Hardware Trojan Mitigation for Securing Network-on-Chip Communication Against Packet Routing Attacks

Manju R

Hardware Trojan Mitigation for Securing Network-on-Chip Communication Against Packet Routing Attacks

Thesis submitted in partial fulfilment
of the requirements for the degree of

Doctor of Philosophy

in

COMPUTER SCIENCE AND ENGINEERING

by

Manju R

Under the supervision of

John Jose

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI

September 2024

To my Mother

- your unwavering strength drove me forward, making this dream possible.

DECLARATION

I hereby certify that

a. The work contained in this thesis is original and has been done by myself under the general supervision of my supervisor.

b. The work has not been submitted to any other institute for any degree or diploma.

c. Whenever I have used materials (data, theoretical analysis, results) from other sources, I have given due credit by citing them in the text of the thesis and giving their details in the references. Elaborate sentences used verbatim from published work have been clearly identified and quoted.

d. No part of this thesis can be considered plagiarism to the best of my knowledge and understanding, and I take complete responsibility if any complaint arises.

Date : **13 / 09 / 2024** **Manju R**

Place: Guwahati, India

भारतीय प्रौद्योगिकी संस्थान गुवाहाटी
गुवाहाटी - 781039
Indian Institute of Technology Guwahati
Guwahati - 781039

Dr John Jose
Associate Professor

: +91 361 258 3256
: +91 361 269 2787
: johnjose@iitg.ac.in

Department of Computer Science and Engineering

THESIS CERTIFICATE

This is to certify that the thesis entitled **"Hardware Trojan Mitigation for Securing Network-on-Chip Communication Against Packet Routing Attacks"** being submitted by **Manju R** to the Department of Computer Science and Engineering, Indian Institute of Technology Guwahati, is a record of bonafide research work carried out by her under my supervision and is worthy of consideration for the award of the degree of Doctor of Philosophy of the Institute.

To the best of my knowledge, no part of the work reported in this thesis has been presented for the award of any degree at any other institution.

Date : **13 / 09 / 2024**

Place: Guwahati, India

Dr. John Jose
(Thesis Supervisor)

ACKNOWLEDGMENTS

My experience at IIT Guwahati has been incredibly transformative, both academically and personally. The campus has played a vital role in shaping me in ways that are difficult to fully express. I remember feeling uncertain on the day I first arrived, questioning my capabilities as a researcher. Moving to campus with my mother and my 5-year-old daughter, especially without immediate accommodation, presented its own set of challenges. However, amidst these difficulties, I encountered many individuals whose constant encouragement helped me navigate through tough situations. The challenges I faced along the way have only served to make me stronger, both personally and professionally.

First and foremost, I would like to express my sincere gratitude to Dr. John Jose, my Ph.D. supervisor, for allowing me the opportunity to work with him. His lectures on Computer Architecture were invaluable throughout my research, and his unwavering guidance and assistance were crucial in helping me overcome the challenges presented by my work. Our conversations, his wise advices, and his consistent encouragement greatly contributed to my development.

I am also deeply indebted to my Doctoral Committee members, Prof. Tamarapalli Venkatesh, Dr. Moumita Patra, and Prof. Ashok Singh Sairam, for their invaluable feedback and guidance regarding my thesis work. I would like to sincerely thank all the professors of the Department of Computer Science and Engineering for their persistent support, especially Prof. G. Sajith, Prof. Arnab Sarkar, and Dr. Benny George K, whose courses significantly expanded my knowledge base.

I am immensely thankful to my mentor, Prof. Prabhat Mishra, whose guidance steered me towards my research area and fuelled my motivation to explore deeper. His unwavering commitment to research and invaluable insights played a vital role in the publication of my first paper.

A comfortable stay on campus is incomplete without supportive lab members. I extend my heartfelt thanks to all the members of the Multi-Core ARchitecture and Systems (MARS) Lab, especially Abhijit, Dipika, Sivakumar, Manjari, and Amit, for their immense support in every possible way. Abhijit, in particular, has been both a close friend and a mentor, providing unwavering support throughout my journey. His valuable input in the early stages of my research was particularly beneficial, and his support as a true friend has been invaluable.

Sivakumar has always been there for me as a brother since the day I joined IITG. His support has been invaluable, and I am deeply grateful for it. Dipika, your friendship has been a source of strength, and I thank you for your unwavering support. Amit, being one of my good companions, has also been a significant source of support. Abeer and Syam, despite our limited interactions, thank you for the invaluable support you provided during my visit to IITG for my synopsis seminar.

My time at IIT wouldn't have been the same without my dear friends Debanjan and Nilotpal. From comprehensive preparations to late-night discussions, they have been there through it all. Our shared memories of Holi celebrations and Friendship Day remain cherished. Thank you for making my campus life enjoyable and reminiscent of the good old college days.

I am immensely grateful for the unwavering mental and moral support from Prof. Amal Dev Parakkat (Associate Professor, Institut Polytechnique de Paris), Prof. Jalaja M J (former Principal, GEC Idukki), Dr. Sangeetha Jose, and Asha Ali (my colleagues from GEC Idukki), all of whom played integral roles in my journey towards completing my thesis. Special gratitude is extended to my mentor Prof. Balu John (Retd. Head of the Dept, CSE, CET) and my dear friend and colleague, Anoop S K M for being my pillars of support as always.

The final thesis completion phase is always challenging. Completing a thesis while working in an institution is a big responsibility. I must mention my deepest gratitude to my friend and head of the department, Information Technology, GEC Barton Hill, Dr. Haripriya A. P, for providing me with the support and freedom to do my work. Special mention to my dearest colleague Prof. Suryapriya S and Sreehari for being with me through thick and thin in all phases of my thesis completion.

None of this would have been possible without the unwavering support of my family. To my mother, Vasanthamma, who courageously accompanied me to this new place, I am eternally grateful for your unconditional support. And to my beloved daughter, Neelanjana, your encouragement, the way you handled so many situations despite my absence, and your special gifts to cheer my days have been my greatest source of strength.

Last but not the least, I extend my gratitude to my husband, Sandeep, for his unwavering belief in me and for encouraging me to pursue my passion. In conclusion, I am profoundly grateful to each and every individual who has contributed to my journey at IIT Guwahati. Your support and encouragement have been invaluable, shaping me into the person I am today.

Guwahati, September 2024 **Manju R**

ABSTRACT

As Tiled-Chip Multi-core Processors (TCMPs) gain widespread adoption in various domains such as automotive systems, IoT devices, and consumer electronics, ensuring the security and integrity of these systems against hardware-level attacks is of utmost importance. Among the potential threats, Hardware Trojans (HTs) pose a significant risk to the reliable operation of TCMPs, particularly in Network-on-Chip (NoC) interconnects. As NoC serves as the backbone for all inter-core communication, it becomes susceptible to many attacks like information leakage and bandwidth denial. This thesis explores the impact of HT attacks on NoC architecture and proposes novel detection and mitigation strategies to fortify TCMPs against such threats. The thesis covers the mitigation strategies against three prominent classes of HT-induced Denial of Service (DoS) attacks: Packet misrouting attack, Packet looping attack, and Packet duplication attack. For each attack, techniques are proposed to detect, localize, and neutralize HT-induced threats within the NoC architecture.

The thesis begins with a comprehensive survey of existing literature on HT attacks in TCMPs, outlining various attack models and their potential impacts on system performance and security. The primary contribution of the thesis focuses on mitigating the HTs that deploy packet misrouting within NoC routers, which results in DoS attacks and injection suppression. A dynamic shielding technique is introduced to isolate HT-infected NoC routers, coupled with a secure routing algorithm to bypass such compromised NoC routers. Meanwhile, our second contribution addresses HT-induced packet looping attacks, where packet delay is selectively induced in the NoC routers through the manipulation of the output path selection strategy used in the routing algorithm. A security wrapper module, along with path monitoring, is proposed to mitigate the impact of HTs with minimal performance overhead, maintaining comparable packet latency to baseline architectures. In our third contribution, we target HT-induced packet duplication attacks, where HT is inserted in the network interface that facilitates unauthorized packet duplication, thus affecting the system's performance. To mitigate the HT effect, a Packet Status Holding Register is integrated within the network interface, which blocks packet duplication with minimal hardware overhead. Experimental results on real benchmarks demonstrate significant improvements in packet latency and throughput.

Table of Contents

List of Figures

Page

List of Tables

Page

List of Acronyms

Acronym	Expansion
IIoT	Industrial Internet-of-Things
SoC	System on Chip
MPSoC	MultiProcessor System on Chip
ASIC	Application Specific Integrated Circuit
FPGA	Field Programmable Gate Array
ISA	Instruction Set Architecture
IC	Integrated Circuit
IP	Intellectual Property
QoS	Quality of Service
TCMP	Tiled Chip Multicore Processor
CAD	Computer Aided Design
NoC	Network on Chip
HT	Hardware Trojan
DoS	Denial of Service
LLC	Last Level Cache
RTL	Register Transfer Level
VHDL	Very High-Speed Integrated Circuit Hardware Description Language
EDA	Electronic Data Automation
VC	Virtual Channel
SPEC	Standard Performance Evaluation Corporation
MPKI	Misses Per Kilo Instruction

CHAPTER

1

Introduction

The digital transformation has facilitated the seamless integration of modern Industrial Internet-of-Things (IIoT) platforms with machines and cyber-physical systems. These platforms heavily depend on sophisticated System-on-Chips (SoCs) within process control systems, utilizing Application Specific Integrated Circuit (ASIC) technology evolving into SoCs to meet contemporary power, performance, and area demands. Figure 1.1 illustrates the structure of an SoC, where all functional blocks, including microprocessors and Digital Signal Processors (DSP), are integrated into a monolithic Integrated Circuit (IC). SoCs typically employ a traditional hierarchical bus or crossbar approach for on-chip communication. The rise in consumer electronics' demand has led to the incorporation of numerous Intellectual Property (IP) blocks into SoCs, resulting in the emergence of Multiprocessor SoCs (MPSoCs) [1]. As applications became more compute-intensive, multimedia systems, video surveillance systems, and automotive vision systems turned to MPSoCs for their high throughput and parallel processing capabilities.

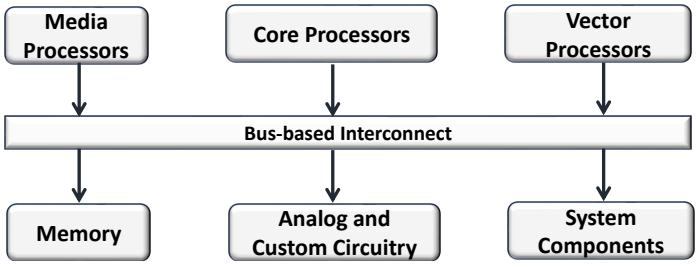


Figure 1.1: System-on-Chip architecture

However, the traditional bus-based/crossbar interconnects used in MPSoCs faced challenges such as routing congestion and low operation frequencies, impacting the quality of service (QoS).

Eventually, the increased complexity of MPSoCs, driven by smaller critical dimensions, intricate power management, and the need for faster interconnect IP speeds, prompted the adoption of advanced MPSoCs in various market verticals. Accordingly, there has been a shift towards Tiled-Chip Multi-core Processors (TCMPs), where multiple processing cores are arranged in a tiled manner, with each core consisting of one or two levels of cache memory modules and interconnected through a lossless packet-based interconnects known as Network-on-Chip (NoC) [2][3]. The incorporation of packet-based communication in NoC enhances bandwidth utilization significantly.

1.1 Multiprocessor System-on-Chip Security

With the increase in the demand for high-performing MPSoCs in safety-critical systems and consumer devices, the IC fabrication units incur considerable costs to update their technology to reduce the gap between manufacturing and synthesis technology. This has led to the outsourcing of the fabrication of ICs, where modern MPSoCs started using commercial off-the-shelf components manufactured by untrusted third-party vendors. Consequently, the long and globally distributed supply chain of hardware IPs makes SoC design increasingly vulnerable to diverse trust/integrity issues [4]. Research shows that supply-chain attacks can happen in any of the phases of IC supply-chain like design, fabrication, and test [5][6][7][8]. Some potential threats in SoC design are EDA tool vulnerabilities, untrusted vendor or foundry, IC subversion, reverse engineering, counterfeit IC, IP piracy, etc [5]. Even complex TCMPs built with third-party NoCs create more vulnerabilities due to their emphasis on performance and backward compatibility [9]. For example, to achieve higher throughput or lower latency, designers often prioritize optimizations that weaken security mechanisms or introduce vulnerabilities. Consequently, chip suppliers rely on application constraints and methodologies to implement proper security policies [6][8] to counteract the hardware attacks.

1.2 Hardware Trojan Attacks in Tiled Chip Multicore Processors

Hardware-oriented attacks on TCMPs often involve exploiting vulnerabilities within the ICs, which can compromise the system's security, integrity, and functionality [10]. This type of attack capitalizes on various hardware information, including power consumption, cryptographic keys, memory access patterns, etc [4]. Some of the potential hardware attacks are cache-based channel attacks, power-analysis attacks, and NoC vulnerabilities. While cache-based attacks like prime+probe attacks [11] exploit the shared caches to monitor cache access patterns, flush+reload [12] attacks observe memory access patterns to deduce information about parallel tasks on different cores. On the other hand,

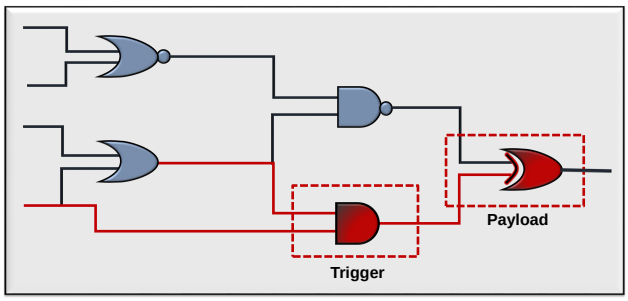


Figure 1.2: An example hardware Trojan with trigger and payload (red gates). The Trojan is activated when all nodes within the circuit connected to the AND gate inputs output a logical value of 1, and the payload inverts the original output to launch an attack on the circuit or system.

power analysis attacks involve monitoring power consumption patterns of individual cores to gain insights into cryptographic algorithms or other sensitive computations [13].

Another major area of attack lies in the vulnerabilities associated with the NoC. For example, traffic analysis attacks targeting the NoC can expose communication patterns between cores, while injection attacks may encompass scenarios where malicious nodes disrupt communication or manipulate messages within the network. One of the significant hardware-level security compromises in TCMPs is the insertion of malicious circuits called Hardware Trojans (HTs) [14], which can alter the functionality of the system. As NoC acts as the backbone for all the on-chip communication, it is often as considered as a primary target for HT attacks [2][9][15][16][17]. An HT typically consists of two parts: a Trigger and a Payload. An example of an HT is shown in Figure 1.2, where the trigger and the payload comprising of an AND and an XOR logic gates are added to the original 4-input circuit. The trigger is usually created using one or more extremely subtle events, such as rare inputs, signals, or transitions. Once triggered, the payload launches an attack, like information leakage, data corruption, Denial-of-Service (DoS), performance degradation, etc. Here, the AND gate serves as the trigger component of the HT. The inputs of this AND gate are sourced from a subset of uncommon nodes within the circuit. The payload of the HT is implemented using an XOR gate. The Trojan is activated when all nodes within the circuit connected to the AND gate inputs output a logical value of 1. Subsequently, the Trojan inverts the logical value at the payload node.

With NoC's wide popularity in modern TCMPs, which are often used in smart TVs, vision systems, and automotive TCMPs [2][18], any HT embedded in NoC can create performance degradation in the system. Through this thesis, we try to draw attention to vulnerabilities associated with the NoC IP and possible ways to counteract such attacks. To understand the architecture of NoC IP used in today's TCMPs, we explore the NoC architecture model published by Arteris for their FlexNoC NoC IP [2]. Figure 1.3 shows the block diagram of Arteris FlexNoC resilience

package NoC IP. FlexNoC uses functional safety-critical NoC for the parts where critical core communication happens. Additionally, the same package has NoC without any safety goals, which handles the communications associated with the input/output system. Through this thesis work, we present the possibility of HT attacks on the NoC routers residing in the non-safety zone of an IP package, which can cause the system to fail to meet performance specifications.

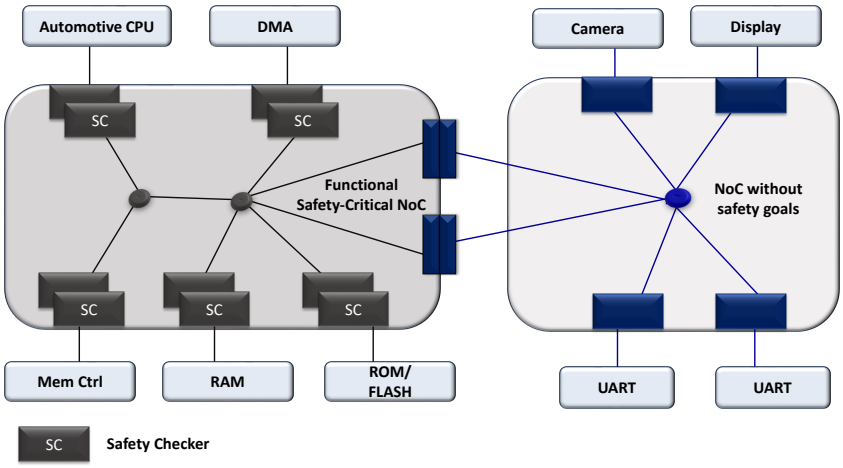


Figure 1.3: FlexNoC resilience package IP

Conclusively, the pervasive threats posed by HTs in NoC design demand a proactive and comprehensive approach to ensure the security and integrity of modern TCMPs. As technology advances and the complexity of integrated circuits grows, the challenges associated with detecting and mitigating HTs become more pronounced. Researchers have explored techniques like reverse chip engineering [19] to counteract the performance degradation of systems due to HTs. These techniques make use of additional hardware with intelligent trade-offs in terms of area and power overheads. The evolving landscape of cyber threats thus requires continuous research and innovation to develop more robust methodologies for safeguarding NoC designs. By exploring novel detection techniques, enhancing the efficiency of various validation methods, and devising strategies to counteract sophisticated HT attacks, researchers can contribute significantly to the resilience of future TCMPs. This motivated us to explore various possibilities of HT attacks in NoC and methods to fortify the foundations of NoC security. The insights gained from such research will not only protect TCMPs but also fortify the very fabric of our digital infrastructure.

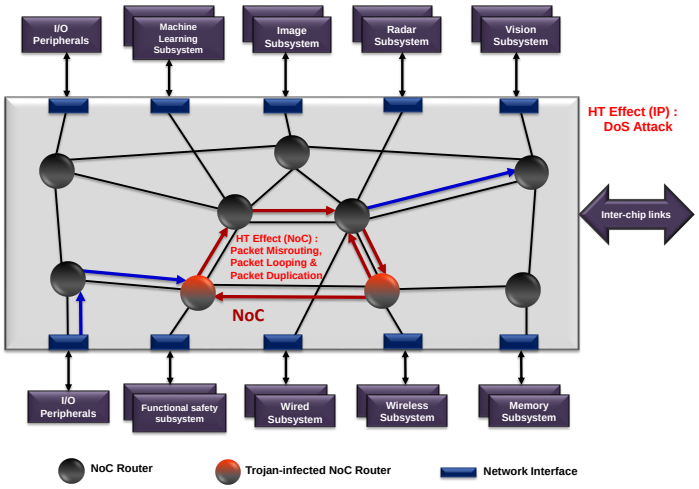


Figure 1.4: NoC without safety goals

1.3 Thesis Motivation

The growing complexity of NoC interconnects in modern many-core systems, particularly within safety-critical domains such as autonomous vehicles, increases the risk of HT attacks. These threats particularly that induce packet misrouting and denial of service (DoS) attacks within NoC routers, pose a significant threat to the seamless communication required between vital subsystems. As illustrated in Figure 1.4, HTs implanted in non-safety zones of the NoC can severely disrupt critical data flows, such as those originating from the visual subsystem. Assume a scenario where an adversary with malicious intentions implants an HT that could initiate packet misrouting in NoC routers within the non-safety NoC zone to deploy DoS attacks, where the packets get indefinitely delayed in the path or never reach their destination. Such HTs, when activated, can compromise the flow of service packets from the visual subsystem to other critical systems that might never reach destinations or encounter unexpected delays. For instance, in an autonomous vehicles, it can impair the vehicle's ability to make timely and accurate decisions—jeopardizing safety during operations like lane changes or obstacle avoidance, etc. The realisation that HT-induced DoS attacks could compromise these systems motivated us to investigate the stealthiness and impact of such packet misrouting HT circuits within NoC-based architectures. To handle such packet misrouting HT attacks, we propose a dynamic shielding and Trojan bypass technique that effectively suppresses HT's impact on system performance.

Furthermore, as described earlier, an adversary can implant variant HT circuits in visual subsystems, which, when activated collaboratively, can intermittently cause on-chip data packets to loop, leading to delays, potential DoS attacks, and degraded performance of connected IPs. To

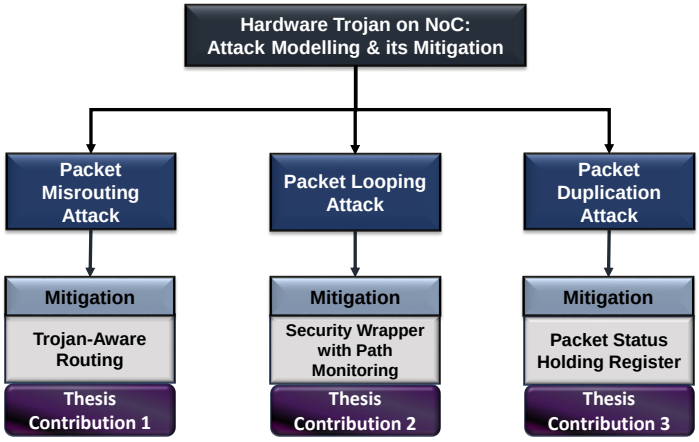


Figure 1.5: Overview of thesis contribution

illustrate the impact of these HT attacks, consider the scenario of visual layer attacks within a NoC-interconnected visual subsystem (refer to Figure 1.4). HT-induced packet looping can cause delays in inter-core communication, resulting in issues such as black pixels, distorted images, and impaired object detection. In safety-critical environments like autonomous vehicles, even a brief delay can hinder the system's ability to respond to dynamic changes, potentially leading to severe consequences. These challenges higlight the need to investigate the stealth and impact of packet-looping HT circuits in NoC architectures, particularly when triggered across multiple locations. To counter these attacks, we propose a detection and mitigation framework that prevents DoS attacks by addressing HT-induced packet cycling.

In addition to these HT attacks, this thesis explores the potential for HTs in NoC router architecture to manipulate messages during inter-core communication. For instance, an adversary could deploy HTs within specific NoC routers to target the vehicle's cache infrastructure, introducing packet duplication. This attack increases cache miss latency and overall packet latency, compromising the vehicle's computing performance. The influx of redundant data causes more Last Level Cache (LLC) misses, leading to delayed responses and impaired decision-making. These issues highlight the critical need to protect the cache subsystem and communication systems from HT-induced attacks. Motivated by this, the thesis proposes an HT model that induces packet duplication attacks and a mitigation framework integrated into the Network Interface (NI) to monitor and prevent such threats.

1.4 Thesis Contribution

This dissertation addresses the key research challenges associated with HT attacks on NoC. Figure 1.5 presents the overview of our thesis contribution. Our research is centred on developing robust HT attack models and effective mitigation strategies against three prominent classes of attacks: packet misrouting, packet looping, and packet duplication. Our primary thesis contribution addresses the HT-induced packet misrouting attacks in NoC. To fortify NoC against such threats, we propose a Trojan-Aware Routing mechanism. In the second contribution, we target the HT-induced packet looping attack in NoC. To mitigate this threat, we introduce a novel security wrapper augmented with a path monitoring module. The third contribution of our thesis addresses the HT-induced packet duplication attack, a sophisticated breach that compromises data integrity by creating unauthorized copies of transmitted packets. To counteract this, we propose the integration of a Packet Status Holding Register in NoC. Through these contributions, we aim to fortify the security of NoC architectures against HTs, thus providing solutions discussed above to mitigate packet misrouting, packet looping, and packet duplication attacks. For better understanding, we present a summary of each contribution as follows.

1.4.1 Secure NoC by Mitigating Packet Misrouting Trojan Attacks

Among the vulnerable components in a TCMP, NoC is particularly significant due to its role in facilitating communication between cores, thereby granting access to various system components [9][20][21]. Consequently, any malicious circuits in an NoC IP can affect the entire system's performance. For instance, an HT that misroutes packets can consume network resources without contributing to useful data transmission, effectively reducing the overall throughput of the NoC. This inefficiency can limit the system's ability to handle high-bandwidth applications or multiple concurrent tasks. Motivated by this, through this work, we present a packet misrouting HT threat model and the techniques to detect and mitigate the impact of such HT activation.

Threat Model: The proposed HT model tampers the routing algorithm (XY) employed in the Route Computation unit of the NoC router to enable packet misrouting. When triggered, the HT maliciously assigns a wrong output port to the head flit of a packet, making it travel to the wrong router. As a result, all the packets get misrouted (due to wormhole routing) and contribute to one of the attack scenarios: DoS and injection suppression. DoS is a scenario where the packets get indefinitely delayed in the path or never reach their destination. The injection suppression scenario is a by-product of DoS, where new packets cannot be injected into the network due to the unavailability of router buffers caused by HT-induced backpressure.

