflected packet latency. We observe that TAR reduces the effective average deflected latency significantly with the help of a bypassing algorithm. While analyzing the results, we observe that TAR reduces average deflected packet latency by 97% compared to HT infected NoC. However, due to the bypass-induced deflection of packets, TAR shows an average of 38% increase in the deflected packet latency over the baseline.

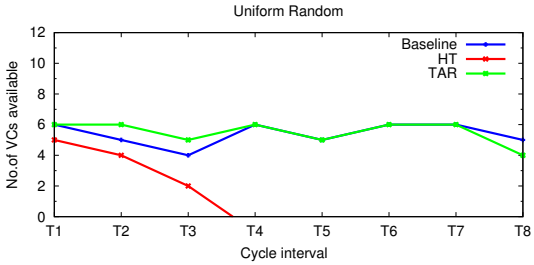


Fig. 9: Virtual channel (VC) availability

We also analyze effective average deflected packet latency in real application workloads using SPEC CPU 2006 benchmark mixes, as shown in Fig. 8b. Experimental results show that across all benchmark mixes, TAR reduces the average deflected packet latency by 62% over HT infected NoC, and increases by 40% over the baseline.

D. Throughput

We also analyze the impact of HT on the throughput of NoC. Throughput is defined as the number of packets that have reached its destination per router per clock cycle. In baseline and TAR, almost all injected packets are ejected after passing through the NoC, whereas in HT infected NoC this delivery rate is less than 75%. Here few packets are stuck in the routers due to the ping-pong effect. This leads to a lack of free VC buffers in neighboring routers and can block new packet injections. This injection suppression together with ping-pong effect reduces the throughput. Our analysis (Fig. 7c and Fig. 7f) shows the difference in throughput across various techniques. In the case of real benchmark simulations (Fig. 8c), across various mixes, HT infected NoC receives an average of 80% fewer packets compared to baseline. However, TAR suffers only 6% throughput reduction over the baseline.

E. Injection Suppression Avoidance

Due to the ping-pong effect, the number of packets processed around the HT infected router is very high. This can block the router VCs of HT, its neighbors and subsequent back pressure leading to injection suppression, as shown in

Fig. 3. We study the average number of input VCs available on the NoC router during continuous time intervals T1 to T8, while simulating *uniform_random* traffic at pre-saturation load and the results are given in Fig. 9. We observe that as the simulation progresses, the impact of HT results in a fewer number of input VCs being available in the routers. When the simulation reaches close to T4, input VC availability becomes zero, which indicates the injection suppression in the whole network. TAR ensures that none of the packets is under DoS attack, and the packets are deflected by its one-hop neighbor with the help of our shielding approach. This keeps the input VC availability as close as possible to the baseline, which prevents injection suppression in the network.

F. Overhead Analysis

Timing Overhead: We implement the standard 3-stage pipelined input buffered router where the stages are (1) buffer write and route computation, (2) VC allocation and switch allocation, and (3) switch traversal. The detection module used in TAR works in parallel with the route computation stage to identify whether the previous router is HT infected. Our dynamic shielding phase is completely independent and works in parallel with the normal router operation. Trojan bypass routing is an additional feature in the existing XY routing algorithm which works in the route computation stage. Since none of the phases of TAR execution lies in the critical path of the router pipeline, we confirm that TAR enabled NoC can function at the same operating frequency.

Hardware Overhead: An additional circuitry is used for the detection and mitigation of HT. 1-bit *alert_flag* and 3-bit *alert_dir* used in each NoC router incurs a storage overhead of 4-bits per router and only 32B (4-bits $\times$ 64-core) for the entire system. We use DSENT [18] to evaluate the area and power of 8$\times$8 2D mesh NoC with TAR. In DSENT, we use 22nm processor technology at 1GHz operating frequency. The addition of a detection module and alert flit generator incur a negligible area overhead of 2.78% and a leakage power overhead of 3% compared to the baseline router.