Mitigation Technique: To detect and mitigate the misrouting HT in NoC, we propose a Trojan-aware Routing (TAR) technique, which has been incorporated in every router on NoC. TAR involves three phases: Detection, Shielding, and Bypass. The core function of the Detection phase is to ascertain whether the routing unit returns a misdirected output direction for a flit by analyzing the difference in (x, y) coordinates between the current router and the intended destination router. During the Dynamic Shielding phase, the location of the HT is propagated among all the neighbouring routers of HT, which accordingly updates the direction where the HT resides. Eventually, a dynamic shielding ring is created around the HT, thereby isolating it from the rest of the network. Once the shielding ring is activated, all packets that are meant to travel through the HT get rerouted with the help of the proposed Trojan bypass algorithm.

1.4.2 Secure Routing Framework by Mitigating Packet Looping Trojan Attacks

In TCMPs, the performance of the NoC is usually measured by the QoS expected from the applications running in the system. To achieve the desired performance specification, the routing algorithms employed in an NoC router play a significant role because of their ability to balance network load and packet latency. In this work, we present an HT that induces delay for a few selected packets passing through the NoC router by modifying the path selection strategy used in the underlying routing algorithm.

Threat Model: To deploy the attack, we mount the HT on the Route Computation unit of an adaptive NoC router that uses a non-minimal odd-even (OE) routing algorithm for selecting a path [22]. In odd-even routing, to ensure deadlock freedom, East $\rightarrow$ North and East $\rightarrow$ South turns are prohibited in the even column routers. Similarly, North $\rightarrow$ West and South $\rightarrow$ West turns are prohibited in the odd column routers. Among the two phases of the OE algorithm, based on the turn restrictions, phase 1 of OE returns the possible output ports that a packet can take from the current router to reach its destination. If phase 1 returns multiple output ports, phase 2 uses an output port selection strategy to select one output port from the possible set of output ports computed in phase 1. Most of the popular output port selection strategies in OE consider buffer availability in the downstream neighbours for selecting the final output port [23][24]. Consequently, non-optimal path selection strategies can adversely affect the QoS of the underlying applications in the system. Motivated by this, we introduce an HT that corrupts the outcome of VC-based output port selection strategy [23] to deploy packet looping attacks in NoC.

Mitigation Technique: To keep the NoC capable of detecting the effect of packet looping HTs, we propose an HT detection framework that can identify the unintended packet traversal in the network with the help of a Traffic Monitor module and a Path Monitor module. With the help of the Traffic Monitor module, an NoC router detects malicious traffic and accordingly identifies the hotspot created by the HT-induced packet looping. When a router's Traffic Monitor module identifies such suspicious activity, it sends a notification to the neighbours to enable a Path Monitor module, which detects the occurrence of HT-induced cycles. To mitigate the effect of the packet looping HT, we implement a secure wrapper module, SecRC, which analyses the packets passing through the routing unit of the NoC router and generates a valid output direction for the packets.

1.4.3 Fortifying NoC Security Against Trojan-Induced Packet Duplication Attacks

Various cores in TCMPs communicate by exchanging NoC packets, consisting of a header and a payload. Usually, these NoC packets are either control packets or data packets. Control packets are used to request for data or send coherence messages, and the data packet carries the requested data [25]. To prevent data stealing in modern TCMPs, various encryption techniques have been proposed to encrypt only the packet's payload, as the header information is required for routing and arbitration decisions in the NoC. On the other hand, control packets, which do not carry any data, are usually not encrypted. The inherent necessity of the NoC packet's unencrypted header for routing and arbitration decisions at intermediate routers poses a vulnerability. The proposed HT exploits this vulnerability to deploy a packet duplication attack on NoC.

Threat Model: The proposed HT tampers the network interface (NI) unit of NoC to deploy packet duplication attacks. Similar HT model [9] facilitates packet duplication in NoC to enable data-snooping attacks. In our research, we present a refined variant of this HT model, wherein the HT, when placed in the NI, triggers packet duplication that can significantly degrade the system performance. Differing from Raparti et al.'s [9] packet-duplication HT model, our proposed variant efficiently duplicates packets and injects them into the system with minimal resource utilization.

Mitigation Technique: The proposed mitigation framework, HULK, uses three principal components: Message ID (MID), Checksum, and Packet Status Holding Register. HULK calculates MID as $MID = SRC \oplus DEST \oplus NI_{SRC} \oplus KEY$, where NI_{SRC} is the source NI and KEY is an 8-bit key generated using a pseudo-random number generator. MID is calculated at the Miss Status Holding Register (MSHR) of the L1 cache controller and added to one of the optional fields in the packet header. To preserve the authenticity of the messages, a checksum (CS) is calculated by running Alder-32 [26] on the MID and appending it in the message header. Inspired by Miss

Status Holding Registers (MSHRs), we propose Packet Status Holding Registers (PSHRs) at the NIs to track in-flight request packets. When NI_{SRC} receives a request message from its connected core, a unique entry is made with the MID, source, destination, and CS and added to its PSHR. When a response packet is received, the corresponding PSHR entry is deleted before the NI's de-packetisation is initiated. Consequently, no duplicate responses get a valid entry in PSHR, thus preventing it from entering the network.
Overall, in this thesis, we make the following contributions.

1. We propose a framework to secure NoC by mitigating the packet misrouting Trojan attacks in NoC.

2. We propose a secure routing framework for mitigating packet looping Trojan attacks in NoC.

3. We propose a security framework for NI which fortifies NoC security against Trojan-induced packet duplication attacks.

1.5 Thesis Organization

The chapter-wise organization of the thesis is as follows.
Chapter 2 provides a comprehensive background on HT attacks in ICs, the NoC architecture in TCMP and covers the potential state-of-the-art HT attacks in the NoCs. It also covers the experimental framework used for our research, including details on simulators, application workloads, benchmarks, and evaluation matrices considered for analysis.
Chapter 3 presents the primary contribution of the thesis that focuses on modelling packet misrouting HT, its impact on NoC, and strategies for detection and mitigation.
Chapter 4 presents the second thesis contribution, addressing the modelling of packet-looping HT and the proposed techniques to suppress the HT effect in NoC.
Chapter 5 discusses the third thesis contribution that involves modelling packet duplication HT and proposes the mitigation technique which counteracts the impact of such HTs in NoC.
Chapter 6 concludes the thesis by summarising the chapters and suggesting potential future works aligned with the scope of the research.

CHAPTER

2

Background

2.1 Introduction

Semiconductor companies are at the forefront of recent technologies that focus on digital transformation. With the digital transformation, the modern Industrial Internet-of-Things (IIoT) platform integrates seamlessly with machines and cyber-physical systems. Accordingly, the complexity of the services, products and processes has increased as it require more data analysis and management. To meet the needs for distributing computing, the demand for edge devices and sensors started showing exponential growth. These devices mostly rely on embedded systems that use complex ICs for smooth functioning and execution. Consequently, the market shows increased demand for ICs, making it the fourth most traded product in the world [27]. At the same time, when the sudden surge in need for digital devices went way beyond the design, verification and manufacturing capacity available with big industry players, the semiconductor industry rely on third-party vendors, thereby making the IC supply chain a globally distributed one [5]. This chapter investigates the scope and potential for embedding malicious circuits within ICs. It also examines the experimental framework employed and the performance metrics considered in the thesis.

The chapter is structured as follows. Section 2.2 provides an overview of the IC supply chain, followed by an examination of IC security in Section 2.3, exploring the potential of HT circuits within ICs. Section 2.4 presents a model and taxonomy for categorizing these potential threats. The TCMP architecture is then outlined in Section 2.5, and Section 2.6 examines potential HT-based attacks in NoCs. In Section 2.7, we present the existing methodologies to protect the NoC infrastructure from HT threats. Subsequently, Section 2.8 discusses the experimental setup

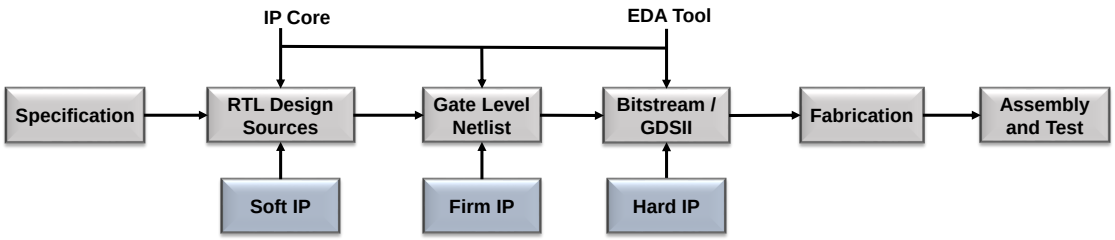


Figure 2.1: Overview of IC supply chain

considered in the thesis, while Section 2.9 discusses the applications and workloads considered for the thesis. Section 2.10 elaborates on performance metrics employed in the thesis for evaluating and analyzing the effectiveness of the proposed architectures. Finally, the chapter concludes in Section 2.11.

2.2 IC Supply Chain and EDA Tools: An Overview

The IC supply chain [28] constitutes a complex journey from conceptualization of the specification to the final product, as illustrated in Figure 2.1. The initial Specification phase defines the requirements and specifications for the IC. This process includes determining the desired functionality, performance characteristics, power consumption, and any special features or constraints that necessitate consideration. In digital hardware design, the Register-Transfer Level (RTL) design phase is a critical step in turning a conceptual design into a concrete realization. It serves as the bridge between the high-level architectural design and the physical implementation of a digital circuit. This phase involves creating the digital logic design, specifying the behaviour of the IC, and selecting the appropriate components or building blocks. Subsequently, the RTL design evolves into a gate-level netlist representing the IC's logic gates and their interconnections. This is a critical intermediary step before progressing to the physical design, as it defines the logic and connectivity of the IC. The gate-level netlist serves as the foundation for creating the physical layout of the IC, typically in a format like GDS II. This layout encompasses the positioning of transistors, interconnections, and the physical details of the IC's components, and it is essential for the subsequent processes involving manufacturing masks and fabrication. The fabrication stage primarily involves the creation of masks and the manufacturing of the silicon wafer, which includes processes such as photolithography, etching, and doping. This phase culminates in the production of silicon wafers, each bearing multiple copies of the IC. Once the silicon wafers are manufactured, they undergo separation into individual ICs through a process known as dicing. The ICs are then packaged and subjected to

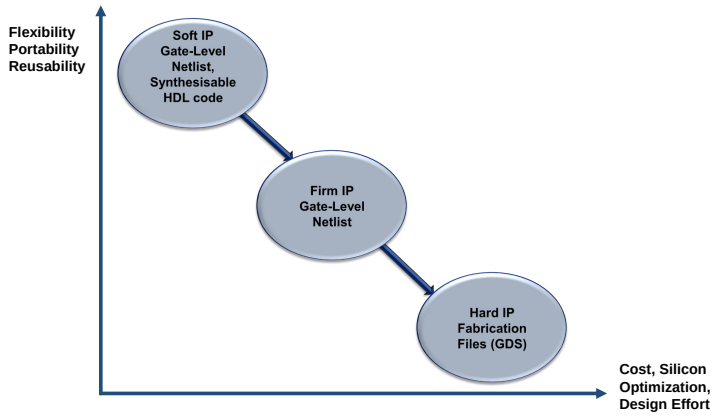


Figure 2.2: IP Core: types and tradeoffs

rigorous testing. The assembly/packaging step entails encapsulating the IC in a protective housing and connecting it to external pins for various electrical connections. The testing phase ensures that the ICs meet the specified functional and quality requirements, which may encompass various tests, including functional testing, performance testing, and reliability testing. After successfully passing the testing phase, the ICs are ready for distribution to customers or integration into larger electronic products catering to various applications.

As today's chips integrate multiple functionalities into a single chip, using pre-designed, pre-verified IP core and software blocks in the ICs is a common practice [29]. Nowadays, most IC designing starts with procuring IP cores, which represent a reusable unit of a system's functionality or its layout design whose license can be issued to multiple vendors for them to use as basic blocks in their chip design. For example, Arteris, a leading provider of SoC IPs, offers cutting-edge semiconductor system IPs to enhance SoC development. Their Arteris NoC interconnects IP and SoC integration technology have led to improved product performance, reduced power consumption, and faster time-to-market needs [30]. Since these IPs can be used for a variety of applications, vendors provide the IPs to the system designers as soft, firm, or hard IPs [5][29] with trade-off as shown in Figure 2.2. Soft IP blocks are generated as RTL models with the help of hardware description languages like System Verilog or VHDL, making them more flexible, portable, and reusable. However, hard IP blocks often come as fixed layout designs that can be directly added to the final chip layout, which makes it difficult to customize as per needs. Firm IP cores present parameterised layout descriptions that the system designers can customize and optimise based on their needs. This makes firm IPs more flexible and portable than hard IPs and more adaptable than soft IPs. Based on the complexity and application requirements, the designer teams accordingly make IP-reuse decisions and select any of the forms of IP core for the product design.

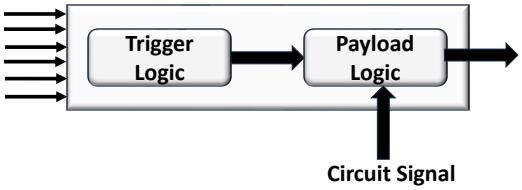


Figure 2.3: Structure of a hardware Trojan

Advancing towards the physical realisation of integrated circuit logic, logic synthesis is the major next step in the process. This crucial stage involves the transformation of the chip's RTL logic into a gate-level net-list. Using the netlist, the designer can view the entire connection list and analyse the timing and functional behaviour accordingly. Moreover, to test the performance of the device post-deployment, a system designer can verify and validate the approved design using Electronic Data Automation (EDA) tools. These EDA tools can predict an IC's behaviour after its implementation. The design team can also use EDA tools for logic synthesis. Similarly, EDA tools can generate geometric shapes that specify the actual circuit implementation, which is generally termed as place and route. A wide range of popular commercial EDA software [31] is available for various stages of chip design, including tools like Design Compiler for logic synthesis (Synopsys), Astro for placement and routing (Synopsys), and Calibre for sign-off tasks (Mentor). In addition to these commercial EDA tools, a significant and diverse selection of open-source EDA tools serve as valuable supplements, covering nearly all aspects of chip design. These open-source tools include Qflow [32], Ngspice [33] and Xyce [34] for simulation, Yosys [35] for logic synthesis, DREAMPlace [36] for placement and routing, etc. Furthermore, a growing focus is on creating comprehensive open-source EDA toolchains that can facilitate the entire chip design process, including tape-out. One noteworthy project in this area is the OpenRoad project [37], which aims to automate the RTL-GDS11 design process within a 24-hour time frame. Presently, it offers support for the commercial GF12 PDK for the 12nm process and the open-source ASAP7 PDK for the 7nm process. Overall, open-source EDA tools have significantly reduced the barriers for SoC design and have provided engineers with viable alternatives to commercial solutions.

2.3 Hardware Trojan Circuit in Integrated Circuits

The hardware-oriented attacks on TCMPs often involve exploiting vulnerabilities within the integrated circuits, which can compromise the system's security, integrity, and functionality [10], leading to security breaches in computing platforms [10] that capitalizes on various hardware information, including power consumption, cryptographic keys, memory access patterns, etc [4]. Examples of

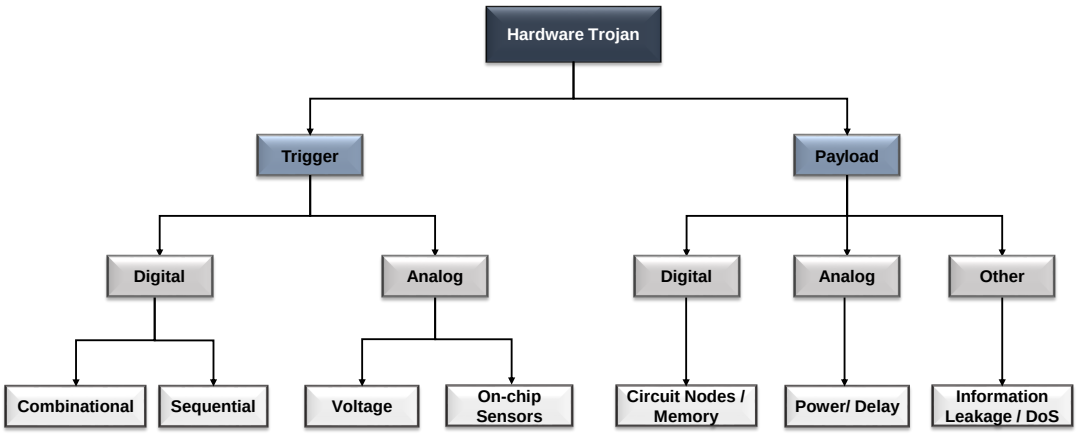


Figure 2.4: Trigger and payload logic types of hardware Trojan

hardware attacks include manipulating hardware control signals to create faults, exploiting security gaps in multiple platform interactions, vulnerabilities in firmware, and maliciously influencing hardware operations using HTs. These HTs may exploit vulnerabilities in hardware to launch attacks, such as DoS, information leakage, and unauthorized memory access. The stealthy nature of some HTs makes them hard to detect, allowing them to bypass root-of-trust techniques in device firmware, posing a significant threat to the security and functionality of MPSoCs. Figure 2.3 shows the structure of an HT. It consists of a trigger logic, which initiates the activation of an HT and a payload logic, which performs the malicious modification of the circuit or its functionality [38][39].

The trigger logic is the part of the malicious code determining when the malicious activity should be initiated. It serves as the activation mechanism for the malicious payload. The trigger could be based on specific conditions, events, or a combination of factors. It can be categorized into digital and analog triggers, each with its own classifications further, as shown in Figure 2.4. The digital trigger relies on the instantaneous logic level of the input signals to activate the HT. It is based on the current combination of inputs, making it responsive to specific conditions in the digital circuit. In contrast, sequential digital triggers depend on the state of the circuit over time. They may activate the HT based on a predefined sequence of states or events within the hardware. In the case of analog triggers, it exploits on-chip sensors to activate the HT based on environmental conditions such as temperature, voltage, or electromagnetic radiation. In some cases, manipulation of the voltage levels can serve as an analog trigger, initiating the HT based on specific voltage thresholds. Understanding the triggers is crucial for identifying the conditions under which an HT may activate, aiding in developing detection and prevention strategies.

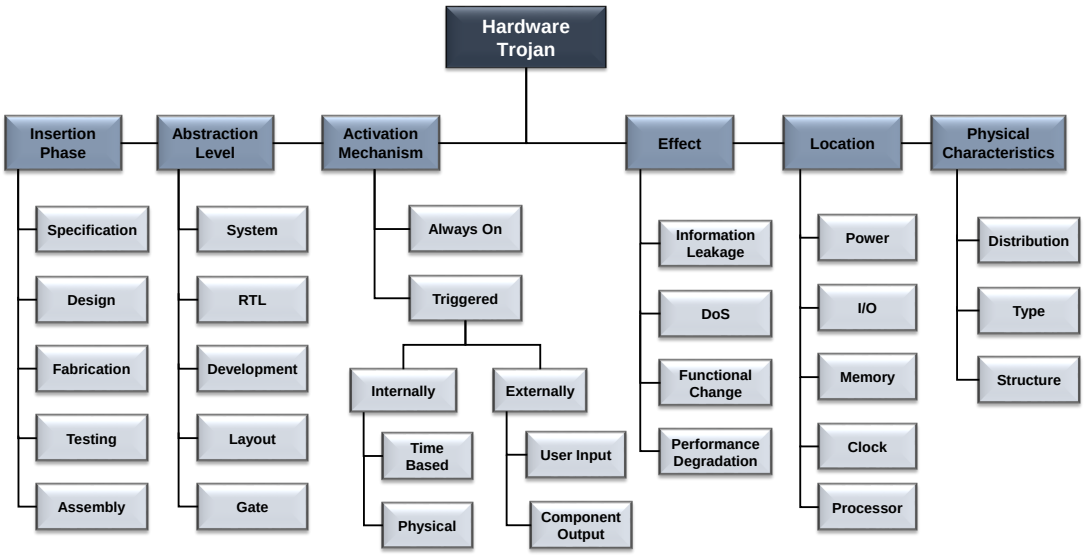


Figure 2.5: Hardware Trojan taxonomy

The payload logic of the HT performs malicious modifications to the targeted hardware system or its functionality to achieve the attacker's objectives. The payload can be in different forms, including digital and analog payloads. A digital payload may manipulate circuit nodes or memory elements within the hardware. This could involve altering data stored in memory or disrupting the functionality of specific circuit nodes, leading to unauthorized access or data corruption. On the other hand, analog payloads can affect power consumption or introduce delays in signal propagation, which could result in altered power profiles or timing characteristics, impacting the overall performance of the hardware. Beyond this, an adversary can insert HTs in circuits to deploy an attack like information leakage, DoS attacks, etc.

The feasibility of HT attacks within TCMPs is limited to the resources available to the attacker and their level of control. Typically, during the outsourcing of chip fabrication, the third-party vendor gains access only to the chip layout. From the attacker's perspective, lacking access to the RTL code or netlist makes it infeasible to replicate existing RTL code or netlist to insert a malicious circuit at this stage. However, leveraging the capabilities of EDA tools, a rogue/untrusted employee can insert or modify a malicious circuit to introduce an HT into the final layout [21][40][41]. Similarly, a rogue/untrusted employee working in an outsourced design team (where design files and RTL code are easily accessible) can implant malicious circuits that leverage rare events for HT triggering and exhibit low power variations [17][42][43]. Although such HT circuits may evade detection through logic testing or formal code checks, side-channel analysis remains a viable detection method [44][45][46].

2.4 Hardware Trojan Taxonomy

An HT can do alterations in any of the stages in the IC life cycle or in the abstraction level, as shown in Figure 2.5. IP vendors can easily manipulate the RTL and insert malicious codes to modify the system's behaviour [14]. Even EDA vendors can alter the logic implementations deduced given by EDA tools, which can do more than required to activate the HT. The chip fabrication process can also become insecure by untrusted staff with access to the fabrication process [38][39]. Usually, an HT is termed functional if it alters the number of components in the original design, whereas a parametric HT modifies the existing code. The taxonomy in Figure 2.5 provides a comprehensive framework for HT categorisation based on different phases, characteristics, and activation mechanisms. The detailed taxonomy outlined below offers a structured method for understanding the complex nature of HTs.

1. **Insertion Phase:** HTs can be introduced during the specification phase, where the functional requirements of the hardware are defined. This may involve manipulating the intended functionality to create vulnerabilities. Even during the design phase, malicious elements can be inserted into the hardware through manipulations in the design specifications, potentially exploiting weaknesses in the architecture. The fabrication phase involves the physical creation of the hardware. HTs can be implanted at this stage, either intentionally or due to malicious components that may be introduced during the testing phase, taking advantage of the complexity and scale of modern hardware testing procedures. Assembling the components into a complete system offers another opportunity for HT insertion, particularly if there are vulnerabilities in the assembly process.

2. **Abstraction Phase**: At the system level, HTs can be strategically placed to compromise the overall functionality and security of the hardware system. HTs may exploit vulnerabilities at the RTL level, influencing the behaviour of the hardware at a lower abstraction level. Manipulating the development environment can also introduce HTs, affecting the hardware's intended design and functionality. The layout phase involves arranging components on the chip. HTs could be inserted by manipulating the physical layout to compromise performance or security. Furthermore, it may alter the logical gates and connections at the gate level, impacting the hardware's operation.

3. **Activation Mechanism:** Some HTs remain continuously active, potentially performing malicious actions throughout the hardware's life. In other cases, HT activation may occur

at predetermined intervals due to internal factors such as time, physical conditions, specific events, or environmental conditions like temperature or power variations, user inputs, etc.

4. **Effect:** With activation, HTs may compromise the confidentiality of data, leading to unauthorized access or exposure of sensitive information. While some HTs may disrupt normal system operations, causing a denial of service to legitimate users, other HTs can alter the intended functionality of the hardware, impacting overall system performance.

5. **Location:** HTs can be strategically positioned within the power distribution system to manipulate power-related characteristics. Malicious components might be placed on input/output interfaces also to alter data and commands. When inserted into memory modules, HTs can influence data storage and retrieval operations. Another potential target for HTs is the clocking system, where manipulation can impact the timing and synchronization of the hardware. Placing HTs within the processor can profoundly affect computational operations. These considerations highlight the diverse locations within a hardware system where HTs can be deployed to compromise functionality and security.

6. **Physical Characteristics:** HTs can exhibit variability in their distribution across the hardware, influencing their detectability and impact. Different types of HTs may demonstrate distinct characteristics, necessitating customized detection and mitigation strategies. The internal structure of HTs is subject to variation, impacting their behaviour and the potential for detection. HTs may be categorized into various types based on their intended function and the extent of their impact on the hardware.

While HT implants are feasible in other phases of the IC supply chain, such as testing and distribution, attacks rooted in untrusted CAD tools appear to be stealthy, given their ability to integrate malicious circuits during the synthesis and verification stage of the IC [47]. As IC designers predominantly rely on CAD tools for various chip design and synthesis phases, a compromised CAD tool or its script can modify the IPs from HDL code to netlist. The attacker's capabilities in this context encompass adding back doors for information leakage, performance degradation, or time bomb attacks. Tiago et al. [41] demonstrated that a side-channel HT could be added by replacing spare/filler cells in the design, enhancing its stealthiness and making it harder to detect. Basu et al. [48] demonstrated the feasibility of such CAD attacks by launching an HT attack on an ARM Cortex processor. The motives behind deploying these HTs often include corrupting critical computations in systems to degrade system performance, exposing cryptography keys, breaching confidentiality and integrity, resource depletion attacks, fault injections, etc. Such HT attacks are

extremely challenging to detect, even during verification, as the CAD tools used in the design house are often provided by the same vendor [47]. Moreover, malicious circuits in the back-end design go unnoticed, as few techniques exist to compare the final IC design with the intended one.

2.5 Overview on Tiled Chip Multicore Processors

TCMP represents an advanced multicore processor architecture that optimizes parallelism and efficiency in computing systems. Unlike traditional multicore processors, where multiple cores share a unified memory architecture and shared bus for inter-core communication. TCMP consists of a collection of tiles. These tiles are arranged in a grid-like fashion, and they communicate with each other through a packet-based on-chip interconnect, NoC. Figure 2.6 represents the 8×8 mesh NoC, where tiles are arranged in a grid manner using mesh topology. Here, each tile represents a processing core accompanied by a private L1 cache, a shared L2 cache, and an NI linked to an NoC router for inter-core communication. The private L1 cache in the tile serves as a storage for frequently accessed data and instructions specific to each core, optimizing performance by reducing access latency. Meanwhile, the shared L2 cache acts as a larger cache memory pool accessible by all cores, enhancing overall system efficiency. To enhance data access and minimize contention, the Sequentially Non-Uniform Cache Access (SNUCA) technique is employed for mapping L2 cache sets to different tiles. This approach ensures that the L2 cache, while shared, is also physically distributed across tiles, thus mitigating bottlenecks and optimizing cache utilization. Each tile is connected to an NoC router through NI, which enables efficient packet routing across cores, ensuring reliable and efficient inter-core communication within the system. These routers are equipped with ports facilitating communication in different directions (North, South, East, West) as well as a local port enabling communication with the tile itself.

NoC is the most popular on-chip communication framework for TCMPs [49][50][51]. It is characterized by technological features like (i) die size reductions, (ii) packetized communications, and (iii) multiple levels of clock gating. It also provides separation between computation and communication, supports modularity and IP reuse via standard interfaces, and handles synchronization issues, improving the system's performance [51]. An NoC router plays a vital role in forwarding packets to their desired destination cores. Due to bandwidth limitations, all packets that reach an NoC router are converted into a sequence of flow control units called flits. Based on the content and control information, these flits are categorised as head flit, set of body flits, and tail flit [3][52][53]. The head flit contains essential packet header information such as the destination address, packet type, and control information used for routing through the network. Body flits

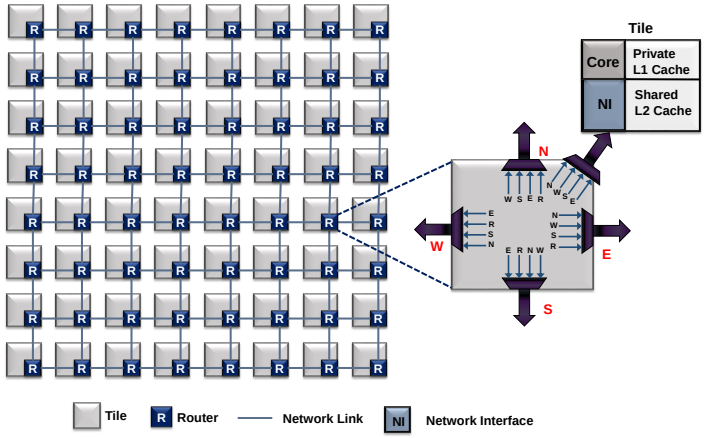


Figure 2.6: 8×8 mesh-based NoC in Tiled-Chip Multi-core Processor

carry the main payload or packet data. Depending on the size of the packet, there may be multiple body flits following the head flit. The tail flit indicates the end of the packet and may include checksums or error detection codes for data integrity verification.Figure 2.7 shows the architecture of a conventional NoC router. It consists of three main phases: Route Computation (RC), Virtual Channel (VC) Allocation, Switch Allocation (SA).

The process begins with the input ports, where flits are received and prepared for routing. Each input port has a set of flit buffers (VCs) to store incoming flits. VCs help in managing and prioritizing incoming flits. When the head flit is present at the VC head, the RC logic determines the output path for the packet based on the destination information carried by the head flit. Different routing algorithms may provide different numbers of output paths and VCs, resulting in varying RC outcomes, which impact the complexity of the VC allocator and switch allocator. The VC allocator handles requests from input VCs after the RC stage. It ensures that each input VC is allocated at most one output VC and vice versa. Input VCs that do not grant output VCs retry in the subsequent cycle. Both RC and VC allocation occur at the packet level and involve only the head flit. All body and tail flits follow this routing decision, thus complying with worm-hole switching. Following successful VC allocation, the router checks if the downstream VC has available buffers. If so, the flit requests switch traversal from the SA stage. Like VC allocation, the switch allocator ensures that each switch output port is assigned to at most one flit. It also generates control signals to connect the input and output ports of the crossbar. To minimize area and power consumption, these allocators typically employ simple round-robin arbiters. Once a flit completes the SA stage, it goes to the crossbar for further traversal to its next hop router. Also, NoC routers employ credit-based flow control to maintain a smooth data flow. This mechanism involves using credit counters, where each router maintains information about the available VCs in its downstream routers. Hence a packet is

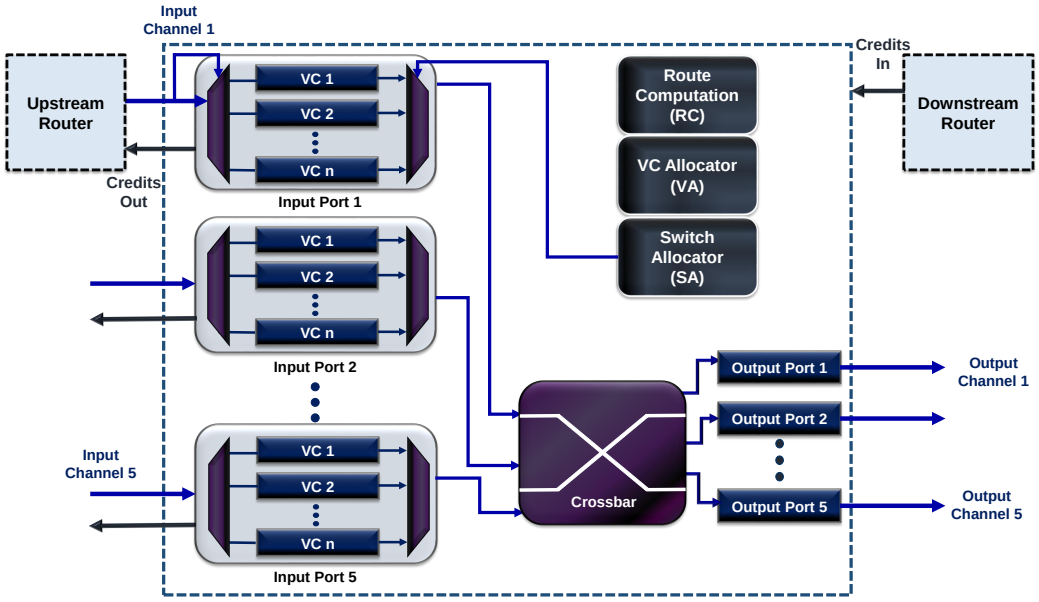


Figure 2.7: Conventional NoC router architecture

forwarded to the next router only when sufficient credits are available in the downstream router. As all the connected cores of TCMPs share NoC, they are vulnerable to hardware-oriented security threats, where an attacker can analyze the communication flow and variation in power consumption to launch attacks like information leakage, side-channel attacks, DoS attacks, etc. Among other attacks, HTs mounted on NoCs impose unique challenges as they remain hidden until triggered. Identifying the rare condition that triggers an HT requires an examination of all possible input patterns to the NoC. However, mostly, it is not feasible due to the time-constrained post-silicon debug and validation [5][15][21][54][55].

2.6 Hardware Trojan Impact on NoC

This section explores various attacks and their impact on NoC-based systems. Figure 2.8 illustrates several well-known HT attacks in NoC, which are discussed in detail below.

2.6.1 Denial of Service Attack

DoS attack by an HT on NoC mainly aims at resource depletion, including bandwidth, being one of the most critical resources in a communication framework. The HT can deploy such an attack by flooding the network with frequent and useless packets [21][56][57] in such a way that the genuine packet will never reach its actual destination or get delayed [58][59]. As a consequence, the victim packets suffer from buffer and link unavailability, delaying or halting the entire NoC communication.

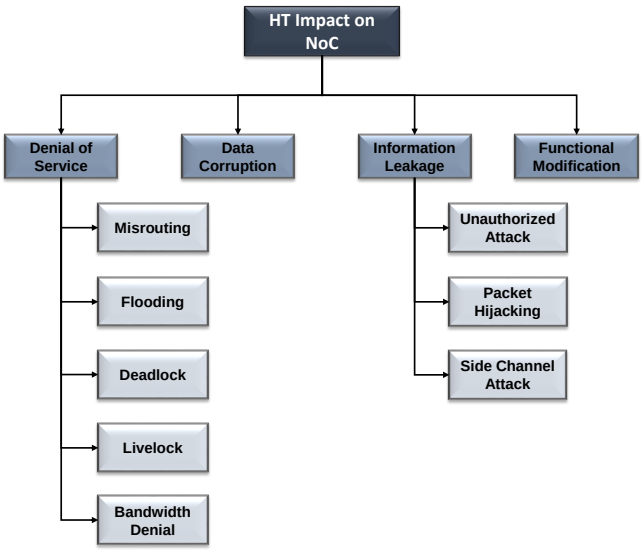


Figure 2.8: HT classification based on impact

Deadlock occurs in NoC when packets keep some resources (channels and buffers) while requesting other resources. HTs in the route compute module of an NoC router can activate an attack by changing the routing algorithm. In some cases, these modifications on routing done by an HT can create deadlocks and livelocks. HTs can also misroute packets, thus denying them to reach their destination [60]. The purpose of misrouting HT is to send the packets away from the destination, which can lead to DoS attacks. Such misrouting can be done by techniques like changing the packet's source or destination address to an illegal address [61], transmitting wrong information about the availability of the buffers in an NoC router, creating routing loops, etc. This behaviour of the HT makes the victim packets suffer from livelock, thus wasting the resources without any productivity. Recent research works discuss the possibilities of such HT attacks in NoC circuits that eventually tamper the quality of service of the applications running in TCMPs [7][56][62][63][64].

2.6.2 Information Leakage Attack

The primary goal of a leakage HT is to intercept ongoing data communications within a NoC and illicitly acquire crucial information. Attackers exploit shared on-chip resources like caches and the NoC itself as covert entry points to compromise victim processes, applications, and IPs. Prime+Probe attacks in caches identify the cache line eviction pattern of a victim process by creating conflicts to steal sensitive information [11]. Other well-known cache attacks include Flush+Reload [12], Flush+Flush [65], and Streamline [66]. An NoC-based variant of the Prime+Probe side-channel attack monitors the contents of the shared caches through the infrastructure of the NoC [67]. In

certain instances, information leakage and data snooping attacks involve duplicating incoming packets from a processing core [68][69]. Such HTs are typically triggered by an accomplice thread residing in a separate core. The HT thus initiates the transmission of snooped critical data to the accomplice thread [70][71]. A similar attack occurs wherein the HT is mounted on the NI of one IP and extracts information for an accomplice thread concealed in another IP [9].

2.6.3 Data Corruption Attack

A data corruption HT attempts to learn the data transmitted through the NoC when certain conditions are met, such as specific data patterns being transmitted through the NoC at a particular time or a signal sent by the attacker [57][72][73]. Once activated, the HT modifies the data passing through the NoC. This modification could involve flipping bits [74], altering packet headers [58][59][64], or injecting false data packets into the network[62]. As a result, the integrity and reliability of the data being transmitted within the NoC are compromised. The consequences of the data corruption attack can be severe, depending on the application running on the affected system. In TCMPs, the corrupted data exchanged between processor cores via the NoC could lead to incorrect computation results, system crashes, or unauthorized leakage of sensitive information. For example, through side-channel analysis, an adversary can learn the key used for packet encryption [72].

2.6.4 Functional Modification Attack

In a functional modification HT attack in NoC, an attacker implants malicious circuitry into the design of the NoC hardware during its fabrication. This HT is designed to modify the normal functionality of the NoC components, potentially leading to various security breaches or system malfunctions. When the trigger conditions are met, these HTs may modify the routing tables within the routers to redirect data packets to unauthorized destinations or to introduce delays or congestion in specific network paths[60][75][76]. It could even intercept and modify data packets traversing the NoC, altering their contents, headers, or destinations [42][64].

2.7 Classification of HT locations in NoC Routers

This section outlines some of the state-of-the-art HT models. Here, we classify the existing HT-based attacks on NoC routers based on the location of HT as shown in Figure 2.9. We summarise the HT models and their impact and also discuss various HT detection and mitigation strategies within the NoC router, IPs, and network links, as presented in Table 2.1 and Table 2.2.

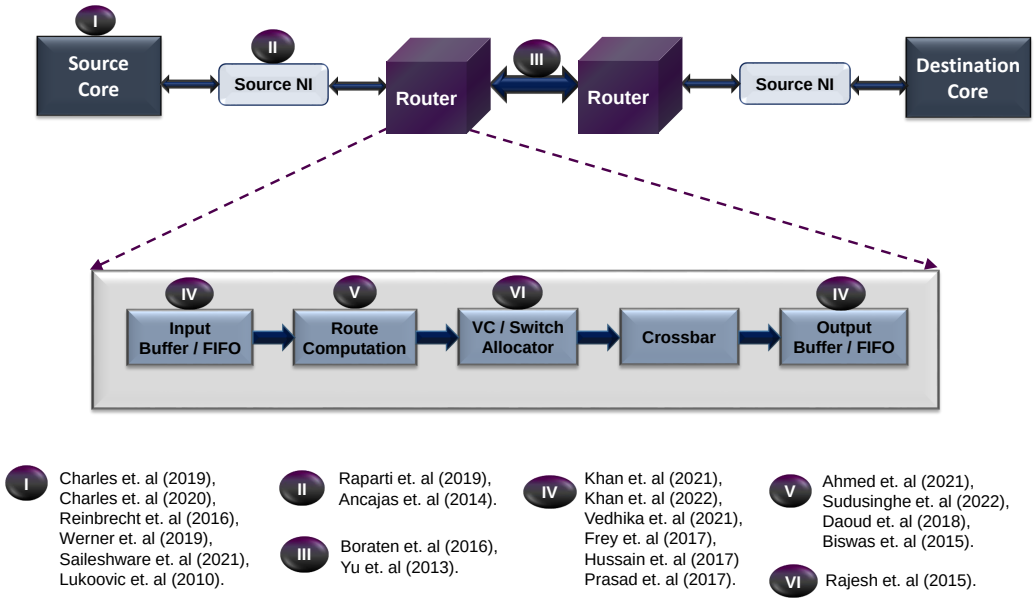


Figure 2.9: Trojan locations in NoC-based systems

2.7.1 Trojan in Route Computation Unit

An HT in the route computation unit changes the routing information to activate the attack. Biswas et al. propose an HT that changes the routing table contents [75], which in turn launches misrouting and information leakage in the network. Here, the cores connected to the NoC are categorized as a secure region and a non-secure region. An attacker will try to change a router's routing table nearer to the secure region to leak the data arriving from or destined to a secure region or divert the data in the wrong direction. Here, HT detection is done with the help of run-time monitors, which keep track of acknowledgement packets during the inter-core communication between secure and non-secure regions. The attack is mitigated by restart monitors, which check for the malicious router before starting the communication. The above system has several limitations that need to be addressed. Firstly, a notable increase in area and power overhead is attributed to the additional registers and circuitry required by the run-time and restart monitors. This can impact the overall efficiency of the system. Furthermore, altering all packets from secure to non-secure regions with router addresses in the run-time monitor can lead to congestion issues and throughput reduction. The need to have knowledge about network traffic conditions introduces a dependency on several external factors. The system is also susceptible to run-time delays caused by congestion resulting from monitor packets moving in the network. Application halts during the initiation of restart monitors create a bottleneck for the seamless operation. Recent research utilized a fusion of machine learning techniques and an integrated collective decision-making strategy to detect eavesdropping attacks [69].

Table 2.1: Hardware Trojan placements in NoC (1)

Type of Attack	*Trojan Location & Mitigation Techniques Used*	*References*
Trojan Circuit in Route Computation Unit		
Information Leakage Attack	HT: periodically packetise the count and send it to an external attacker for information leaking. Mitigation: machine learning techniques with decision making strategies.	Ahmed et. al [15] (2021) Sudusinghe et. al [69] (2022)
Denial of Service Attack	HT: modify the routing algorithm to deploy packet misrouting. Mitigation: hardware-software oriented shielding mechanism.	Daoud et. al [60] (2018)
Information Leakage Attack	HT: modify the routing table to deploy data leakage. Mitigation: run-time monitors which keep track of acknowledgment packets during the inter-core communication.	Biswas et. al [75] (2015)
Trojan Circuit in Network Interface		
Information Leakage Attack	HT: duplicate packets in NI and inject into NoC. Mitigation: snooping invalidation module.	Raparti et. al [9] (2019)
Information Leakage Attack	HT: deploys packet duplication along with the signals from accomplice thread. Mitigation: three-layer protection scheme consisting of data scrambling, packet certification and node obfuscation is used.	Ancajas et. al [68] (2014)
Trojan Circuit in Switch Allocator		
Bandwidth Denial Attack	HT: modifies the source/destination of flits. Mitigation: proximal analogous packets using the proxy core.	Rajesh et. al [62] (2015)

HTs in an NoC router can misroute packets to trigger a DoS attack [60]. A runtime HT detection algorithm uses the incoming direction of the packets to detect the location of such HTs. 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. Remote access HT [15] can count the number of packets traversing the specific NoC routers over a specified time interval. These HTs periodically packetise the count and send it to an external attacker. This can help in traffic profiling for floating attacks on selected packets.

2.7.2 Trojan in Network Interface

An adversary can deploy an HT that can initiate a data-snooping attack by modifying the NI of an NoC router. A third-party NoC IP provider can insert an HT, which is capable of detecting and responding to commands embedded in the flits by an accomplice thread [68]. During the operational phase, the accomplice thread issues commands to initiate malicious activities, such as duplicating specific ongoing data communication. To mitigate the effect of the HT, the security framework uses a three-layer mechanism. Layer 1 introduces a low-overhead data scrambling technique at the lowest level. Layer 2 establishes a dynamic packet certification mechanism between the NoC and the processing element, preventing the forwarding of flits with invalid certificates. Finally, Layer 3 introduces a node obfuscation technique at the topmost layer, dynamically concealing communication nodes in the NoC and introducing substantial noise to enhance side-channel resilience. These schemes lack an efficient and low-power attack detection mechanism.

HTs can also modify the flit queue in the NI of an NoC router to deploy the information leakage attack through packet duplication [9]. In certain cases, these HTs get activated when an accomplice thread sends the message. Once activated, the HT starts monitoring the core's cyclic flit queue located in the NI. Whenever a head flit is transmitted from the NI queue to the router, HT exploits its location in the queue. Before the location of the head flit is overwritten in the queue, HT copies the content of the head flit into a new flit and sets its destination as the address of the malicious core where the accomplice thread resides, thus forming the information leakage attack. Here, the packets that reach NI are duplicated by HT with less interference to the standard NI functions and sent to the accomplice thread. Unlike the state-of-the-art techniques [68], a snooping invalidation module is integrated with the processing element for mitigating such HT attacks. The circuit utilises threshold voltage degradation for run-time mitigation of data snooping.

2.7.3 Trojan in Switch Allocator

HT can manipulate source and destination flits during the router micro-architecture's arbitration and allocation stages [62]. Hence, HT can de-prioritise victim flits in the arbitration phase and subsequently suppress them, leading to delays during the allocation phase. To mitigate the HT effect, specialized packets are used to identify the latency of ingress and egress packets in the network. These packets are sent to the nodes proximal to HT, and the packet latency is then compared with their original counterparts for HT detection. This can introduce resource contention and network latency, which can affect the overall system performance.

2.7.4 Trojan in Input/Output Buffers

An HT at input/output buffers can initiate an attack that changes the flit type and packet destination address and can sabotage the integrity of a packet, which results in a bandwidth depletion attack [77]. In NoCs, FIFO-based input/output buffers become a primary target for attackers due to their regularity in implementation and large area footprints. Resource depletion is the main consequence of an HT at input/output buffers. This can be mitigated using a dynamic flit permutation technique with the help of a local random vector generator for the incoming flits in the input buffer. Flits can be encoded with error control codes to retain integrity. At the destination, a flit de-permutation and error-correcting code decoder unit is used to recover the corrupted flit back to its original form. However, adding error-correcting code and a local random vector generator in each router contributes to a substantial area overhead. This increase in routing switching power is another concern, impacting the overall power consumption dynamics of the system. The absence of a specified trigger for HT initiation is identified as an additional limitation in the system's functionality.

Another possible attack in the input buffer is an illegal packet request attack [61], where HT creates a new packet and injects it into the network when the local core is idle. This can lead to a DoS attack. Here, the attack is detected by a security unit that verifies the source/destination of the header flit for the modification. Buffer masking or buffer isolation is used as the mitigation technique. The above system has several limitations. The implementation of buffer isolation, intended for security purposes, results in a degradation of network performance. Moreover, a dedicated security unit is necessary for each virtual channel in the router unless a unified buffer management scheme is employed. As the number of virtual channels per input port rises, the system experiences increased power and area overhead. Additionally, any reduction in virtual channels directly impacts the execution time of applications, posing challenges to overall system efficiency.

A recently proposed HT in the NoC router alters the destination field in the header of packets to attack an SoC for performance degradation [58]. Once triggered, the proposed HT can bring the application to a complete halt by stalling the instruction issue. Modification of control fields in the packet by an HT for leaking packet data to malicious applications is explored recently [74]. To mitigate this type of HT, the authors suggest an authentication mechanism where control fields of the packet are tagged with a dynamic random value and the tag is scrambled with the packet data. The suggested mitigation technique that involves tag generation and key distribution among various cores makes unrealistic assumptions and involves unacceptable processing time in TCMPs.

A delay Trojan [59] that deliberately imposes random delays on the flits entering the input buffers of NoC routers can create a delay of service attacks. To counteract the impact of the delay

Trojan, they introduce a novel dynamic adaptive caging circuit [78]. This circuit analyses the average time spend by a packet in the router and the time spent in the previous router. Upon the arrival of a packet, these values undergo comprehensive processing using a delay comparator. Hence packets that exhibit a significantly greater delay in their previous router than the average delay experienced across all the routers they have traversed are identified. This detection mechanism excels at identifying delays originating from HTs manipulating internal buffers. However, this detection methodology fails when HT-induced delay is not quantifiable in one router alone.

2.7.5 Trojan in Network Link

Existing literature has mentioned about an attack using HT that snoops on passing packets in the network link to inject a fault into some target packets [57]. The fault corrupts a packet and triggers re-transmission, which in turn creates congestion and then deadlock in the on-chip network [79]. This attack uses a kill switch to control the HT activation and avoid HT triggering during the verification process. Here, the HT stores a target block, which is used to identify the victim packet by checking the information like source, destination, VC, memory address, etc. It has a payload counter that is used to inject faults at different locations and thus avoids getting noticed by fault-aware architectures. The HT also has an XOR tree to change the bits on the wires selected within the link during the attack. A threat detection module integrated into the output buffer of an NoC router is used to monitor the re-transmission of packets and the possibility of transient or permanent faults. These kinds of attacks can be circumvented by a switch-to-switch mitigation technique, which obfuscates the packets to avoid HT triggering. The utilization of flit obfuscation techniques, such as scrambling, inverting, and shuffling at each router, introduces performance degradation. The employed flit shuffling mechanism may also potentially violate the concept of wormhole switching.