V. Related Work

Several survey articles [2] are published about threats and challenges associated with securing hardware systems. Researchers in the hardware security domain are adopting new technologies for securing NoC based MPSoCs [19]. The data protection technology [11], which is suitable for dynamic and reconfigurable systems, ensures secure memory access in NoCs. Li et. al. [3] discusses various on-chip and off-chip monitoring techniques for HT attacks that could affect the behavior and performance of ICs. Travis et. al. [12] presents an HT model that generates DoS attack by inspecting the links in NoC. The system makes use of the vulnerabilities of the error correction code to cause the DoS attack. It is configured with a switch-to-switch mitigation technique to obfuscate the data in the packet. Three-layer protection mechanism [8], which uses data scrambling, data integrity protection, and node obfuscation technique, can be used to prevent side-channel attacks by HTs located in NoC routers. HT models can also target vital fields in an NoC packet. A pre-planned shuffling pattern can impede these attacks in runtime [20]. There exist HTs that infect IP to launch attacks such as DoS, flooding to waste bandwidth, and high communication latency, which can result in network saturation [21]. These can be detected and localized by monitoring packet arrival curves.

VI. Conclusion

NoC technology gained popularity in MPSoCs due to its ability to separate transport, transaction, and physical layers. The security of NoC routers is vital. An HT infected NoC router can deteriorate the performance of applications running in the system. In this work, we model an HT that performs misrouting and lead to DoS, delay of service, and injection suppression in the network. We proposed a Trojan-aware routing that can effectively detect and bypass Trojan infected components to enable trusted communication in the presence of untrusted components. Experimental results show that TAR mitigates such HT attacks with graceful degradation in system performance. The proposed system improves throughput and shows a substantial reduction in average packet latency and deflected packet latency compared to HT infected NoC.

References

[1] F. Farahmandi *et al.*, *System-on-Chip Security: Validation and Verification*. Springer, 2019.

[2] N. Potlapally, "Hardware security in practice: Challenges and opportunities," in *HOST*, 2011.

[3] H. Li *et al.*, "A survey of hardware Trojan threat and defense," *INTEGRATION*, 2016.

[4] S. Bhunia *et al.*, "Hardware Trojan Attacks: Threat Analysis and Countermeasures," *JPROC*, 2014.

[5] S. Malik and P. Subramanyan, "Specification and modeling for systems-on-chip security verification," in *DAC*, 2016.

[6] W. J. Dally and B. P. Towels, *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2004.

[7] A comparison of Network-on-Chip and Busses. [Online]. Available: https://www.design-reuse.com/articles/10496/a-comparison-of-network-on-chip-and-busses.html

[8] D. Fang *et al.*, "Robustness Analysis of Mesh-based NoC Architecture under Flooding-Based Denial of Service Attacks," in *NAS*, 2013.

[9] D. M. Ancajas *et al.*, "Fort-NoCs: Mitigating the Threat of a Compromised NoC," in *DAC*, 2014.

[10] J. Coburn *et al.*, "SECA: Security-Enhanced Communication Architecture," in *CASES*, 2005.

[11] L. Fiorin *et al.*, "Secure Memory Accesses on NoC," *IEEE TC*, 2008.

[12] T. Boraten and A. K. Kodi, "Mitigation of Hardware Trojan based Denial-of-Service attack for secure NoCs," *JPDC*, 2018.

[13] J. Rajesh *et al.*, "Runtime Detection of a Bandwidth Denial Attack from a Rogue Network-on-Chip," in *NOCS*, 2015.

[14] L. Daoud and N. Rafla, "Routing Aware and Runtime Detection for Infected Network-on-Chip Routers," in *MWSCAS*, 2018.

[15] C. Rooney *et al.*, "Creation and Detection of Hardware Trojans Using Non-Invasive Off-The-Shelf Technologies," *Electronics*, 2018.

[16] J. Frey and Q. Yu, "A hardened network-on-chip design using runtime hardware Trojan mitigation methods," *INTEGRATION*, 2017.

[17] A. Das *et al.*, "Critical Packet Prioritisation by Slack-Aware Re-routing in On-Chip Networks," in *NOCS*, 2018.

[18] C. Sun *et al.*, "A Tool Connecting Emerging Photonics with Electronics for Opto-Electronic Networks-on-Chip Modeling," in *NOCS*, 2012.

[19] M. Hussain and H. Guo, "Packet Leak Detection on Hardware-Trojan Infected NoCs for MPSoC Systems," in *ICCSP*, 2017.

[20] K. Manoj *et al.*, "Run Time Mitigation of Performance Degradation Hardware Trojan Attacks in Network on Chip," in *ISVLSI*, 2018.

[21] S. Charles *et al.*, "Real-time Detection and Localization of DoS Attacks in NoC based SoCs," in *DATE*, 2019.