2.7.6 Trojan in Processing Cores

Conventional verification processes and tools encounter limitations in affirming the trustworthiness of third-party IPs, primarily because of the absence of trusted reference designs or golden models in the verification process. An HT can deploy a code-injection attack, which redirects the execution of a trusted application to the malicious code by taking over the instructions' control flow [82]. This is usually done by exploiting buffer overflows and smashing stack content. Beyond this, recent research shows that a malicious IP can launch attacks such as DoS, flooding, and high communication latency, which can result in saturation [56][83]. These can be detected and localized by monitoring packet arrival curves. Destination packet latency curves are generated at each router

Table 2.2: Hardware Trojan placements in NoC (2)

Type of Attack	*Trojan Location & Mitigation Techniques Used*	*References*
Trojan Circuit in Input/Output Buffers		
Delay of Service Attack	HT: imposes intermittent packet delay on the flits. Mitigation: dynamic adaptive caging circuit.	Khan et. al [59] (2021) [78] (2022)
Packet Dropping Attack	HT: modifies the destination field of the flit.	Vedika et. al [58] (2021)
Denial of Service/ Information Leakage Attack	HT: modifies the header information of the flit. Mitigation: dynamic flit permutation technique.	Frey et. al [77] (2017) Hussain et. al [74] (2017)
Deadlock	HT: deploys packet duplication. Mitigation: buffer masking and buffer isolation.	Prasad et. al [61] (2017)
Trojan Circuit in Network Link		
Denial of Service/ Permanent/ Transient Link Error	HT: deploys packet corruption. Mitigation: flit de-permutation and error correction code decoder module at output buffer.	Boraten et. al [57] (2016) Yu et. al [79] (2013)
Trojan Circuit in Processing Cores		
Distributed DoS Attack/ Bandwidth Depletion Attack	HT: deploys packet flooding. Mitigation: statically profiled traffic behaviour analysis.	Charles et. al [80] (2020) [56] (2019)
Last Level Cache Attack	HT: cache content snooping. Mitigation: isolation-based techniques.	Reinbrecht et. al [67] (2016) Werner et. al [81] (2019) Saileshwar et. al [66] (2021)
Buffer Overflow Attack	HT: deploys code injection. Mitigation: stack protection unit, instruction trace unit and security manager at NoC.	Lukoovic et. al [82] (2010)

with the help of a leaky bucket algorithm. This algorithm uses packet arrivals and the history of packet streams to mitigate the attack. However, as the system relies on prior knowledge of the communication patterns within the network, it may not always be accurate.

Attackers typically use shared on-chip resources like LLC and NoC as a backdoor to access the victim processes, applications and IPs. One of the earliest attacks on LLC, called Prime+Probe, tries to identify a victim process's cache line eviction pattern by creating conflicts to steal sensitive information [11]. Other popular attacks on LLC includes Flush+Reload [12], Flush+Flush [65], and Streamline [66]. Another Prime+Probe-like attack compromises TCMP security by monitoring the contents of the shared LLC [67]. The NoC Prime+Probe attack comprises four stages. The infection stage involves injecting malware into the MPSoC, potentially spreading across multiple IPs intersecting the sensitive path. In the Prime stage, the infected IP prepares the cache by overwriting memory locations, ensuring the absence of AES lookup tables, and initiating cryptographic tasks. The Probe stage monitors accessed cache locations during AES execution through the identification and reading phases, collectively revealing memory locations. Here, the strategic placement of infected IPs reduces false positives by detecting collisions and identifying cache requests. An LLC attack can even be deployed that leverages the cache to buffer data between the sender and receiver [66]. The technique relies on thrashing to naturally expel the transmitted data from the cache after the communication process. Such HTs can be detected by utilizing performance counters or specialized hardware. However, the above-mentioned LLC attacks that resemble those of generic memory-intensive applications may evade detection. On the other hand, isolation-based approaches [81] can prevent cache sharing among different trust domains, effectively mitigating covert channels with trade-offs such as performance costs and scalability challenges.

2.8 Experimental Modelling

Using the right experimental framework for modelling hardware units and assessing their performance is a significant step in computer architecture research. For such experiments, FPGA and ASIC boards offer a flexible platform for simulating and analyzing circuit behaviours, enabling researchers to explore the performance of different hardware architectures. While real hardware provides valuable insights, its high implementation costs, resource limitations, and scalability challenges make researchers rely on computer architecture simulators, which enable the emulation of various hardware operations in a controlled environment. To enhance the authenticity of the experimental framework and replicate real-world conditions, researchers use workloads that reflect the operational conditions and hardware responses in diverse scenarios. This section discusses the

experimental framework used in this thesis to design and analyse the proposed HT models, their detection/mitigation techniques and the workloads that are used to assess the performance of the various architectures under consideration.

2.8.1 Computer Architecture Simulators

Computer architecture research often uses simulators to test and analyse the architectural and micro-architectural characteristics, performance, and power consumption of proposed processor/memory/communication models. Usually, the simulators are classified based on the type of simulation, the scope of the system that is being simulated (also referred to as the target machine) and the type of input given to the simulator. Simulators are broadly classified as functional and timing simulators. Functional simulators emulate a target's instruction set architecture (ISA), similar to emulators. While they are generally faster than other simulators, they do not incorporate micro-architecture implementation. However, timing simulators, also called performance simulators, replicate the micro-architecture of processors and provide detailed insights into the timing and performance aspects of a target system, such as throughput, program runtime, memory system performance, etc. Timing simulators come in various sub-types: cycle-level, event-driven, and interval simulators. Cycle-level simulators usually replicate the architecture by mimicking the operation of the simulated processor for each cycle. On the other hand, event-driven simulators simulate the target system based on events rather than cycles. Typically utilizing event queues, they advance simulation to the time when a scheduled event occurs, bypassing the need to go through all cycles. Interval simulation represents the conventional flow of instructions through the pipeline that can be segmented into intervals defined by certain events, such as cache misses or branch mispredictions. Here, branch predictors and the memory system of the architectural simulators are used to simulate the respective events and accordingly determine their precise timings.

Functional simulators are frequently combined with timing simulators to enhance simulation flexibility and accuracy. This integration improves the precision of modelled timing-dependent instructions, such as synchronization and I/O operations [84]. A classic example of a simulator employing this approach is gem5 [85]. Another relevant factor when categorising simulators is the scope of the simulated target system. Based on this criterion, simulators can be divided into full-system and application-level/user mode simulators. A full system simulator can fully boot an operating system and execute application benchmarks on that operating system, replicating the normal operation on a real target machine. Application-level/user mode simulators focus on emulating the microprocessor and a restricted set of memory and peripherals. In these simulators,

the system calls are typically bypassed by the simulator and are handled by the underlying host operating system. In the subsequent sections, we discuss various simulators employed for modelling the hardware architecture considered in the thesis, along with the tools for analysing the power and hardware footprint of both the proposed techniques and state-of-the-art methods.

2.8.2 gem5

The gem5 is a full-system event-driven simulation tool that models a complete computer system with a flexible and diverse set of CPU, system execution, and memory system models. It combines the strengths of GEMS (General Execution driven Multiprocessor Simulator) [86]and M5 [84] simulators. Here, GEMS contributes complex memory models and interconnect models to gem5. M5 focuses on CPU models, ISAs, I/O devices, etc. The simulator operates in an event-driven manner, allowing various components to schedule their events. gem5 is known for its flexibility and supports various CPU models and system execution scenarios. The modular design allows seamless integration of different components. The gem5 is predominantly written in C++, and some aspects are written in Python, adding a layer of flexibility and ease of use to the simulation environment.

One of the primary reasons for gem5's popularity is its ability to run real workloads. This provides researchers with a realistic environment for testing and evaluation. This capability is crucial for assessing the performance and behaviour of hardware architectures under conditions that closely resemble real-world scenarios. The gem5 facilitates rapid early prototyping of hardware architectures. Its modular and object-oriented design allows the users to prototype and experiment with different components and configurations quickly. Memory-system modelling is a critical aspect of computer architecture, and gem5's ability to simulate intricate memory hierarchies, interconnects, and interactions provides a detailed understanding of how different hardware configurations affect overall system performance. This unique attribute allows gem5 to strike a balance between adaptability in experimentation and maintaining a high level of precision, making it a versatile choice for various stages of hardware design and validation.

Figure 2.10 shows the system-level view of gem5. It supports various Instruction Set Architectures (ISAs), encompassing Alpha, ARM, MIPS, Power, SPARC, and x86. The simulator operates in System-call Emulation (SE) mode and Full-System (FS) mode. In SE mode, gem5 emulates common system calls. This mode is particularly useful for simulating specific functionalities or applications without requiring a complete operating system environment. In FS mode, gem5 provides a comprehensive simulation environment that mimics a booting OS and running an application on top of it. gem5 supports various CPU models, such as AtomicSimple, TimingSimple, InOrder, and

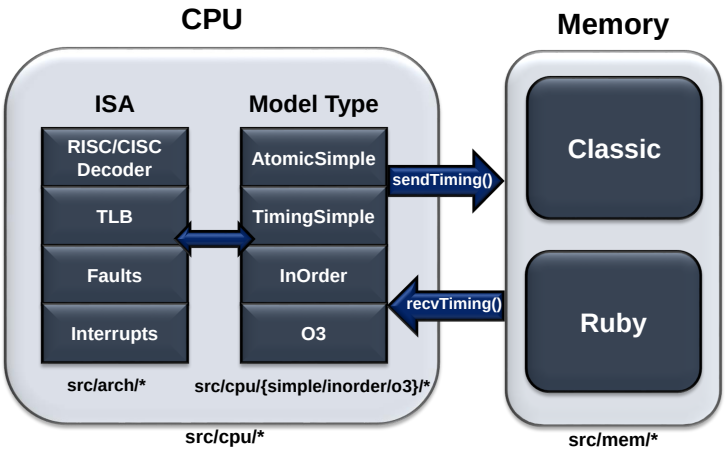


Figure 2.10: System-level view of gem5

Out-Of-Order (O3), each designed to address specific simulation needs. The AtomicSimple model executes all necessary operations for an instruction on every CPU tick. It ensures atomic memory access, making it the fastest for functional simulation. In TimingSimple, memory accesses use timing paths, and the CPU waits until the memory access returns. This model balances speed and incorporates some level of timing considerations. The Inorder model represents a detailed in-order CPU with a suitable pipeline. It can be configured to model different pipeline stages, issue widths, etc. The O3 model is a detailed out-of-order CPU designed to simulate superscalar architectures with a deeper pipeline. In terms of memory models, gem5 provides two types: Classic Memory and Ruby model. Classic Memory, inherited from the M5 simulator, is fast, flexible, and easily configurable, making it a reliable choice for memory system simulation. The Ruby memory model, inherited from the GEMS simulator, is versatile. It can model various coherence implementations such as broadcast, directory, token, and region-based coherence. For interconnection networks, gem5 provides detailed NoC modelling, including router micro-architecture, resource contention, and flow control timing using the Garnet module coupled with the Ruby memory model.

The gem5 simulator is generally organized into distinct modules each serving a specific purpose. Some of the modules are shown in Figure 2.11 whose functionalities are described as follows.

- **build:** This module contains the ISA created or developed within the gem5 framework.
- **configs:** The configs module contains the simulation configuration scripts for the setup and defines various configuration parameters for gem5 simulations.
- **ext:** The ext module encompasses external tools that gem5 supports. These tools may extend

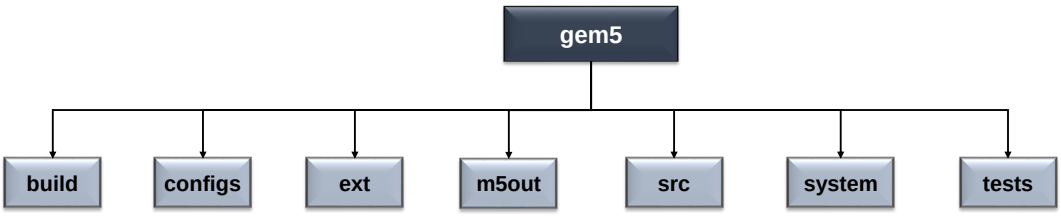


Figure 2.11: Overview of gem5 design modules

the functionality or integrate with gem5 for various purposes.

- **src:** The source code of gem5 is contained in the src module. It represents the core of gem5, such as the types of ISAs, CPU, memory models supported, and interconnection models, thus providing the implementation of its features, algorithms, and simulation logic.
- **system:** Within the system module, gem5 includes bootloaders that are utilized in simulated systems. This is essential for emulating the startup processes of complete systems.
- **tests:** The tests module contains files used for testing gem5.
- **m5out:** The m5out directory stores the output of simulation statistics in the form of comprehensive statistics.

2.8.3 McPAT

McPAT (Multicore Power, Area, and Timing) is an integrated power, area, and timing modelling framework used in multithreaded and multicore/many-core processors [87]. McPAT is known for its flexibility, allowing users to specify low-level configuration details while also providing default values for high-level architectural parameters. The framework employs an XML-based interface for communication with performance simulators, enabling the specification of static microarchitecture configuration parameters and the passage of dynamic activity statistics. These parameters include fundamental elements such as frequency, Vdd (supply voltage), in-order or out-of-order execution, cache size, NoC type, core count, and multithreaded configurations. McPAT operates independently from the simulator, reading performance statistics and, if needed, sending runtime power dissipation information back to the simulator through the XML interface. The key components of McPAT include an optimizer for determining circuit-level implementations and an internal chip representation for analyzing power, area, and timing. These statistics are essential for understanding how efficiently the hardware resources are utilized during operation.

Users can specify target clock frequency, area, and power deviation, guiding McPAT in exploring the optimization space. This thesis uses McPAT power analysis to assess various power-related metrics for NoC routers. This analysis is based on the 65nm technology node, providing an accurate representation of the impact of the design choices. Specifically, we provide the configuration file of the NoC design being evaluated as input to McPAT and analyse the power for different hardware components. With the help of McPAT's power analysis, a comprehensive evaluation of power metrics can be done, providing crucial insights for optimizing power efficiency in NoC router designs. In this thesis, we use McPAT to analyse the peak dynamic power and runtime dynamic power of the various architectures under evaluation. With the help of peak dynamic power analysis, we can analyse the maximum power consumed during dynamic activities. Runtime dynamic power represents power consumed during the active operation of the router. In NoC routers, runtime dynamic power considers power during data packet transmissions and routing decisions.

2.8.4 Architectural Parameters

Our implementation is based on a conventional 2-stage pipelined input-buffered NoC router. To construct the model of various NoC architectures used in the thesis and perform the experimental evaluation, we use the Garnet framework within gem5's ruby memory model [85]. The system configurations used in this thesis are provided in Table 2.3. For experimental analysis, we use an 8×8 2D mesh NoC. Each tile within this TCMP consists of an advanced Out-of-Order CPU core. The clock speed of each core is fixed at 3 GHz. Each core has a private L1 cache with a size of 32KB, a 4-way set-associative, and a cache line/block size of 64B. The shared L2 cache is 256KB, is 16-way set-associative, and has a cache line/block size of 64B. The system uses the MESI (Modified, Exclusive, Shared, Invalid) protocol for cache coherence with a CMP (Directory-based) approach. For inter-core communication, we follow either XY and odd-even routing algorithms. We use a one-cycle link delay and a two-cycle router delay in our simulations. NoC has an inter-router link bandwidth is 128 bits. Hence, control packets are transmitted as single flits, while data packets consist of five flits. Since the cache block size is 64 B, we need four body flits to send them across an NoC of 128-bit flit channel. Furthermore, we use a VC buffer size of four.

2.9 Application and Workloads

In TCMPs, workloads denote the set of tasks or applications running on distinct cores, encompassing the operations and processes a system manages within a specific time frame. The nature of these

Table 2.3: Standard system specification considered for experimental evaluation.

Name	Specifications
Processor and Cache	
Processor:	64 OoO x86 cores
Processor frequency:	3 GHz
L1 cache per core:	32KB, 4-way associative, 64B block, private, split (Instruction, Data)
L2 cache:	256 KB, 16-way associative, 64B block, shared
Coherence:	MESI CMP directory protocol
NoC	
Topology:	8×8 2D mesh
Router Pipeline	XY-DOR Algorithm / Odd-Even Adaptive Routing Algorithm
	2-stage, VC based wormhole packet-switching
Flit size	128 bits
Packet size:	1-flit control packet and 5-flit data packets
VC buffer size:	4

workloads can vary significantly based on the application or system. They are characterized by factors such as computational intensity, memory usage, and network communication. Creating representative workloads is essential for evaluating and understanding system behaviour under diverse conditions. Usually, in computer architecture research, workloads are often crafted using combinations of standardized benchmark suites widely accepted in the scientific community. We focus on utilising synthetic traffic patterns as well as multi-programmed, single-threaded benchmarks.

2.9.1 Synthetic Traffic Patterns

The primary goal of synthetic traffic patterns is to mimic the communication patterns observed in real applications closely. As cores generate and send packets, the traffic pattern guides the destination of these packets in a manner that simulates real-world communication scenarios. One of the key parameters in synthetic traffic simulation is the choice of injection rate, which acts as the key parameter in determining the frequency at which cores inject packets into the network. For instance, by specifying an injection rate, such as 0.2 (equivalent to a 20% injection rate), each core engages in packet creation every 20 or 5 cycles.

Various synthetic traffic patterns are available to mimic specific communication behaviours found in real-world applications. The functionality of these traffic patterns is also discussed.

- **Bit Complement Pattern:** In the Bit Complement traffic pattern, the destination core ID is calculated as the complement of the source core ID, thus simulating a request-reply communication pattern or collaborative processes.

- **Bit Reversal Pattern:** In the Bit Reversal traffic pattern, the destination ID is determined by reversing the source ID. This also simulates a request-reply communication pattern.

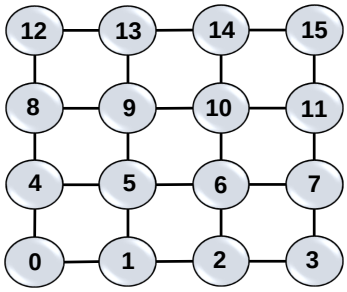


Figure 2.12: 4x4 mesh NoC with routers numbered from 0 to 1

- **Transpose Pattern:** The Transpose traffic pattern mimics a scenario in a core at the ith row and jth column will send packets to the core at the jth row and ith column. This rule simulates request-reply communication where cores exchange information like matrix transposition.
- **Shuffle Pattern:** In the Shuffle traffic pattern, the destination ID is determined by a left circular shift of the source ID. This pattern is particularly interesting as it represents a set of collaborating processes.
- **Uniform Random:** In a Uniform Random traffic pattern, any core can be selected as the destination with equal probability. This pattern reflects a scenario where communication is not influenced by specific rules or relationships between cores.
- **Tornado Traffic Pattern:** In a Tornado traffic pattern, the destination core is determined by the source core's position. Specifically, the destination is (k-1)/2 steps to the right and (k-1)/2 steps above the source core. Here, k represents the dimension of the NoC. This pattern introduces a structured yet calculated communication behaviour, resembling the spiralling motion of a tornado.

Table 2.4: Synthetic traffic communication pattern

Synthetic Traffic Pattern	Source Core	Destination Core
Bit complement	1 [0001]	14 [1110]
Bit Reversal	1 [0001]	8 [1000]
Transpose	1 [0001]	4 [0100]
Shuffle	1 [0001]	2 [0010], 4 [0100], 8 [1000]
Uniform	1 [0001]	Random(15 [1111])
Tornado	1 [0001]	6 [0111]

Table 2.4 provides an overview of the communication patterns exhibited by synthetic traffic when executed on a 4x4 mesh NoC, as illustrated in Figure 2.12. Assuming the source is core

Table 2.5: SPEC CPU 2006 benchmark suite in the context of this thesis

SPEC CPU Benchmarks	General Category	Programming Language	Misses Per Kilo Instructions (MPKI)
—Integer Benchmarks—			
mcf	Combinatorial Optimization	C	High
libquantum (lib)	Physics / Quantum Computing	ISO/IEC	High
xalancbmk	XML Processing	C++	High
perlbench	Programming Language	C	Low
bzip2	Compression	C	Low
hmmer	Search Gene Sequence	C	Low
h264ref	Video Compression	C	Low
sjeng	Artificial Intelligence: Chess	C	Medium
astar	Computer Games, Artificial Intelligence	C++	Medium
omnetpp	Discrete Event Simulation	C++	Medium
—Floating Point Benchmarks—			
leslie3d	Fluid Dynamics	Fortran	High
GemsFDTD	Computational Electromagnetics	Fortran	High
lbm	Fluid Dynamics	C	High
povray	Computer Visualization	C++	Low
gromacs	Biochemistry / Molecular Dynamics	C, Fortran	Low
cactusADM	Physics / General Relativity	C, Fortran	Low
namd	Biology / Molecular Dynamics	C++	Low
zeusmp	Physics / Magnetohydrodynamics	Fortran	Medium
sphinx	Speech Recognition	C	Medium
soplex	Simplex Linear Program (LP) Solver	C++	Medium

number 1, in the case of bit-complement traffic, the destination core is chosen as 14 (complement of 1). Similarly, for bit-reversal traffic, the destination core is set to 8 (the reverse of 1). For Transpose traffic, the destination core is set as 4. However, in shuffle traffic, a group of destinations is set for packet traversal (i.e., from 1 to 2, then 4, 8, and finally back to 1). In uniform random traffic, the destination core for the same source (1) is randomly selected from any core in the network. Meanwhile, Tornado traffic sets the destination as 6.

2.9.2 SPEC CPU Benchmarks

While synthetic traffic patterns mimic the traffic patterns that may happen in TCMPs, they may not accurately reflect the system's performance under real-world application executions. Even though these synthetic traffic patterns can generate the behaviour system behaviour in certain aspects, they are not standardized or widely accepted in the research community. On the other hand, SPEC CPU benchmarks developed by the Standard Performance Evaluation Corporation (SPEC) consist of standardized benchmark programs which are designed to mimic real-world applications and workloads. They encompass a diverse set of applications (including both single-threaded and multi-threaded tasks) ranging from compilers to scientific simulations, encryption algorithms, and image processing, thus ensuring that the benchmark results reflect the system's performance under

Table 2.6: Overview of workloads patterns explored in the thesis

Workload	Workload Pattern: Name of the Benchmark (Number of Instances)								Workload Characteristics
WH1	leslie3d (16)		lbm (16)		libquantum (16)		mcf (16)		100% High MPKI mixes
WH2	GemsFDTD (16)		lbm (16)		xalancbmk (16)		gobmk (16)		
WH3	leslie3d (16)		lbm (16)		GemsFDTD (16)		mcf (16)		
WL1	povray (16)		namd (16)		h264ref (16)		bzip2 (16)		100% Low MPKI mixes
WL2	cactusADM (16)		gromacs (16)		perlbench (16)		hmmer (16)		
WL3	h264ref (16)		bzip2 (16)		cactusADM (16)		gromacs (16)		
WM1	soplex (16)		zeusmp (16)		omnetpp (16)		sjeng (16)		100% Medium MPKI mixes
WM2	astar (16)		sjeng (16)		omnetpp (16)		sphinx (16)		
WM3	soplex (16)		astar (16)		soplex (16)		astar (16)		
WHL1	leslie3d (8)	lbm (8)	libquantum (8)	mcf (8)	bzip2 (8)	h264ref (8)	namd (8)	povray (8)	50% High MPKI, 50% Low MPKI
WHL2	cactusADM (8)	cactusADM (8)	perlbench (8)	perlbench (8)	GemsFDTD (8)	GemsFDTD (8)	xalancbmk (8)	xalancbmk (8)	
WHL3	leslie3d (8)	bzip2 (8)	cactusADM (8)	cactusADM (8)	h264ref (8)	GemsFDTD (8)	lbm (8)	mcf (8)	
WHM1	leslie3d (8)	lbm (8)	libquantum (8)	mcf (8)	soplex (8)	zeusmp (8)	omnetpp (8)	sjeng (8)	50% High MPKI, 50% Medium MPKI
WHM2	GemsFDTD (8)	xalancbmk (8)	GemsFDTD (8)	xalancbmk (8)	astar (8)	omnetpp (8)	astar (8)	omnetpp (8)	
WHM3	leslie3d (8)	GemsFDTD (8)	omnetpp (8)	omnetpp (8)	sjeng (8)	soplex (8)	astar (8)	astar (8)	
WLM1	soplex (8)	zeusmp (8)	omnetpp (8)	sjeng (8)	bzip2 (8)	h264ref (8)	namd (8)	povray (8)	50% Low MPKI, 50% Medium MPKI
WLM2	astar (8)	omnetpp (8)	astar (8)	omnetpp (8)	cactusADM (8)	perlbench (8)	cactusADM (8)	perlbench (8)	
WLM3	sjeng (8)	bzip2 (8)	sphnix (8)	sphnix (8)	h264ref (8)	astar (8)	gromacs (8)	cactusADM (8)	

real-world conditions. SPEC CPU benchmarks are widely accepted in the industry and academia as a standard measure of CPU performance. Notably, SPEC CPU benchmarks, including the widely recognized SPEC CPU 2006 benchmarks [88], serve as an industry standard for evaluating CPU-intensive performance. They rigorously evaluate a system's processor, memory subsystem, and processor efficiency, making them a common reference point for comparing and reporting CPU performance within the industry.

As the thesis primarily focuses on evaluating the performance of various NoC router architectures under different network loads, we utilize the Misses Per Kilo Instructions (MPKI) metric as a key parameter for classifying the benchmarks for workload creation. MPKI quantifies the frequency of cache misses per thousand instructions executed and is commonly used to categorize applications based on their cache miss rates. Accordingly, we choose twenty benchmarks, comprising ten from the integer suite and ten from the floating-point suite. To ensure a comprehensive evaluation, the selected SPEC CPU benchmarks are classified into three categories based on their MPKI values: High MPKI (greater than 40), Medium MPKI (between 20 and 40), and Low MPKI (less than 20).

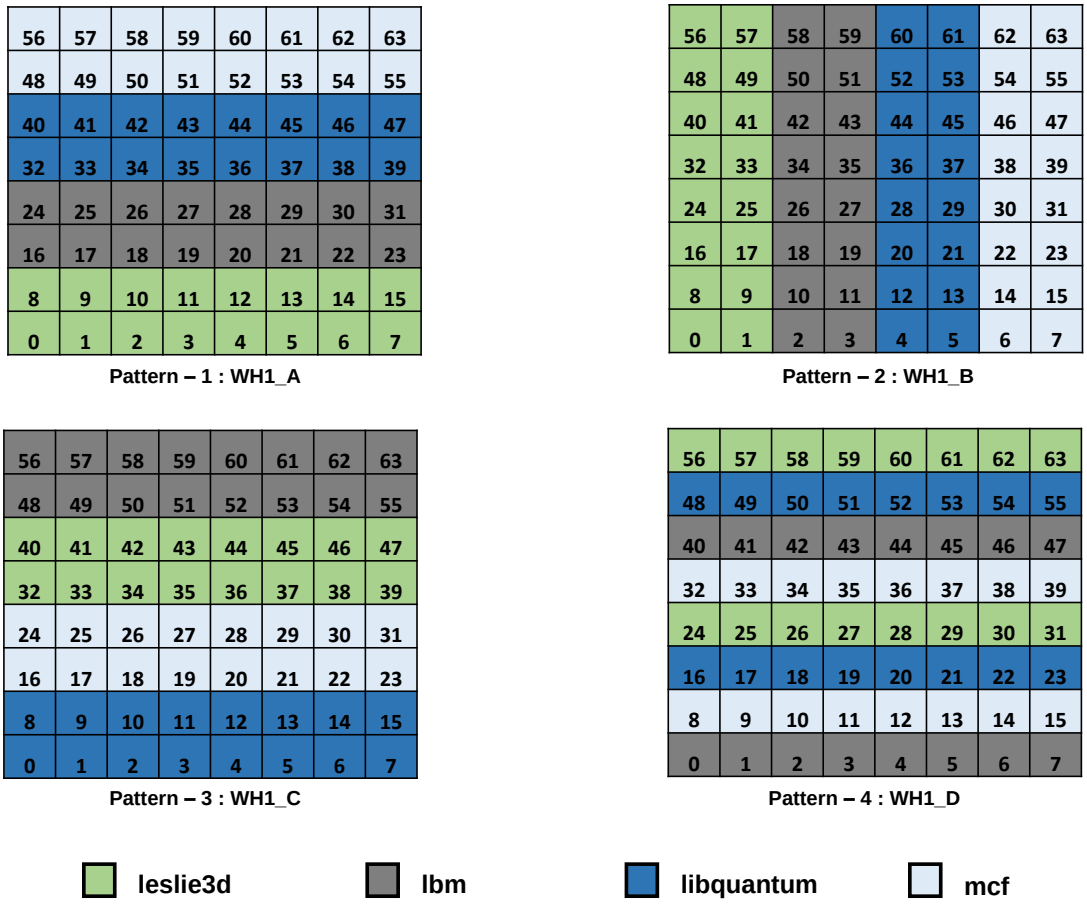


Figure 2.13: Four different mapping pattern considered for the workload WH1. WH1 consists of 4 benchmarks (leslie3d, lbm, libquantum, and mcf)

Consequently, we select six benchmarks from the High category (three from integer and three from the floating point SPEC suite), eight benchmarks from the Low category (four from integer and four from the floating point SPEC suite), and six benchmarks from the Medium category (three from integer and three from floating point SPEC suite) for the evaluation. Table 2.6 represents the workload mixes created using the above-mentioned benchmarks. In total, 18 workload mixes are formulated, with benchmarks chosen from the specified MPKI categories. Three variants of workloads are created for each category, where 100% High MPKI workload mixes are labelled from WH1 to WH3. Similarly, 100% Medium MPKI from WM1 to WM3 and 100% Low MPKI from WL1 to WL3. Additionally, multi-MPKI workload mixes are created, where 50% High and 50% Low MPKI workload mixes are labelled as WHL1 to WHL3. Similarly, WHM1 to WHM3 represent 50% High and 50% Medium MPKI, and WLM1 to WLM3 represent 50% Low and 50% Medium MPKI. This variety of workload mixes enables a thorough evaluation of the system's responses

under diverse network loads and its effectiveness in managing a wide range of application types.

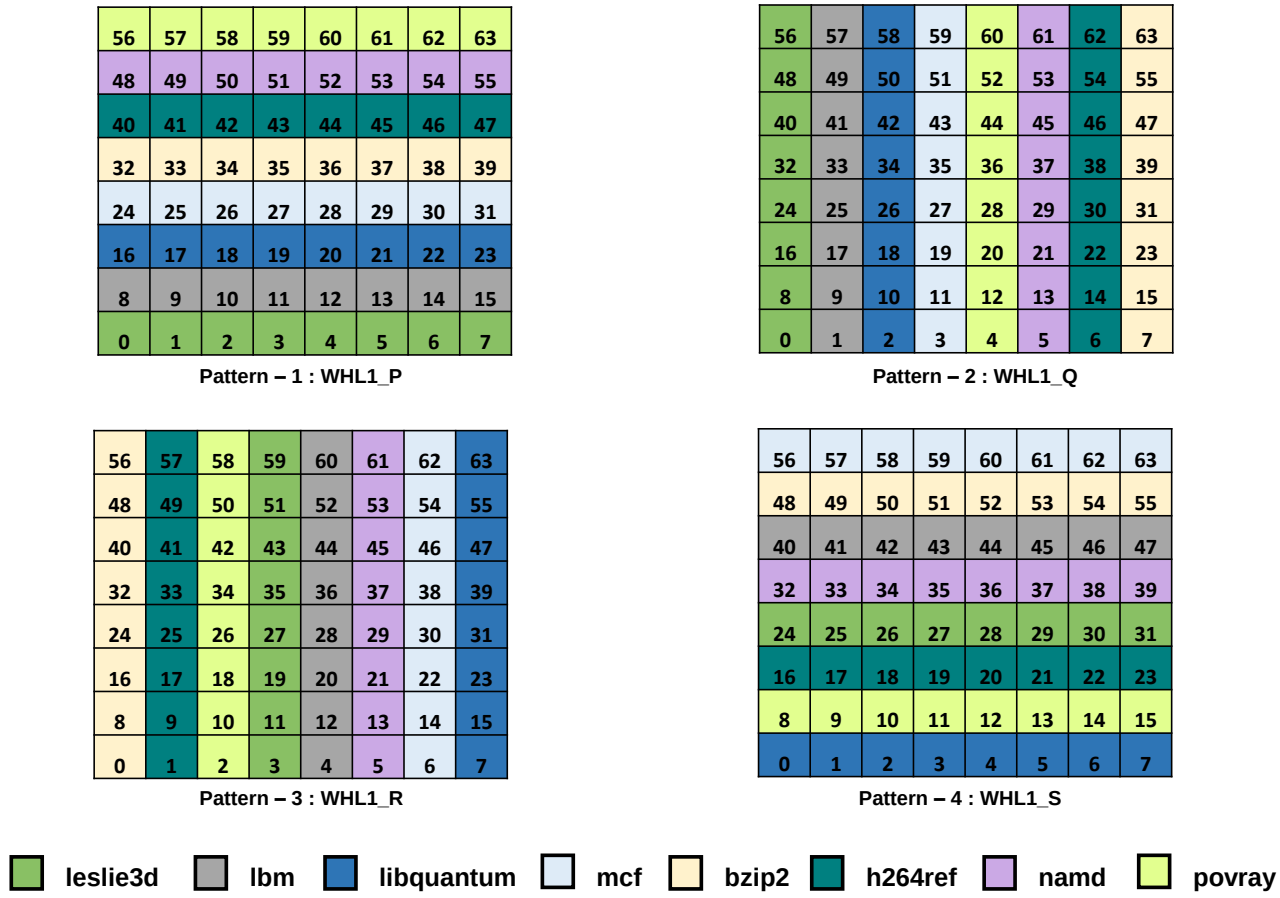


Figure 2.14: Four different mapping patterns considered for the workload WHL1. WHL1 consists of 8 benchmarks (leslie3d, lbm, libquantum, mcf, bzip2, h264ref, named, povray)

Given that the spatial organization of applications within a mesh NoC significantly influences its performance, we explore various static mappings of applications to cores in a 64-core mesh NoC. Specifically, each workload mix given in Table 2.6 is executed with four combinations of application-to-core mapping, and the mean of all these is considered for the final result. The four mapping patterns used for workload WH1 are detailed in Figure 2.13. For instance, WH1_A means the instance of leslie3d is mapped in each of the cores from 0 to 15, one instance of lbm is mapped from core 10 to 31, libquantum is mapped from core 32 to 47, and the mcf benchmark is mapped from core 48 to 63. Similarly, we explored different mapping patterns for WH1 as shown in WH1_B, WH1_C, and WH1_D of Figure 2.13. We consider the workload performance of WH1 as the mean of WH1_A, WH1_B, WH1_C, and WH1_D. This pattern mapping is replicated for WH2 and WH3. Workload WH is then considered as the mean of WH1, WH2, and WH3, as shown in Table 2.7. The

Table 2.7: Workload generation considered in the thesis

Workload	Workload Pattern
WH1	mean(WH1_A, WH1_B, WH1_C, WH1_D)
WH2	mean(WH2_A, WH2_B, WH2_C, WH2_D)
WH3	mean(WH3_A, WH3_B, WH3_C, WH3_D)
WH	**mean(WH1, WH2, WH3)**
WHL1	mean(WHL1_P, WHL1_Q, WHL1_R, WHL1_S)
WHL2	mean(WHL2_P, WHL2_Q, WHL2_R, WHL2_S
WHL3	mean(WHL3_P, WHL3_Q, WHL3_R, WHL3_S)
WHL	**mean(WHL1, WHL2, WHL3)**

Workload	Workload Pattern
WH	mean(WH1,WH2,WH3)
WL	mean(WL1,WL2,WL3)
WM	mean(WM1,WM2,WM3)
WHL	mean(WHL1,WHL2,WHL3)
WHM	mean(WHM1,WHM2,WHM3)
WLM	mean(WLM1,WLM2,WLM3)

workloads WL, WM, WHL, WHM, and WLM are generated following a similar approach. For workloads consisting of 8 benchmarks, we consider the mapping pattern (considered for workload WHL1) given in Figure 2.14 and generate results for WHL, WHM, and WHM as shown in Table 2.7.

2.10 Performance Metrics

Performance metrics are fundamental in all stages of system architecture design, from initial concept and evaluation to implementation, optimization, and validation. They provide quantitative measures to assess and improve the performance, efficiency, and reliability of computing systems. In this section, we discuss some of the performance metrics considered more significant for evaluating and analyzing the effectiveness of each proposed architecture.

2.10.1 Average Packet Latency

In NoC, Average Packet Latency (APL) plays a crucial role, directly impacting the overall performance and efficiency of the network. APL is defined as the number of cycles a packet takes to reach its destination. APL can be represented as follows:

$$APL = \frac{Network\_Latency + QueuingLatency}{No\_of\_Packets\_Received} \tag{2.1}$$

where network latency is defined as the time it takes for a packet to travel from the source router to the destination router in NoC. It includes propagation delay, transmission delay, and processing delay on a packet as it traverses the network. Queuing latency is the delay that occurs when a packet is waiting in a queue before a router can process it. This delay is influenced by factors such as the queue length, the packet arrival rate, and the processing speed of the NI unit in NoC. APL significantly influences the performance of the system. Applications with real-time requirements or strict latency constraints rely on maintaining low APL to meet their QoS effectively. Reduced latency helps in meeting tight timing constraints, avoiding potential failures or performance degradation

due to communication delays.

2.10.2 Instructions Per Cycle (IPC)

We also analyze the impact of HT on the system performance by evaluating the system throughput. To observe the throughput of the application running in TCMP, we analyse the number of instructions executed per cycle (IPC). IPC is a metric that represents the average number of instructions executed per clock cycle. It provides insights into the efficiency of the system in terms of instruction execution. HTs have the capability to either delay packets or drop them entirely. This can lead to an increase in the time required to process instructions, consequently resulting in a decrease in IPC.

We can define IPC as

$$IPC = \frac{Total\_no.\_of\_instructions}{Total\_No.\_of\_cycles\_to\_complete\_the\_instructions} \tag{2.2}$$

2.11 Chapter Summary

This chapter provided a comprehensive overview of the IC supply chain, the categorization of HTs, and the HT taxonomy. It further explored the TCMP architecture and various computer architecture simulators, such as gem5, McPAT, and ProNoC. The use of Synopsys Design Compiler for RTL synthesis is also presented. Furthermore, it discussed the workloads and benchmarks utilized in the thesis, providing a detailed explanation of the experimental setup. Finally, the chapter thoroughly examined and discussed the various performance metrics employed in the thesis.

CHAPTER

3

Secure NoC by Mitigating Packet Misrouting Trojan Attacks

This chapter presents an HT model within the NoC router, designed to initiate misrouting of packets, resulting in DoS attack and injection suppression. Subsequently, the chapter discusses a dynamic shielding technique capable of isolating the HT-infected router. We introduce a secure routing algorithm to navigate around the NoC router affected by the HT.

3.1 Introduction

Among the vulnerable components in a TCMP, NoC is particularly significant due to its role in facilitating communication between cores, thereby granting access to various system components [9][20][21] . Consequently, any malicious circuits in an NoC IP can affect the entire system's performance. For instance, an HT that misroutes packets can consume network resources without contributing to useful data transmission, effectively reducing the overall throughput of the NoC. This inefficiency can limit the system's ability to handle high-bandwidth applications or multiple concurrent tasks. Researchers explored the possibility of a runtime detection algorithm for packet misrouting HT attacks [60]. This approach uses the incoming direction of the packets to detect the location of such HTs. 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 may give a better action response.

In this chapter, we make the following contributions:

- We implement an HT model which is designed to trigger packet misrouting in NoC routers, resulting in the DoS attack and injection suppression.
- To counteract the impact of the HT, we propose a Trojan-aware routing framework capable of dynamically identifying misrouting caused by the HT. The framework establishes a dynamic protective shield around the router affected by the HT. Additionally, we introduce a bypass routing algorithm to mitigate the effects of an HT.
- Our experimental results showcase the effectiveness of our approach in successfully mitigating DoS attacks and injection suppression.

The remainder of this chapter is structured as follows: In Section 3.2, we discuss the routing algorithm utilized for this work. Following that, Section 3.3 outlines the proposed threat model and its implications for TCMPs. Subsequently, Section 3.4 presents the mitigation technique employed for addressing HT-induced packet misrouting. The performance analysis conducted for this work is discussed in Section 3.5. Section 3.6 presents the analysis of area and power overhead. Finally, Section 3.7 provides a summary and conclusion for this chapter.

3.2 Dimension Order Routing

The routing scheme employed in an NoC router plays a critical role in network channel load balancing and packet latency, impacting overall system performance. The fundamental task in the network layer of an NoC involves designing the routing algorithm and selecting the most suitable paths. Routing can be broadly categorized into oblivious routing and adaptive routing [89]. In oblivious routing, a predefined set of optional paths is established in advance for each source-destination pair, and every packet between these pairs must follow one of these predetermined paths. Thus, the path a packet takes is solely determined by its source-destination pair or may involve a random selection among the available options. One drawback of oblivious routing is that it does not take into account the current network state when selecting output ports, potentially resulting in an under-utilization of path diversity and the creation of congestion points in the system. In contrast, adaptive routing algorithms employ strategies that consider channel load information or any other suitable information when making routing decisions. These algorithms can be categorized as minimal or non-minimal. Minimal routing algorithms select the shortest path to the destination. Turn model routing protocols [90] are recent developments in routing where routing decisions are

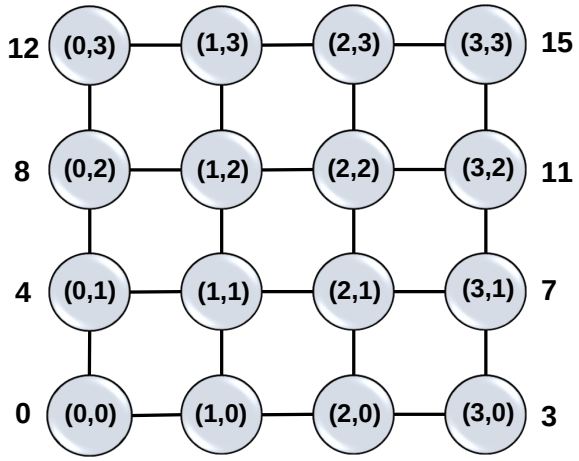


Figure 3.1: A two dimension 4X4 mesh NoC topology

based on the turns taken along the routing path. These protocols are designed to prevent packet routes that may lead to formation of cycles, livelock, and deadlock in the network.

One of the subsets of oblivious routing algorithms is the Deterministic Routing algorithm [89]. In NoC systems, deterministic routing algorithms consistently opt for the same path when routing data between a pair of routers. This predictable behaviour can lead to an uneven distribution of network traffic, causing some paths to become congested while others remain underutilized. Despite this drawback, many designers choose deterministic routing for its simplicity and predictability. Moreover, deterministic routing inherently avoids deadlock situations, ensuring smooth data flow within the network. Dimension Order Routing (DOR) is one such deterministic routing algorithm that selects specific dimensions (X or Y) to route the packets, thus following a specific pattern. One of the DOR algorithms is XY, which makes routing decisions based on a comparison of NoC router coordinates at each intermediate router. Algorithm 1 represents the XY routing algorithm. Consider a 2-dimensional 4×4 mesh NoC given in Figure 3.1. Here, each router is identified by its coordinates. For ease of reference, let (x, y) represent the coordinates of an NoC router. With the XY routing algorithm in action, whenever a packet reaches an NoC router, it compares the current router's address (Cx, Cy) with the destination router's address (Dx, Dy) stored in the header flit of the packet. The rest of the routing process, as per the XY routing algorithm, is given in Algorithm 1, which involves the following steps:

1. If the current router's address (Cx, Cy) matches the destination router's address (Dx, Dy), the flits are routed to the local port of the router.

2. If the horizontal address (Cx) of the current router does not match the horizontal address (Dx) of the destination, the routing decision is based on the comparison of Cx and Dx. Flits will be

ALGORITHM 1: Working of XY routing algorithm

Source Router: (Sx,Sy)

Destination Router: (Dx,Dy)

Current Router: (Cx,Cy)

Output: Output direction of a flit: $outv\_dir$

if *Dx* > *Cx* **then**
  set $outv\_dir$ as EAST

else if *Dx* < *Cx* **then**
  set $outv\_dir$ as WEST

else if *Dx* == *Cx* **then**

  if *Dy* < *Cy* **then**
    set $outv\_dir$ as SOUTH

  end

  else if *Dy* > *Cy* **then**
    set $outv\_dir$ as NORTH

  else if *Dy* == *Cy* **then**
    The current router is the destination router

directed to the East port when Cx is less than Dx, to the West port when Cx is greater than Dx, and if Cx equals Dx, the header flit is already horizontally aligned.

3. If the horizontal alignment condition is true, the vertical address (Dy) is compared to the current router's vertical address (Cy). Flits are routed to the South port when Cy is less than Dy and to the North port when Cy is greater than Dy.

Let P be a packet with source $S(x1, y1)$ and destination $D(x2, y2)$. As per X-Y routing, when P reaches an intermediate router $R(x, y)$, it is forwarded along the X direction until $(x < x2)$. When P reaches a router where $(x == x2)$, it changes the direction and starts travelling along the Y direction until $(y < y2)$. When P reaches a router where $(y == y2)$, it reaches the destination $D(x2, y2)$. For example, in 4×4 NoC shown in Figure 3.1, consider core 0 wants to send a packet to core 15. With XY routing, these packets move in X direction till they reach router 3. From there, it moves in Y direction (upwards) to reach its destination 15.

3.3 Threat Model: Trojan Design and its Impact

In this section, we present the design of the proposed packet misrouting HT in TCMPs with underlying routing algorithm as XY. The section also covers how such misrouting HT can affect the system's performance.

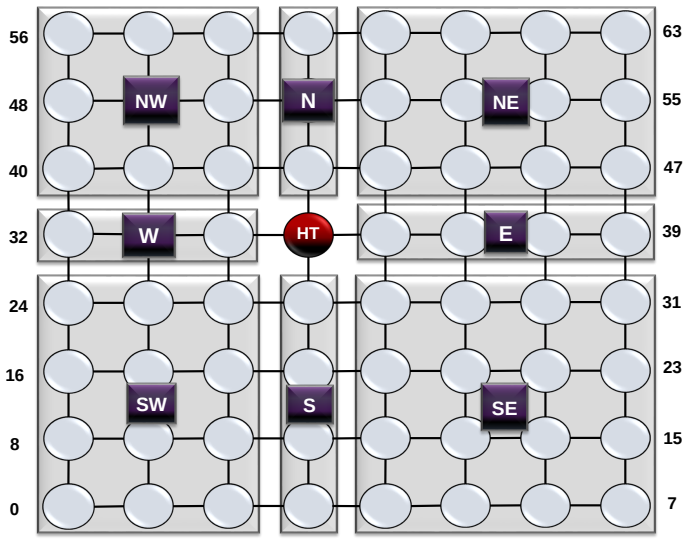


Figure 3.2: 8×8 mesh NoC with an HT at router 35

3.3.1 Packet Misrouting Trojan Design

We present an HT threat model that alters the routing algorithm (XY) employed in the Route Computation unit of the NoC router to enable packet misrouting. When triggered, the HT maliciously assigns a wrong output port to the head flit of a packet. Since we follow wormhole routing, all the flits of the packet get misrouted and contribute to DoS and injection suppression. DoS is a scenario where the flits of a packet get indefinitely delayed in the path and never reach their destination. An injection suppression scenario is a by-product of DoS where new flits cannot be injected into the network due to the unavailability of router buffers.

The X-Y dimension order routing algorithm decides the output port for a packet based on the position of the destination router with respect to the current router. This routing algorithm does not consider the input port of the packet and its previous router for its routing decisions. To model our HT model, we exploit this feature of the routing algorithm and enable misrouting. Now, even if a packet is misrouted and reaches a router where it should not have reached as per X-Y routing, the employed routing algorithm cannot detect it. The packet is forwarded to the destination without knowing the misrouting that has brought the packet to this router. Packets carry cache miss requests, cache miss replies, evicted cache blocks, and coherence messages from the source to their destination through the underlying NoC. A router infected with the proposed HT can misroute these packets and degrade the application-level performance of latency-critical applications.

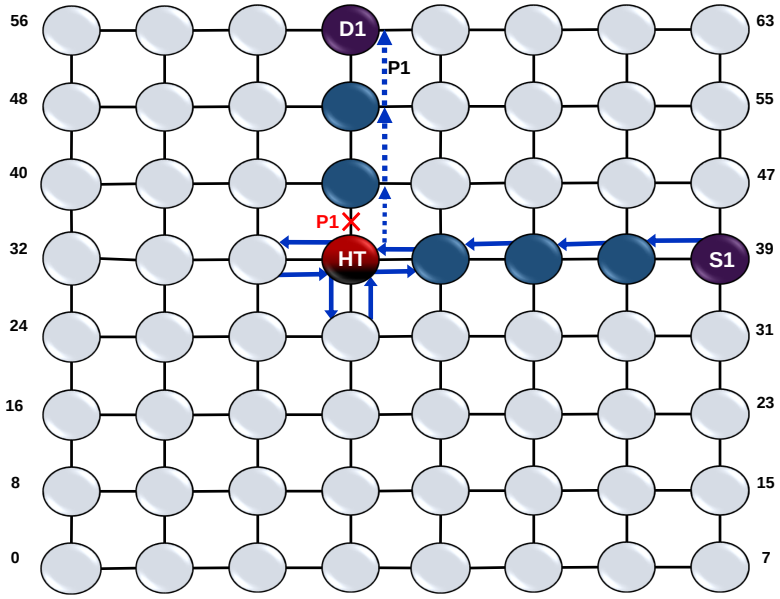


Figure 3.3: A denial of service (DoS) attack scenario -1

3.3.2 Impact of Packet Misrouting Trojan Attack

Figure 3.2 shows an illustration of an 8×8 mesh TCMP with the proposed HT mounted on router 35. An adversary can insert 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 detected. To make it hard to get detected, we assume that the adversary had infected only a single router with the proposed HT. Based on the location of HT, the entire 8×8 NoC is divided into eight different regions: N, E, S, W, NE, SE, SW and NW. When triggered, the impact of HT varies across different regions based on their inter-core communication. The possible attack scenarios due to the compromised TCMP are described in the following sections.

3.3.2.1 Denial-of-Service: Attack Scenario 1

To understand how a misrouting HT can initiate a DoS attack, consider a case shown in Figure 3.3 with the HT at router 35. The underlying NoC employed XY dimension order routing algorithm for packet traversal. During inter-core communication, a packet $P1$ with source $S1$ on its way to destination $D1$ reaches router 35. Instead of forwarding $P1$ to router 43 as per X-Y routing, the HT at router 35 misroutes $P1$ to router 34. Note that the HT can misroute this packet to any other direction than the valid one (say router 27, router 34, or router 36). When the misrouted packet reaches router 34, following X-Y routing, it will be re-sent to router 35. Destination $D1$ is at router

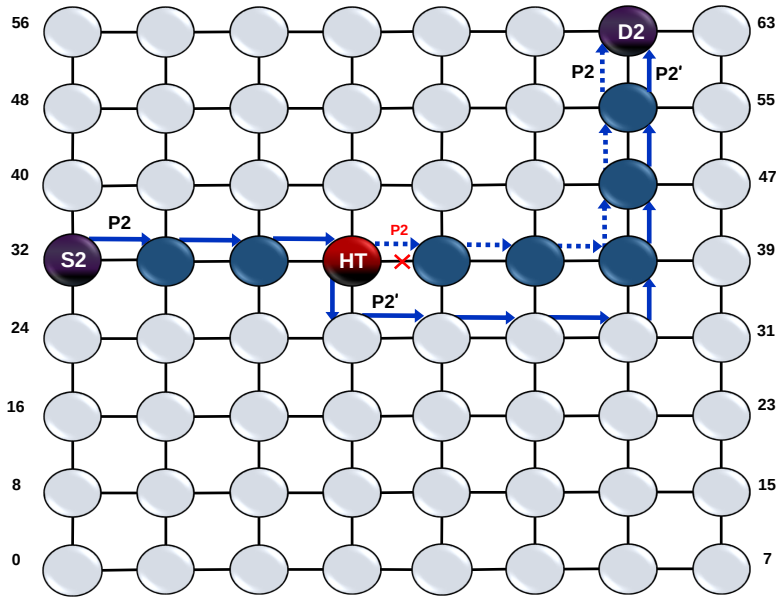


Figure 3.4: A denial of service attack scenario - 2

59, which is in the same column as that of HT (router 35). So, as per X-Y routing, $P1$ can reach destination $D1$ only through router 35, which is currently compromised. Since router 35 always misroutes, $P1$ will never reach its destination $D1$. This is a DoS attack scenario created by the proposed misrouting HT threat model discussed in this work. According to Figure 3.2, source $S1$ is in region E and destination $D1$ is in region N. With further analysis, it is concluded that an inter-region communication of type $E \rightarrow N$ leads to a DoS attack scenario. In general, for all the inter-region communication where the destination router is on the same column as that of the HT (router 35), this kind of DoS attack scenario is possible. Hence, any packet traversal between the following regions is susceptible to a DoS attack scenario:
$E \rightarrow N,\ E \rightarrow S,\ W \rightarrow N,\ W \rightarrow S\ \ NE \rightarrow S,\ NW \rightarrow S,\ SE \rightarrow N,\ SW \rightarrow N.$

3.3.2.2 Denial-of-Service: Attack Scenario 2

Another attack scenario created by the proposed HT threat model is a delay of service, which is a variant of a DoS attack. Consider an inter-core communication, where a packet $P2$ with source $S2$ is travelling towards its destination $D2$, as shown in Fig 3.4. When $P2$ reaches the HT (router 35), instead of forwarding the packet towards router 36 (as per X-Y routing), the HT at router 35 misroutes $P2$. Since the direction of misrouting is random, packet $P2$ can reach any one of the neighbours of router 35, like 27, 34 or 43. If $P2$ reaches either router 27 or 43, following X-Y routing, it can reach the destination $D2$, incurring a small delay. However, if the HT misroutes $P2$

towards router 34, then it enters a ping-pong state between router 34 and 35, something similar to the DoS scenario (refer to Figure 3.3). But, due to the randomness of misrouting, the ping-pong breaks when $P2$ gets misrouted to either router 27 or 43 in the near future. In that case, packet $P2$ eventually reaches its destination with an arbitrary delay. This is another DoS attack scenario. Again, according to Figure 3.2, source $S2$ is in region W and destination $D2$ is in region NE. Thus, an inter-region communication of type $W \rightarrow NE$ creates this kind of DoS attack scenario. To generalise, a DoS attack that ends in a packet delay-like scenario is possible when there is communication between the following regions:
$E \rightarrow W,\ E \rightarrow NW,\ E \rightarrow SW,\ W \rightarrow E,\ W \rightarrow NE,\ W \rightarrow SE.$

3.3.2.3 Injection Suppression: Attack Scenario 3

A misrouting HT can also create injection suppression in the network. Most of the time, this occurs as the byproduct of a DoS attack. We defined the HT in such a way that the direction of misrouting is non-deterministic. For example, in the case considered in Figure 3.3, the HT (router 35) can misroute packet $P1$ in different invalid directions at different instances. As a result, $P1$ gets trapped into a ping-pong state between the neighbours of router 35, except with 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 neighbouring routers and propagates the effect to others by back-pressure. Eventually, a scenario of injection suppression arises. When the traffic is high, the unavailability of NoC resources due to the ping-pong effect is also high.

3.4 TAR: Mitigation Framework for Packet Misrouting HT Attack

In order to detect and mitigate the misrouting HT in NoC, we propose a Trojan-aware Routing (TAR) technique, which can be employed in every router on NoC. TAR involves three phases: Detection, Shielding, and Bypass. The functionality of these phases is elaborated in the subsequent sections.

3.4.1 TAR Phase 1: Trojan Detection

To detect both packet misrouting and HTs, a detection module has been integrated into each NoC router. This module relies on two key variables: a 1-bit alert_flag and a 3-bit alert_dir. The alert_flag is activated only when a neighbouring router is identified as an HT router and is reset otherwise. The alert_dir can denote either no specific direction or the specific direction in which the HT infection is detected, including North, East, South, or West. The operation of this detection module is outlined

ALGORITHM 2: Working of the HT detection module in TAR

Input : Input Direction of flit; in_dir

Output : Violated output direction of a flit: $outv\_dir$

Terminology:

x_{diff}: Difference between the x coordinates of current and destination NoC router.

y_{diff}: Difference between the y coordinates of current and destination NoC router.

if $x_{diff} < 0$ *and in_dir* $is\ WEST$ **then**
| *set* $outv\_dir$ *as WEST*

else if $x_{diff} > 0$ *and in_dir* $is\ EAST$ **then**
| *set* $outv\_dir$ *as EAST*

else if $x_{diff} > 0$ *and in_dir* $is\ NORTH$ **then**
| *set* $outv\_dir$ *as NORTH*

else if $x_{diff} > 0$ *and in_dir* $is\ SOUTH$ **then**
| *set* $outv\_dir$ *as SOUTH*

in Algorithm 2. This module's core function is to ascertain whether the routing unit returns a misdirected output direction for a head flit by analyzing the difference in (x, y) coordinates between the current router and the intended destination router, as denoted by x_{diff} and y_{diff} in Algorithm 2.

To achieve this, when a head flit reaches the route computation phase, the detection module examines both the input direction of the flit and its intended destination. If the module identifies that a packet's intended output direction matches its input direction, it is reported as a case of packet misrouting. For example, if a packet is moving towards the West and its input direction is also West, this signifies that packet misrouting has occurred in the network. Subsequently, the module saves the information regarding the violated direction ($outv\_dir$ in Algorithm 2). Whenever a router identifies a packet misrouting incident, it sets its alert_flag and updates alert_dir with the direction in which the violation occurred ($outv\_dir$). This mechanism effectively aids in the detection and flagging of packet misrouting occurrences and potential HT routers within the NoC.

Illustrative Example: While considering the DoS attack scenario shown in Figure 3.3, when HT is active, the 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 the East input port from router 35. Analysing the position of destination $D1$ at router 59 with respect to router 35, the detection module concludes that X-Y 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. alert_flag and alert_dir are also used in the subsequent phases of shielding and bypassing.

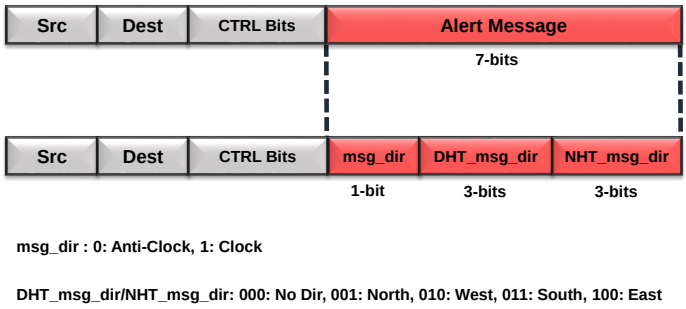


Figure 3.5: Structure of an alert_flit

3.4.2 TAR Phase 2: Shielding the Trojan

Once the HT is detected by one of the neighbours of an HT, a dynamic shielding protocol is activated. The router that detects the HT generates a special alert_flit to be sent to its neighbours about the detection of the HT. In our proposed TAR technique, such routers are known as generators. Neighbours, upon receiving the alert_flit, forward the message further by creating a propagation flit. In TAR, routers generating the propagation flits are called propagators. The structure of these special flits is very similar to normal flits, as shown in Figure 3.5. The alert_flit contains a 1-bit *msg_dir* indicating the direction, to which it needs to be forwarded by the 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. Figure 3.5 presents all the possible values for different fields of the alert_flit. When the message of HT detection is propagated among all the neighbouring routers using these alert flits and propagation flits, each router accordingly updates its alert_flag and alert_dir. This results in a shield being created around the HT that successfully isolates the HT router from the rest of the network. TAR's third and final phase uses this shielding to route packets by bypassing the isolated HT router.

The working of the dynamic shielding phase in TAR is explained using Figure 3.6. From the previous phase of HT detection, let us assume that router 34 has identified router 35 as an HT. The alert_flag in router 34 is now set to 1, and the alert_dir as 100 (East). As shown in Figure 3.6, router 34 generates two alert flits, G_N and G_S, which are sent to its North and South neighbours. 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 the 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 to router 43. When P_E reaches router 43, NHT_alert_dir = 011 (South) indicates that the HT is detected in the South

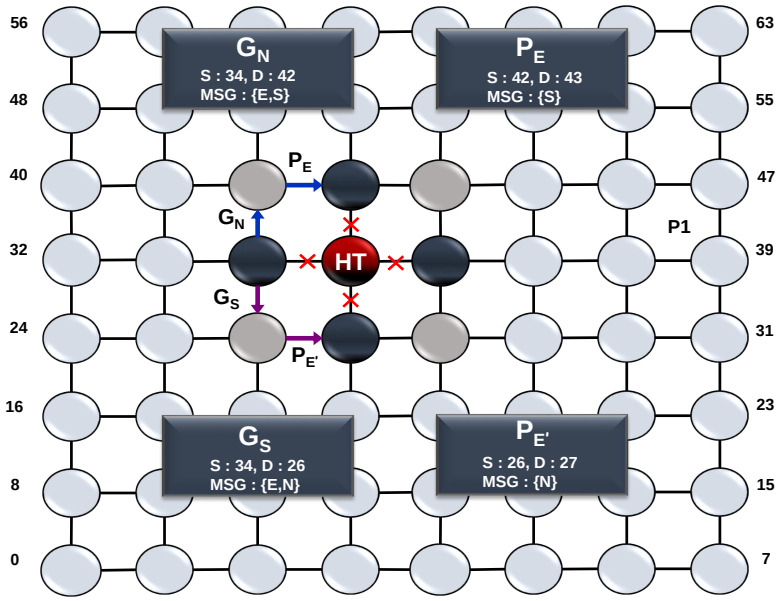


Figure 3.6: Working of dynamic shielding in TAR

direction of router 43; which is router 35. The alert_flag and the alert_dir are updated as 1 and south, respectively, in router 43, which can be a generator for other neighbours. Similarly, G_S and $P_{E'}$ (propagation flit generated by router 26) also propagate the message of HT detection to other neighbours. 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. 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 is isolated from the rest of the network.

3.4.3 TAR Phase 3: Trojan Bypassing

The final phase of TAR implements a bypass routing, as presented in Algorithm 3. When a packet arrives at a router, the 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 X-Y routing to reach its destination. The working of the Trojan bypassing phase is explained using Figure 3.7. Let's consider the same scenarios of DoS attacks shown in Figure 3.3 and 3.4 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 X-Y routing, the desired output port of packet $P1$ at router 36 is west, which matches with the alert_dir

ALGORITHM 3: Trojan Bypass Routing Algorithm

Input : Packet header

Output : Output port direction of a flit

Terminology

x_{diff}, y_{diff} : x and 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.

in_dir and $out\_dir \in$ (EAST || WEST || NORTH || SOUTH)

$alert\_dir$: HT alert direction of the current router.

$alert\_flag$: HT alert flag of the current router.

/*Part I: Mitigation by generator routers */

if $alert\_flag\ is\ SET$ **then**
- **if** $x_{diff} \neq 0$ && $y_{diff} \neq 0$ **then**
  - **if** $alert\_dir \neq EAST$ **then**
    - **if** $x_{diff} > 0$ && $in\_dir \neq EAST$ **then**
      - $out\_dir = EAST$
  - **else if** $alert\_dir \neq WEST$ **then**
    - **if** $x_{diff} < 0$ && $in\_dir \neq WEST$ **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** $alert\_dir \neq NORTH$ **then**
  - **if** $y_{diff} > 0$ && $in\_dir \neq NORTH$ **then**
    - $out\_dir = NORTH$
- **else if** $alert\_dir \neq SOUTH$ **then**
  - **if** $y_{diff} < 0$ && $in\_dir \neq SOUTH$ **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**
  - $out\_dir = EAST$

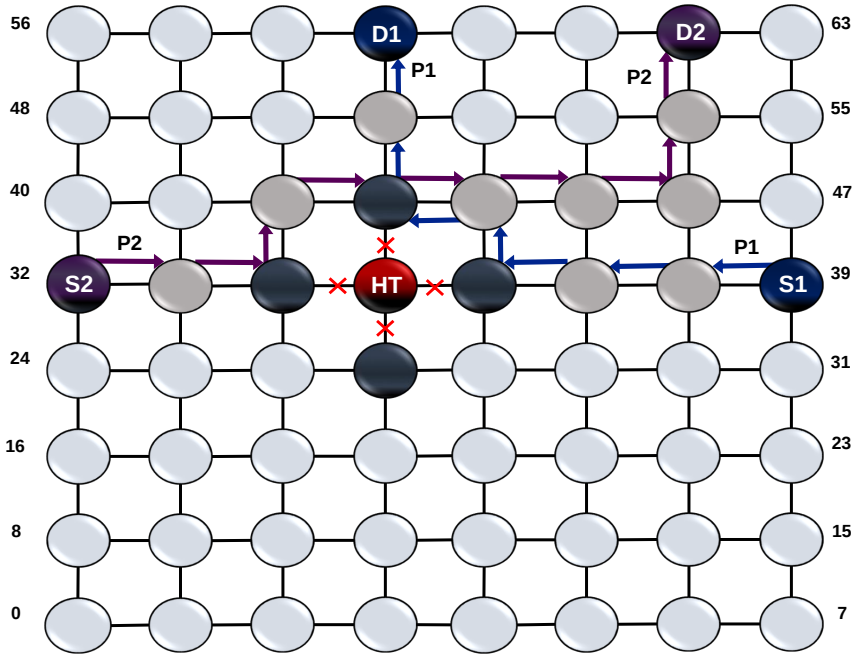


Figure 3.7: Working of Trojan bypassing in TAR

of router 36. Now, the Trojan bypass algorithm initiates and reroutes packet $P1$ away from the HT (router 35) as presented in Part I of Algorithm 3. Packet $P1$ is rerouted from router 36 to router 44, and Part II of Algorithm 3 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.

Since destination $D1$ is in the same column as that of HT (router 35), with HT activated, it becomes impossible for $P1$ to reach $D1$ using the conventional approach and hence it results in a DoS-like scenario. With the Trojan bypass algorithm in place, $P1$ can now reach its destination, thus mitigating the impact of DoS. Since packets like $P1$ are not trapped in the network anymore, the proposed bypass routing also reduces 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. Note that router 35 misroutes only those packets that are passing through it. Hence, even after bypassing is activated, the packets whose source or 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 the first few flits of some packets while the 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 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 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.

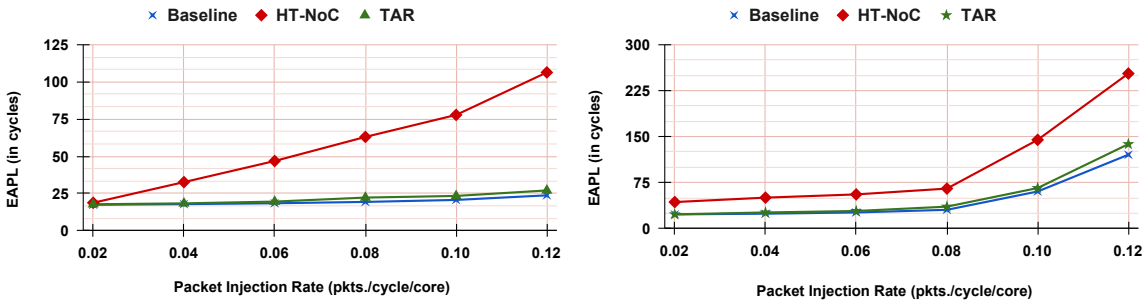
Rerouting packets using the bypass algorithm violates normal X-Y routing and creates a possibility for network deadlock. To ensure deadlock prevention, TAR employed the concept of intermediate destination [91]. When packet $P2$ is rerouted from router 34 to router 42, it starts travelling in the Y direction. However, when it travels from router 42 to router 43, $P2$ violates X-Y routing since turning X from Y direction is prohibited. Using the concept of intermediate destination, router 42 is made the new destination for packet $P2$. Now, after getting rerouted from router 34, packet $P2$ reaches router 42 and gets space in the local input port VC. This creates an effect that P1 has been freshly introduced into router 42 while simultaneously preventing the selection of YX paths, thereby eliminating the possibility of deadlock.

The HT detection framework algorithm exhibits a time complexity of O(n+m), where n represents the number of intervals processed in the Traffic Monitoring Module, and m represents the number of routers checked during cycle detection in the Path Monitoring Module. The space complexity is O(m), as it depends on the number of routers involved in the cycle detection process. This makes Trojan bypass routing algorithm is highly efficient, particularly for hardware implementation, where minimal resource usage and predictable execution times are critical.

3.5 Results and Discussions

We assess the performance of TAR by examining effective average packet latency, effective average deflected packet latency, IPC, and injection suppression avoidance. In our evaluation, we study the following architectures.

- **Baseline:** A TCMP system having no HT.
- **HT-NoC:** A TCMP with the proposed packet misrouting HT in one random router.
- **TAR:** A TCMP system that uses Trojan-aware routing to detect and mitigate the HT.



(a) Uniform random traffic

(b) Bit complement traffic

Figure 3.8: Effective average packet latency analysis using synthetic traffic patterns

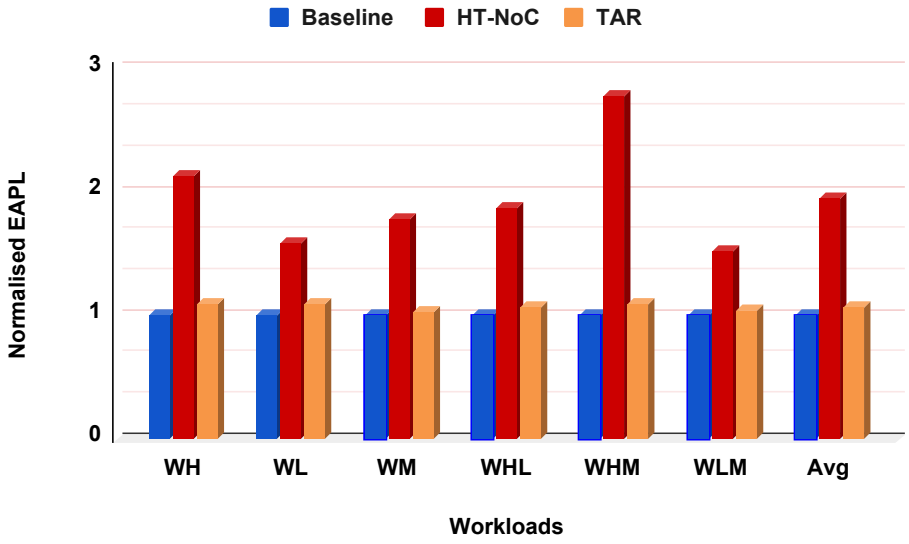


Figure 3.9: Comparison of effective average packet latency in real workloads consisting of SPEC CPU2006 workloads (Normalised to Baseline)

3.5.1 Impact on Effective Average Packet Latency

When the HT is activated, in certain scenarios, the average packet latency on an NoC shows inconsistent values. At higher injection rates, NoC with HT deploys DoS attacks that restrict the packets never reaching their destination cores can result in packet loss and injection suppression. Consequently, the APL values generated for such NoCs may not accurately reflect the network's performance. In such scenarios, we apply a more realistic metric, Effective Average Packet Latency (EAPL), presented by Jonathan et al. [77]. EAPL is defined as follows:

$$EAPL = APL * \frac{Packets\_Ejected_{withoutHT}}{Packets\_Ejected_{withHT}} \tag{3.1}$$

where the ratio of packets that are ejected without HT and packets that are ejected with HT quantifies

the relative impact of the HT on packet ejection. Multiplying the APL by this ratio effectively scales the APL based on the proportion of packets that are ejected in the presence and absence of the HT. This portrays the influence of HT-induced packet ejection on the overall latency experienced by packets in the NoC. During the analysis of the EAPL using synthetic traffic patterns, specifically the Uniform Random and Bit Complement scenarios depicted in Figure 3.8a and 3.8b, a consistent trend is observed. As the injection rate increases, packet latency also rises in all three models: Baseline, HT-NoC, and TAR. The presence of an HT router results in packet deflection, causing DoS and service delay attacks in the TCMP. Consequently, the HT system experiences a 2.8x increase in average latency compared to the Baseline in Uniform Random and a 2.12x increase in Bit Complement traffic. However, TAR achieves a reduction in EAPL when compared with HT-NoC. By employing HT bypassing for secure communication, TAR reroutes the majority of packets originally destined for the HT router through intermediate destinations, adding a few extra hops to their journey. This redirection leads to an increased EAPL when compared to the Baseline model. Simulation results indicate an 8% latency increase for TAR while applying Uniform Random traffic and 9% while using Bit Complement traffic when compared to the Baseline.

Further analysis of EAPL with real workloads presented in Section 2.9.2, we observe that the HT triggering induces a substantial increase in packet latency across all workload mixes, averaging at 1.9x higher than the Baseline, as shown in Figure 3.9. But as expected, TAR demonstrates a noteworthy reduction in EAPL, achieving a 43% decrease on average compared to the HT-NoC model. For workloads with low MPKI benchmarks (WL), TAR demonstrates a reduction in latency by approximately 31% compared to HT-NOC and by approximately 8% increase only, compared to the Baseline. Similarly, for workloads with medium MPKI benchmarks (WM) and a mix of low-medium MPKI benchmarks (WLM), TAR significantly reduces latency by approximately 42% and 31%, respectively, compared to HT-NOC, indicating its adaptability and efficiency in optimizing performance across varying cache sensitivities. Most notably, for high-intensity workloads with high MPKI benchmarks (WH and WHM), TAR achieves the lowest EAPL among all scenarios, showcasing a reduction in latency by approximately 48% and 60%, respectively, compared to HT-NoC. Despite this improvement, TAR does incur a 7% increase in latency compared to the Baseline. This latency variation is attributed to the implementation of bypass routing within the NoC routers, which contributes to the overall efficacy of TAR in mitigating the HT impact.

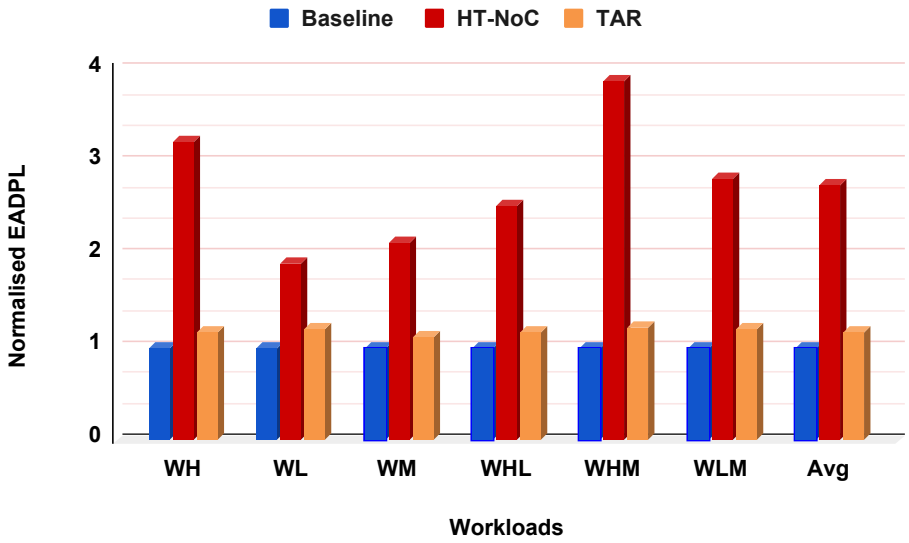


Figure 3.10: Comparison of effective average deflected packet latency in real workloads consisting of SPEC CPU2006 workloads (Normalised to Baseline)

3.5.2 Impact on Effective Average Deflected Packet Latency

Average Deflected Packet Latency (ADPL) is a metric used to measure the average latency of packets that experience deflection while travelling through HT-NoC. When calculating ADPL in a NoC without HT, the focus is on packets passing through a specific router. In the case of an HT-infected NoC, ADPL is specifically computed for packets that undergo deflection due to the presence of an HT at a particular router. This means that the ADPL metric considers the average latency only for those packets that are affected by HT-induced deflection. Similar to effective average packet latency, to get meaningful latency values, we use Effective Average Deflected Packet Latency (EADPL) which is defined as follows:

$$EADPL = ADPL * \frac{Deflected\_Packets\_Ejected_{withoutHT}}{Deflected\_Packets\_Ejected_{withHT}} \tag{3.2}$$

The presence of HT results in some packets entering a ping-pong state between neighbouring routers, causing delays before reaching their destination. Figure 3.10 shows the analysis of the HT effect on NoC while running various SPEC CPU workloads presented in Section 2.8. With HT activation, we observe a 2.7x increase in EADPL on average compared to the Baseline. For workload WHM, consisting of High and Medium MPKI benchmarks, as more packets enter into a ping-pong state between the neighbouring routers, HT induces a 3.8x increase in EADPL on average over the Baseline. TAR, incorporating a bypassing algorithm, mitigates this latency increase in

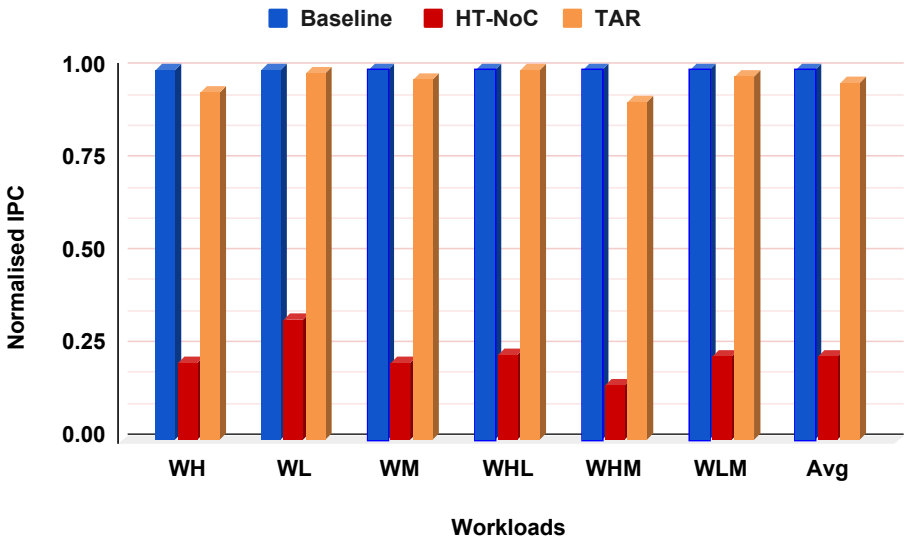


Figure 3.11: Comparison of processor performance in real workloads consisting of SPEC CPU2006 workloads (Normalised to Baseline)

deflected packets. For workloads with low MPKI benchmarks (WL), TAR demonstrates a notable decrease in EADPL by approximately 56% compared to HT-NoC and by approximately 15% increase compared to the Baseline. Similarly, for workloads with medium MPKI benchmarks (WM), TAR achieves a reduction in EADPL by approximately 47% compared to HT-NoC, reflecting its adaptability in optimizing performance across moderate cache misses. For high-intensity workloads with high MPKI benchmarks (WH and WHM), TAR exhibits substantial decreases in EADPL by approximately 71% and 60%, respectively, compared to HT-NoC. Additionally, for workloads with low-medium MPKI benchmarks (WLM), TAR demonstrates significant reductions in EADPL by approximately 58% compared to HT-NoC. Overall, due to the bypass-induced deflection, TAR shows an average EADPL increase of 17% over the Baseline.

3.5.3 Impact on Processor Performance

In every network, throughput plays a vital role in determining the QoS of the underlying applications. To analyse the throughput of the application, we analyse the IPC. Figure 3.11 shows the normalised IPC for various architectural scenarios under diverse workloads. Experimental results show that with an increase in HT-induced packet latency, there is a substantial performance degradation across various workloads, signifying the adverse effects of the HT. We observe that the average percentage decrease of IPC in HT-NoC across all workloads is approximately 77%. For workloads WH and WHM, with the increase of EAPL and EADPL, HT creates an IPC reduction of 78% and 85%. This

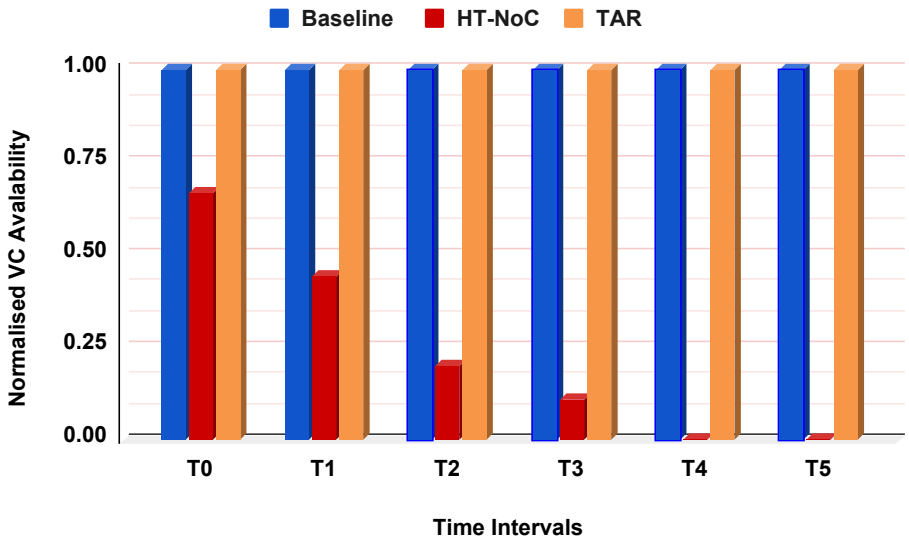


Figure 3.12: Injection suppression avoidance at NoC router (Normalised to Baseline)

reduction in throughput underscores the impact caused by the presence of the HT, highlighting its disruptive influence on the system's overall performance. Additionally, the analysis of TAR values indicates that the TAR model suffers only an average of 3% IPC reduction over the baseline. We observe that TAR achieves IPC close to the Baseline for workloads WL and WLM. However, with workloads having High MPKI benchmarks, TAR shows an IPC reduction of more than 6%, due to packet rerouting done as part of the Trojan bypass phase.

3.5.4 Impact on Injection Suppression

We study the injection suppression created by the HT on an 8×8 TCMP by analysing the buffer (VC) availability in NoC router during the simulation. The average number of input VCs available on the selected NoC router during the selected simulation cycles while simulating the WLM workload is analysed, and the results are given in Figure 3.12. For ease of reference, we map these cycle intervals from T1 to T8. Due to the ping-pong effect, the number of packets processed around the HT is very high. This can block the router VCs of HT and its neighbours, as well as subsequent back pressure, leading to injection suppression. 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 becomes zero, which indicates the injection suppression in the whole network. TAR ensures that none of the packets are under DoS attack and that the packets are deflected by their one-hop neighbour with the help of our shielding approach. This keeps the input VC availability the same as the Baseline, which prevents injection suppression in the network.

Table 3.1: Area and power overhead analysis

Metric	Baseline	TAR
Area (mm2)	1.14	1.17
Peak Dynamic (W)	1.02	1.05
Runtime Dynamic (W)	0.043	0.044

3.6 Area and Power Overhead Analysis

For analyzing the power overheads of the architectures under study, we utilize parameters from McPAT [87]. Table 3.1 shows the power overhead associated with various architectures. The TAR architecture demonstrates a 2.63% increase in area compared to the Baseline. This modest expansion is due to the additional security and monitoring circuitry integrated into TAR. Despite this slight increase, the overall impact on chip size remains minimal, indicating that the architecture delivers enhanced security with negligible area overhead. In terms of power consumption, TAR shows a 2.94% rise in peak dynamic power over the Baseline, primarily because of the extra logic and control units required during peak load conditions. Despite this, TAR maintains efficiency, making it suitable for applications where both performance and security are critical. The runtime dynamic power also exhibits a 2.33% increase, which reflects the continuous operation of security monitoring circuits. TAR's slight percentage increases in both area and power consumption are well justified by the significant security enhancements it offers, making it a viable solution for system security.

3.7 Chapter Summary

This chapter discussed the design and characteristics of the misrouting HT and its potential to disrupt the normal routing operations of deterministic NoC routers. The chapter also projected the proposed Trojan-aware routing scheme, emphasizing its efficiency in detecting and preventing the effect of misrouting HTs. The experimental results section provided a comprehensive evaluation of the proposed approach, and the area and power overhead introduced by the Trojan-aware routing mechanism were also explored.

CHAPTER

4

Secure Routing Framework by Mitigating Packet Looping Trojan Attacks

In this chapter, we explore the potential threat of an HT, targeting the NoC to launch a packet looping attack which degrades the system performance. We focus on introducing an HT model designed to execute a DoS attack within NoC that creates packet delay by exploiting vulnerabilities in the path selection strategy employed by the adaptive NoC router's Route Computation unit. We present a detection framework that leverages packet traffic analysis and path monitoring techniques to localize the presence of the HT precisely. Additionally, we propose the integration of a security wrapper module for the Route Computation unit that aims to mitigate the impact of the HT at the cost of negligible reduction in IPC.

4.1 Introduction

In TCMPs, the performance of the NoC is assessed based on the quality of service experienced by applications. Routing algorithms in NoC routers are crucial for balancing network load and affecting packet latency. Oblivious routing protocols (e.g., XY) use fixed paths and lack path diversity, while adaptive routing algorithms adjust based on channel load information [23][24]. This work explores the potential for hardware Trojan (HT) attacks targeting the routing module of NoC routers. We propose an HT circuit that alters the Route Computation logic of NoC routers, inducing delays in selected packets by disrupting the path selection strategy. While previous work has addressed HT attacks in deterministic NoC routers [56][62][92], this study focuses on adaptive NoC routers. To counteract such HT attacks, we introduce a security wrapper module, SecRC, which detects and mitigates HT effects.

Figure 4.1 provides an overview of our approach. We implement the HT in the routing unit of adaptive NoC routers. The HT is intermittent, and our detection framework uses a packet traffic monitor and a path monitor to identify HT-induced cycles. For mitigation, SecRC activates a turn violation checker and a secure routing unit to prevent packets from entering these cycles.

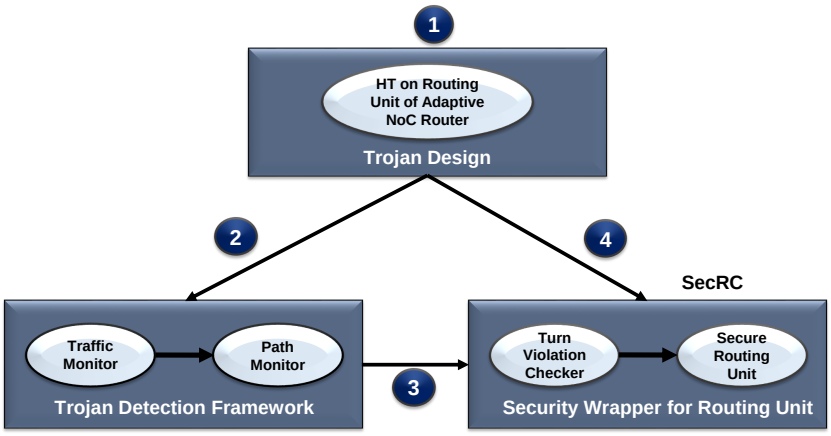


Figure 4.1: Overview of the work

Overall, in this chapter, we make the following contributions.

- The chapter highlights the modelling of an intermittent HT that modifies the output port selection strategy used in the adaptive NoC router. Once triggered, the HT deploys a DoS attack in the NoC by trapping certain packets in HT-induced loops/cycles. To our best knowledge, this work is the first to present an HT model that disrupts the path selection strategy used in the adaptive NoC router to deploy a delay of service attack.

- We propose an HT detection framework that uses a traffic monitor module to analyse the packet traffic and detect the hotspots formed by the packets that enter into HT-induced cycles.

- Once hotspots are detected, with the help of a path monitor module, we analyse the route a packet takes to reach its destination. We use this path information to localise the HT.

- We also implement a lightweight security wrapper (SecRC) module for the routing unit of the NoC router to mitigate the effect of HT. Once activated, SecRC detects packets that violate turn restrictions with the help of turn violation checker. Upon identifying an intended packet traversal, the SecRC module activates the secure routing algorithm, localising the HT and successfully suppressing the effect of HT.

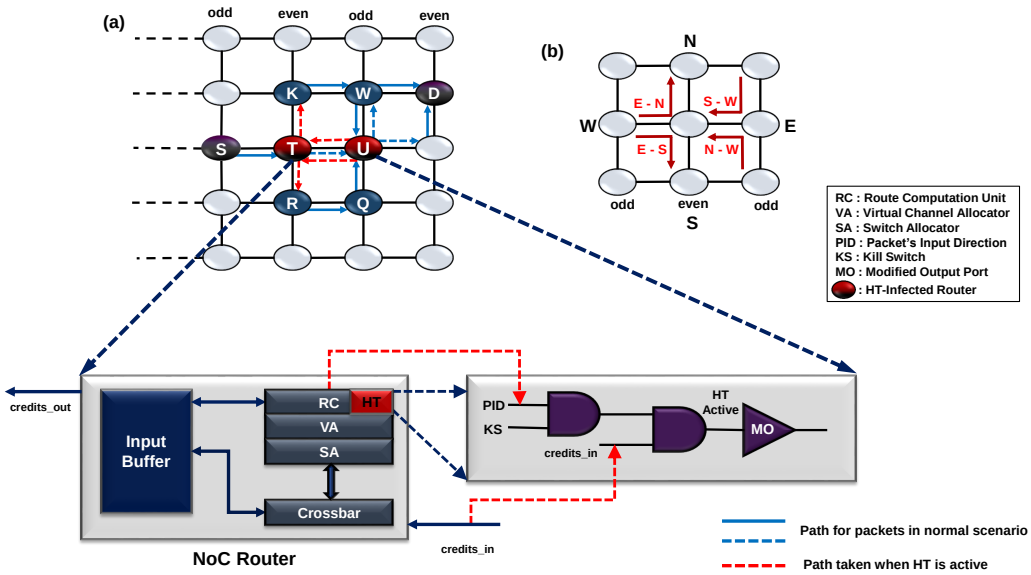


Figure 4.2: (a) Trojan design and its effect, and (b) Turn restrictions with the odd-even algorithm

The chapter is organized as follows. Section 4.2 describes the threat model. Following that, the proposed mitigation technique for packet looping HT is explained in Section 4.3. Section 4.4 explains the performance analysis. Area and power overhead analysis of the system is done in Section 4.5. Subsequently, the chapter is finally summarized in Section 4.6.

4.2 Threat Model: Trojan Design and its Impact

Our work focuses on the HT model that deploys an attack on special-purpose TCMPs used in IoTs, where, more often than not, the functionalities are fixed, and the application's traffic patterns are predictable. We assume that the CAD tools used by SoC designers are not trustworthy. Here, the adversary can be any rouge SoC designer who can integrate malicious circuits by modifying the IPs during the synthesis and verification stage of IC [47][93].

4.2.1 Packet Looping Trojan Design

To implement the attack, we integrate the HT into the Route Computation (RC) unit of an adaptive NoC router, as depicted in Figure 4.2. The HT targets the non-minimal odd-even (OE) routing algorithm used for path selection, which does not always choose the shortest path [22]. In OE routing, certain turns are restricted to avoid deadlock. Specifically, in even column routers, East $\rightarrow$ North and East $\rightarrow$ South turns are prohibited. In odd column routers, North $\rightarrow$ West and South $\rightarrow$ West turns are prohibited (see Figure 4.2b).

The OE routing algorithm operates in two phases. Phase 1 determines the possible output ports for a packet based on the routing constraints. Phase 2 selects one output port from the possible options using a selection strategy that typically considers buffer (VC) availability to avoid congestion [23][24]. The HT model is designed to disrupt this selection strategy. We formulate the HT as follows:

For a packet P with source src and destination dest, the path is:

$$P = \{R_{\text{src}}, \ldots, R_{k-1}, R_k, R_{k+1}, \ldots, R_{\text{dest}}\}$$

where R_i denotes router i in the NoC.

The OE routing algorithm O_i at router R_i can be described as:

$$O_i = \{O_i^{p_1} \parallel O_i^{p_2}\}$$

where:$O_i^{p_1}$ computes the possible output port directions for P and $O_i^{p_2}$ applies the selection strategy to choose the output port from the options given by $O_i^{p_1}$.

The HT modifies O_i to:

$$O_i = \{O_i^{p_1} \parallel O_i^{p_2*}\}$$

where:

$$O_i^{p_2*} \neq O_i^{p_2}$$

The adversary introduces a violated direction into the selection strategy:

$$O_i^{p_2*} = O_i^{p_2} \cup \{out_v\}$$

where $out_v = \{\text{North or South}\}$ for even column routers, and $\{\text{West}\}$ for odd column routers.

When activated, the HT alters the packet's output port to one of the violated directions out_v, causing the packet to enter HT-induced cycles. This manipulation increases packet latency by creating packet loops. The HT circuit is illustrated in Figure 4.2a and detailed in Algorithm 4. The HT is triggered based on the packet's input direction (PID) and the activation of a kill switch (KS) [57]. Typically, adversaries use the kill switch to prevent HT activation during logic testing. For odd column routers, the HT is activated if the packet arrives from the North or South input port, while in even column routers, it is triggered only for packets arriving from the West input port. Once activated, the HT monitors the valid output port direction assigned to an incoming packet during Phase 2 of the OE routing algorithm.

ALGORITHM 4: HT infected selection strategy in OE

Input : Output port computed by the existing selection strategy.

Output : Final output port (violated / valid).

out_dir_{sel}: Output direction given by existing selection strategy.

out_dir_{final}: Final output port selected.

/* Odd column router */

if *(*PID == *North* || PID == *South)* && KS = *True* **then**
    HT_active = True
    MO = {West}

/* Even column router */

if PID == *West* && KS = *True* **then**
    HT_active = True
    MO = random ({North, South})

/* Odd / even column router */

if *HT_active* **then**
    if $numFreeVC(out\_dir_{sel}) < numFreeVC(out\_v)$ **then**
        $out\_dir_{final} = MO$
    else
        $out\_dir_{final} = out\_dir_{sel}$

else
    $out\_dir_{final} = out\_dir_{sel}$

The HT examines credit information (credits_in) from neighboring routers to evaluate the virtual channel (VC) availability for the potentially violated output port direction (MO). This includes North or South for even column HTs and West for odd column HTs. If the VC availability of the violated direction exceeds that of the valid direction, the HT modifies the packet's output port to the violated direction (MO), causing misrouting. For example, in an even column router, an HT may redirect a packet originally destined for the East (valid direction) to the North or South (violated directions) only under specific conditions related to VC availability.

$$numFreeVC(North) > numFreeVC(East) \tag{4.1}$$

$$numFreeVC(South) > numFreeVC(East) \tag{4.2}$$

Similarly, for packets coming from the North going towards the South or packets coming from the South going towards the North, odd column HT misroutes it to the violated direction, West, only when one of the following occurs.

Table 4.1: Turn selection scenario in an even column router with HT

Input Direction	Valid Dir (V)	Malicious Direction (M)	VC Availability (Valid)	VC Availability (Malicious)	Direction Selected
West	East	North	0	1	North
West	East	West	1	0	East
West	East	North	1	0	East
South	East	West	0	1	South
East/ North/ South/ Local	East/ North/ South/ Local	-	1	-	East/ North/ South/ Local

$$numFreeVC(West) > numFreeVC(North) \tag{4.3}$$

$$numFreeVC(West) > numFreeVC(South) \tag{4.4}$$

Table 4.1 shows the possibility of selecting a malicious turn when HT is triggered in an even column NoC router. Here, the packets that are coming from the West (going towards the East) get a violated turn only when the VC availability of North or South is higher than the valid direction (East) allocated for the packet. Consequently, it can be noted that an NoC router selects the malicious output direction only when the number of VCs available in the violated direction is greater than the valid output port direction. In all other cases, when the HT is not triggered, packets travel in the valid direction given by the existing path selection strategy of the underlying routing algorithm. This justifies that the proposed HT is intermittent and hard to detect.

4.2.2 Sample Scenario for Packet Looping Trojan Attack

To investigate the effect of the attack, we consider the activation of two HTs on an 8×8 mesh NoC (one in an even column, router T and another in an odd column, router U, adjacent to router T) as shown in Figure 4.2a. Assume a scenario where a packet P, with source as S and destination as D, reaches router T from its West input port. At router T, the path selection strategy of the underlying OE algorithm returns East as the valid output direction for the packet P. Assume that, at this time, the KS is activated and HT is triggered in router T. Once activated, the HT circuit in router T compares the VC availability of router T's East port with HT's possible choice of violated output directions (in this case North, and South as HT is activated in an even router) (refer Algorithm 4). If router T finds more free VCs in the violated directions, it initiates a turn-violation to misroute the packet to either router R or router K. Assume router T sends the packet P to router R. At R, as per the selection strategy, consider that the P got East direction, reaching router Q. If the packet P gets North as the valid output port at Q, it will be forwarded to router U. Assume that the packet P, while reaching router U, gets the valid output port as North (i.e. to router W). HT in

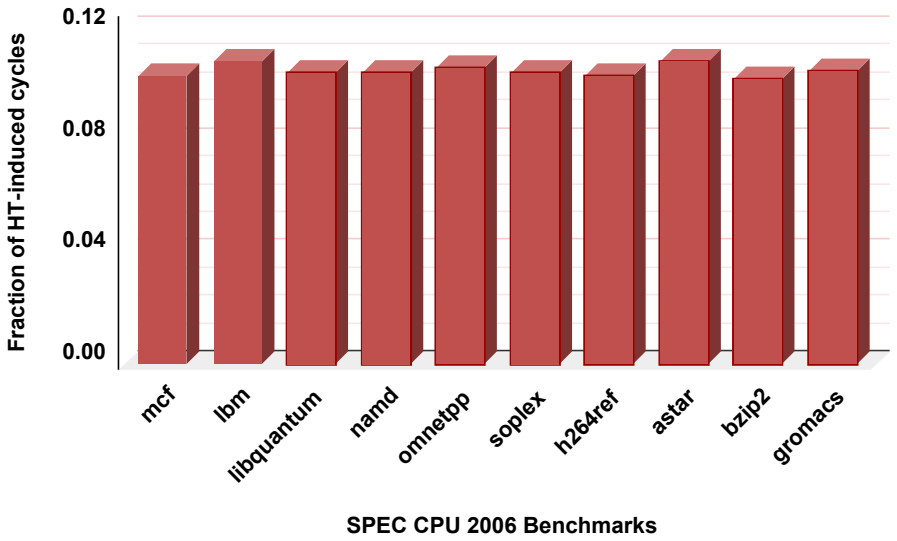


Figure 4.3: Impact of HT-induced cycles on 8× mesh NoC while running selected SPEC CPU 2006 benchmarks

router U can redirect it to router T when it finds more free VCs towards its West output direction than its North port. Consequently, P enters a cycle formed between T→R→Q→U→T. On the other hand, if router T sends the P to router K, it might get trapped in a cycle formed by T→K →W→U→T. Accordingly, P suffers a delay in reaching its destination, making it a victim of the DoS attack. Here, the number of times P iterates the cycle depends upon HTs' activation patterns and VC availability of downstream routers that are part of the cycle. To understand the occurrence of HT-induced cycles in NoC, we analyse the number of cycles generated by HT pairs in an 8×8 mesh NoC while running 64 copies of various SPEC CPU 2006 benchmarks, as shown in Figure 4.3. We observe that up to 11% of packets in the network enter into HT-induced cycles irrespective of the benchmark categories. This creates a delay in delivery for certain packets during their inter-core communication. Hence, we prove that the proposed HT is capable of DoS attack.

4.3 SecRC: Mitigation Framework for Packet Looping HT attack

To keep the NoC capable of detecting the effect of packet looping HTs, we propose an HT detection framework that can identify the irregular packet traversal pattern in the network. To achieve this, the architecture uses a traffic monitor module and a path monitor module, as shown in Figure 4.4. With the help of a traffic monitor module, a router detects malicious traffic in the network. When a router's traffic monitor module identifies a suspicious activity, it sends a notification to the neighbours to enable a path monitor module, which detects the occurrence of HT-induced cycles. We implement a

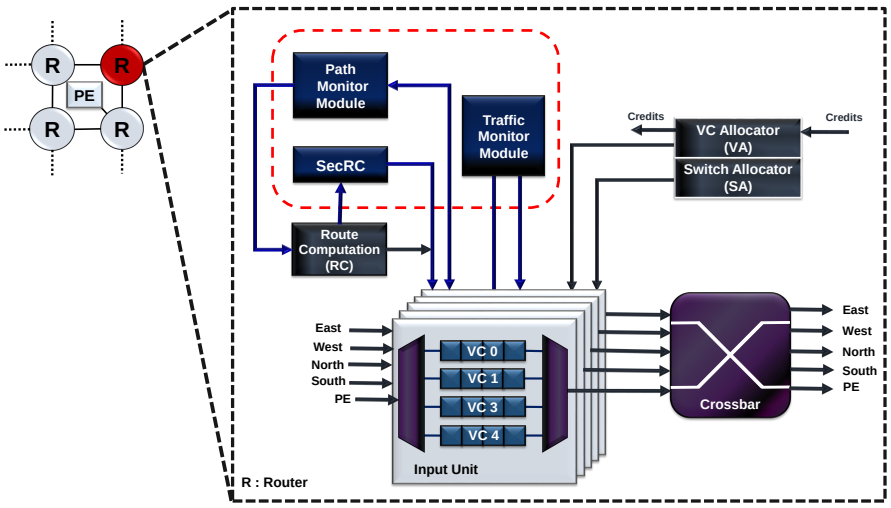


Figure 4.4: NoC router architecture with HT detection framework (traffic and path monitor) and security wrapper (SecRC).

secure wrapper module, SecRC, which will analyse the packets passing through the routing unit of the NoC router (refer to Figure 4.4). The detailed working of the detection/mitigation framework is described in the following sections.

4.3.1 Traffic Monitor Module

The working of the HT detection framework is presented in Algorithm 5. To analyse the impact of HT in NoC, we integrate a traffic monitor module in each router, which monitors the presence of irregular packet traffic in the system. We present the working of the traffic monitor module in Module I of Algorithm 5. The traffic monitor module mainly consists of two modules: Packet Arrival Compute (PAC) module and Notify Flit Generation module. We integrate the PAC module to monitor the packet arrival rate in a router and detect if there is any unusual packet traffic activity. During traffic monitoring, the PAC module is activated for $[t_f - t_s]$ number of cycles with a gap of δ cycles with t_s representing the cycle at which PAC starts and t_f as the cycle at which it stops. Note that in our work, the interval $[t_s : t_f]$ for PAC activation is fixed. Accordingly, multiple PAC activation is done in the duration of T cycles. Here, δ signifies how often the PAC module is activated, whereas the lower the δ, the more times the PAC module is activated and vice-versa. Once monitoring stops, the traffic monitor module computes the mean of the values computed by the PAC module from the collected samples and compares it with the pre-defined system's traffic threshold. Here, the traffic monitor module identifies a communication pattern of router R_i at time T as suspicious only when

$$mean(PAC_{R_i})_T > (Traffic\_Threshold_{Ri})_T \tag{4.5}$$

where $(Traffic\_Threshold_{Ri})_T = (Legitimate\_PAC_{R_i})_T + \Delta$.
Here, we include Δ to represent the jitter associated with the variation of delay due to various factors like network congestion. When a router's PAC mean is greater than the traffic threshold, the traffic monitor module informs the neighbours about this unusual activity using a control flit called notify flit (refer Part B of Algorithm 5).

To analyse the effectiveness of the traffic monitor module, we examine the heat map generated for Uniform Random synthetic traffic benchmark simulation in an 8×8 NoC as shown in Figure 4.5a. From the heat map, we observe that Uniform Random exhibits a high PAC in the centre region of NoC. Furthermore, to understand the proposed packet looping HT's effect on packet processing pattern in NoC routers, with the same simulation environment mentioned above, we trigger the HT in NoC router 42 and 43 and analyse the PAC in NoC. We then compare this PAC with the packet processing pattern of NoC without HT. Figure 4.5b shows the heat map that presents the difference in PAC for the above experiment. We observe that the HTs' activation increases the PAC of HTs compared to NoC without HT. Further analysis also shows that PAC is higher in neighbouring routers of HT. As an HT detection framework that relies only on packet arrival rate can lead to false positives, we integrate a path monitor module in the detection framework that uses the PAC module as a trigger logic for HT detection. The overall time complexity of the HT detection framework is $O(n+m)$, where n is the number of intervals in the Traffic Monitoring Module and m is the number of routers checked in the Path Monitoring Module. The space complexity is $O(m)$, representing the number of routers involved in cycle detection.

4.3.2 Path Monitor Module

The path monitor module is designed to detect packet loops induced by HT, utilizing a monitor flit and a cycle detection unit described in Module II of Algorithm 5. An NoC router activates its path monitor module upon receiving a notify flit. Since the timing of HT triggering is unpredictable, continuously monitoring NoC packets for irregular traffic can increase detection module overhead. To mitigate this, we employ a special control flit called a monitor flit, which records the flit's path during traversal to its destination. Unlike data packets consisting of multiple flits (as described in Chapter 2, Table 2.3), notify and monitor flits are single-flit control packets carrying addresses and flags. The structure of a monitor flit, illustrated in Figure 4.6, includes a 28-bit message field comprising a 4-bit flit ID and a 24-bit route information field. The route information field is divided into multiple 6-bit rid fields storing router IDs in the flit's traversal path. Monitor flits are generated

ALGORITHM 5: HT detection framework

Input : Incoming traffic

Output : HT-induced cycle detection

$num\_PAC\_samples$: No. of PAC samples generated

$PAC_{endTime}$: Cycle at which packet arrival computation ends.

/* Module I: Traffic Monitoring Module (TM)*/

if *TM is active* **then**

  /* Part A: Packet Arrival Compute (PAC) */

  for $interval\ I[t_s : t_f]$ **do**

  if *Curcycle* = *Curcycle* + $\delta < I$ **then**

    PAC = PAC + Packet_processed at cycle interval I Increment num_PAC_sample

  /* Part B: Notify Flit Generation*/

  if *Curcycle* == $PAC_{endTime}$ **then**

    Compute $mean$(PAC) **if** $mean(PAC) >$ *Traffic_Threshold* **then**

      Generate $notify\_flit$

/* Module II: Path Monitoring Module (PM)*/

if *PM is active* **then**

  /* Part C: Monitor Flit Generation */

  if $notify\_flit_{received}$ $<= 2$ **then**

    Generate $monitor$ flits

    Activate cycle detection unit

  /* Part D: Cycle Detection Unit (CD) */

  if *CD is active* **then**

    Analyse incoming $monitor$ flit by checking the rid field in the route info of monitor flit

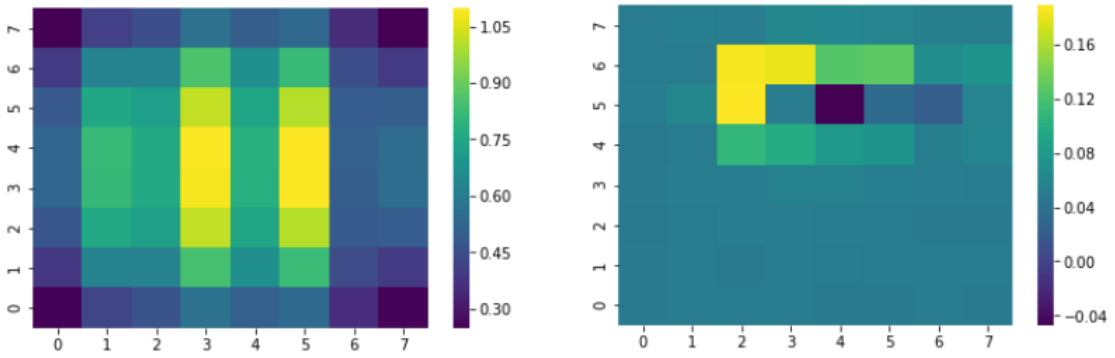
    $(m\_route_{f_m})$

    if $curRouterId \notin$ *m_route*$_{f_m}$ **then**

      $m\_route_{f_m}$ += $currRouterId$

    else if $curRouterId \in m\_route_{f_m}$ **then**

      Report cycle detection



(a) PAC when no HT is activated

(b) Difference in PAC with and without HT

Figure 4.5: Analysis of Packet Arrival Compute (PAC) on 8×8 NoC while running uniform synthetic traffic benchmark. The row and column of heat-map represent the x-coordinate and y-coordinate of the router. The normalized form packet processing rate is given in the legend of each heat map.

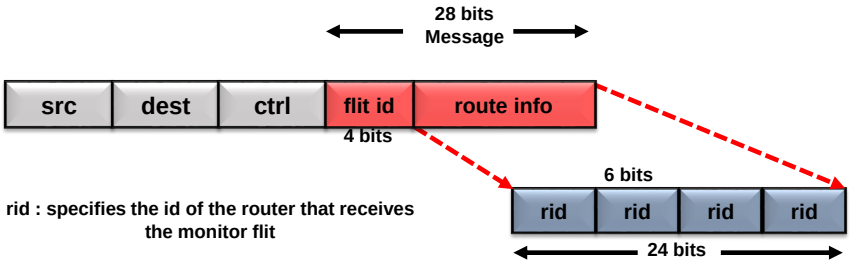


Figure 4.6: Structure of a monitor flit

by a NoC router's path monitor module upon receiving two or more notify flits. Once activated, the path monitor module dispatches monitor flits to one-hop and two-hop neighbors. By utilizing a cycle detection unit, the path monitor module monitors the traversal of monitor flits and detects cycle formations in the network.

The operation of the cycle detection unit, detailed in Part D of Algorithm 5, employs monitor flits to identify HT-induced cycles. Upon receiving a monitor flit, a router adds its router ID (rid) to the monitor flit's route information field and forwards it to the next router. However, each router checks whether its cycle detection module is activated before adding its router ID to the monitor flit. If activated, it verifies whether it has processed the same flit earlier by examining the stored route in the monitor flit. Let $m\_route_{f_m}$ denote the flit route stored in a monitor flit f_m. If the current router's ID is not in the stored path ($m\_route_{f_m}$), it adds its rid to the monitor flit and forwards it to neighbors. For instance, during HT activation, if a monitor flit forms a cycle such as T $\rightarrow$ R $\rightarrow$ Q $\rightarrow$ U $\rightarrow$ T (as depicted in Figure 4.2a), the rid fields in $m\_route_{f_m}$ contain T, R, Q, and U when leaving router U. When router T receives this, it reports HT-induced cycle detection as its router

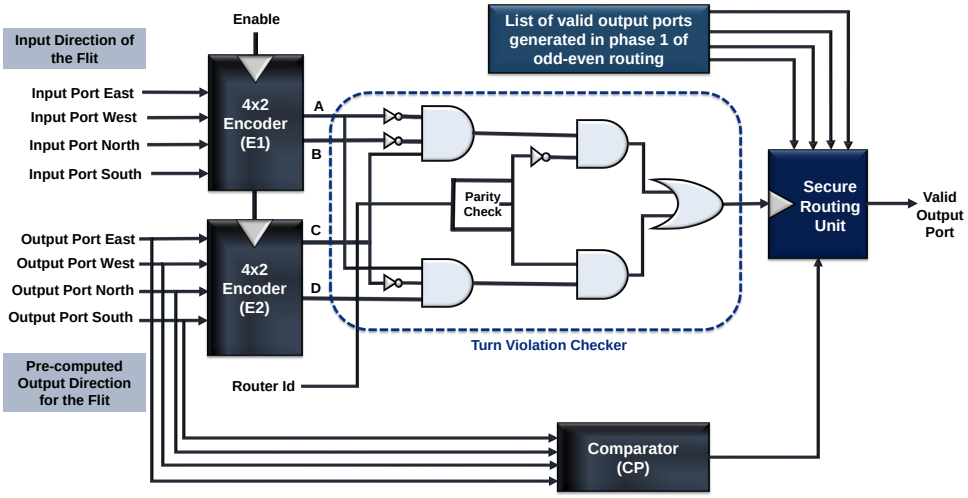


Figure 4.7: Structure of SecRC

ID is identified in the monitor flit's route information field. In general, a router R_i detects a cycle if its router ID is present in an incoming monitor flit f_m. At this stage, it's crucial to ensure that the NoC can intelligently overcome or bypass the impact of such potential cycle sources without disrupting the flow of legitimate packets. For this purpose, we design a security wrapper module for the routing unit, which is explained in the following section.

4.3.3 Security Wrapper for Routing Unit

Once a router detects a cycle using the monitor flit path, a security wrapper module (SecRC) is activated in all NoC routers. Figure 4.7 illustrates the structure of SecRC, which incorporates a turn violation checker and a secure routing unit. The workings of these components are detailed in Algorithm 6. Phase 1 of Algorithm 6 describes the operation of the turn violation checker. Upon activation, SecRC provides the packet's input and output ports to the turn violation checker. Each direction is assigned a two-bit ID (00: East, 01: West, 10: North, and 11: South). Figure 4.7 shows the integration of the turn violation checker within SecRC. The checker employs two 4×2 encoders (E1 and E2), where E1 processes the input direction of the packet and E2 processes the output direction determined by the adaptive routing unit. Based on these input and output directions, the turn violation checker generates a unique 4-bit $turn\_id$ for each packet. A parity bit is assigned, set to 1 for all odd-column routers and 0 otherwise. The $turn\_id$ and parity bit are used to determine whether a packet is making a valid or invalid turn. The truth table for the turn violation checker is presented in Table 4.2, indicating whether the output port assigned by the routing unit results in a valid or invalid turn.

ALGORITHM 6: Working of SecRC module

Input : Input (in_dir) and computed output port (out_dir) for a flit.

Output : Output port selected for a flit

Terminology

out_dir_{list}: set of possible output direction computed in the phase 1 of odd-even algorithm.

out_dir_{sc}: Output port direction computed by the SecRC

$out\_tvc(turn\_id)$: Output of the turn violation checker for the given $turn\_id$.

$genTurnid(in\_dir, out\_dir_{rc})$: returns turn id generated for the input-output direction combination.

$turn\_id$: represents the turn id for a turn.

```
if security_flag is SET then
    /* Phase I: Turn violation checking */
    out_dir_temp = out_dir
    turn_id= genTurnID(in_dir, out_dir_temp)
    if router id is even then
        Set parity bit as 1
    else
        Set parity bit as 0
    if out_tvc(turn_id) is 1 then
        Set violation(turn_id) as TRUE
    /* Phase II: Secure routing */
    if violation(turn_id) is TRUE then
        while turn_id ∈ turn_restrict do
            out_dir_sc = random(out_dir_list)
            turn_id = genTurnID(in_dir, out_dir_sc)
        out_dir = out_dir_sc
    else
        out_dir = out_dir_temp
```

When an invalid turn is detected, the secure routing unit is activated. Phase II of Algorithm 6 outlines the operation of the secure routing unit. It triggers new route computation by randomly selecting one of the output directions computed during Phase I of the odd-even routing as the valid output direction. To avoid turn violations, the secure routing unit generates a $turn\_id$ for the newly computed output direction and cross-references it with potentially violated turn IDs from the odd-even routing algorithm. For example, odd-column routers store $turn_id$s 1001 (North - West) and 1101 (South - West), while even-column routers store $turn_id$s 0010 (East - North) and 0011 (East - South). If head flits already have valid turns, the secure unit assigns the pre-computed output direction as the output for the flit. For instance, if a packet arriving at an even-column router from the West is routed East, and the computed output direction is South, the SecRC module assigns a $turn\_id$ of 0011 and a parity bit of 0. The turn violation checker then verifies whether

Table 4.2: Truth table for turn violation checker

Input - Output Direction	ABCD/Turn id	Even/Odd Parity (P)	X= A ' .B '.C.P ' + A.C '.D.P	Outcome
East - West	0001	0	0	Valid
East - North	0010	0	1	Invalid
East - South	0011	0	1	Invalid
West - East	0100	0	0	Valid
West - North	0110	0	0	Valid
West - South	0111	0	0	Valid
North - East	1000	0	0	Valid
North - West	1001	1	1	Invalid
North - South	1011	0	0	Valid
South - East	1100	0	0	Valid
South - West	1101	1	1	Invalid
South - North	1110	0	0	Valid

the East-South turn is valid, returning 1 for an invalid turn and 0 for a valid turn. Since East-South is a violation in an even-column router, the checker returns 1, activating the secure routing unit to assign one of the pre-computed output directions from Phase 1 of the OE routing. The SecRC module algorithm has a time complexity of O(m), where m represents the number of possible output directions. This complexity arises from the potential number of iterations needed to find a valid turn direction in the secure routing phase. The space complexity is O(m) due to the storage requirements for the $turn\_restrict$ list. The algorithm efficiently handles secure routing with these complexities, ensuring scalability and effectiveness in detecting and mitigating turn violations.

4.4 Results and Discussions

In our experimental evaluation, we compare the proposed security architecture with two existing works: RLAN [62] and SECTAR [92], which address HT-induced packet delays, detection, and mitigation. Both techniques share similarities with our HT model, potentially leading to DoS attacks. To provide a comprehensive comparison, we implement RLAN and SECTAR and conduct a detailed study to evaluate their effectiveness relative to our proposed detection and mitigation approach.
For this evaluation, we consider the following architectures:

- **Baseline:** A TCMP system with no HT.
- **HT-NoC:** A TCMP system with the proposed packet looping HT.
- **SECTAR:** A TCMP system using Trojan-aware routing [92] to detect and mitigate HT-induced misrouting.

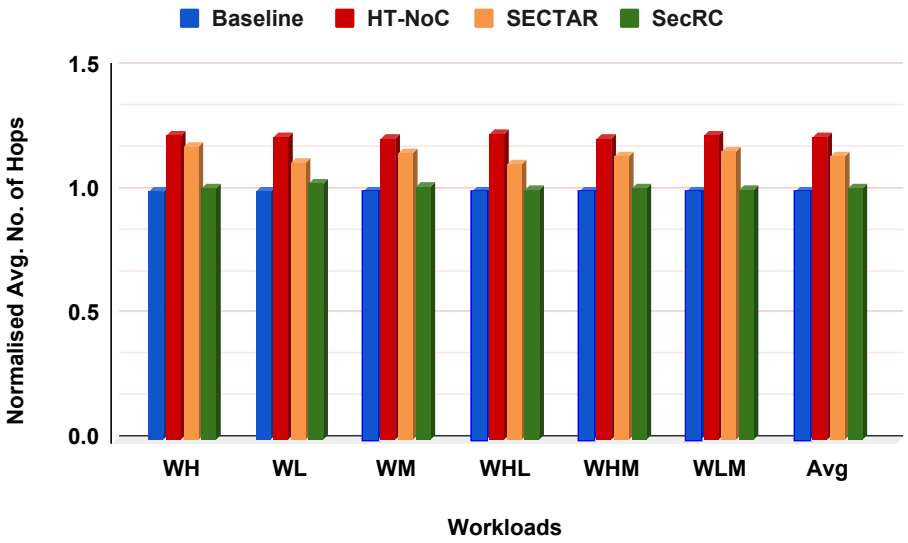


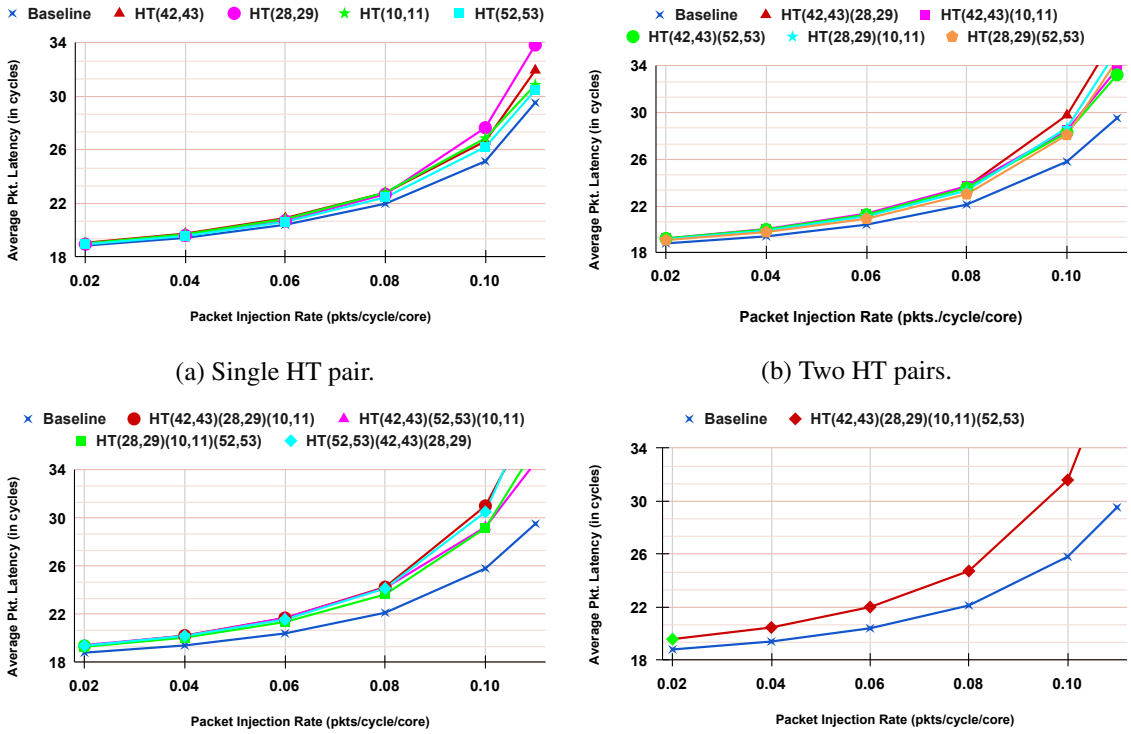
Figure 4.8: Comparison of the average number of hops in real workloads consisting of SPEC CPU 2006 workloads. (Normalised to Baseline)

- **RLAN:** A TCMP system that detects HT-induced bandwidth denial attacks [62].
- **SecRC:** A TCMP system with the proposed architecture for HT detection and mitigation.

HT-NoC represents a system with HTs intentionally inserted into the NoC router hardware. To model HT-NoC, we modified the route computation unit of the NoC router, incorporating additional logic that alters the output port direction, thereby assessing the impact of HT on NoC functionality and security. SECTAR [92] addresses a misrouting HT by modifying the routing module (using the XY routing algorithm) of an NoC router to maliciously assign incorrect output ports, leading to DoS and network injection suppression. SECTAR's mitigation technique, TAR, re-routes packets from HT's neighbors to avoid HT, and we compare TAR's performance with our SecRC approach. RLAN [62] proposes an HT model that deploys a bandwidth attack, manipulating on-chip resource availability. RLAN's detection module uses a Proximity Analog Packet (PAP) to measure latency at nodes near HT nodes, comparing it to latency from original packets. We evaluate RLAN's detection technique in the presence of our proposed HT and compare it to the performance of our SecRC approach.

4.4.1 Impact on Average Network Hops

Our analysis shows that when HT is activated, HT-induced cycles increase the average network hops for selected packets. Figure 4.8 illustrates this, with a 22% increase in average hops for packets in HT-NoC. SECTAR, designed for HT mitigation, introduces a 16% increase in hop count compared



(a) Single HT pair.

(b) Two HT pairs.

(c) Three HT pairs.

(d) Four HT pairs.

Figure 4.9: Average packet latency analysis with uniform random synthetic traffic in 8×8 mesh NoC with HT: (a) represents the single HT pair activation. (b) represents the impact while two HT pairs are activated. (c) represents three HT pairs activation and its impact and (d) represents four HT pairs activation and its impact on latency. Here, lower the line on the graph, the better.

to the Baseline, as it bypasses potentially compromised routers to enhance security, though at the cost of additional routing steps. In contrast, SecRC takes a different approach, allowing packets to pass through potentially compromised routers while implementing secure routing techniques. This results in a 12.5% reduction in hop count compared to SECTAR and keeps hop counts close to Baseline levels, with only a 1.4% increase. SecRC's approach effectively balances security with network efficiency, minimizing the impact on latency and path length.

4.4.2 Impact on Average Packet Latency

Figure 4.9 illustrates the impact on Average Packet Latency (APL) under synthetic traffic with various injection rates and multiple HT pairs activated in different network regions. With a single HT pair (28, 29), APL increases by 15% compared to the Baseline (Figure 4.9a). The packet processing pattern shows a concentration in specific areas, indicating potential exploitation by an adversary familiar with the traffic. As injection rates rise, packet latency consistently increases across different

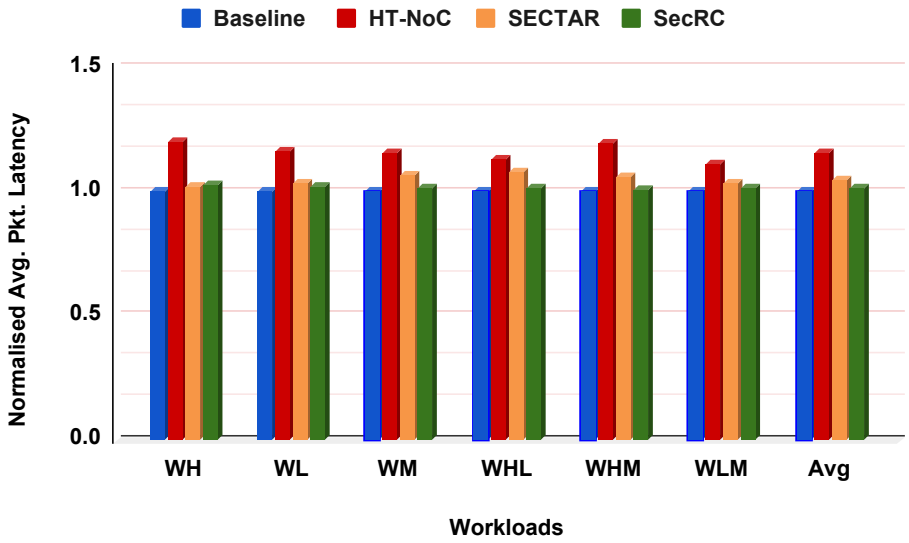


Figure 4.10: Comparison of average packet latency in real workloads consisting of SPEC CPU 2006 workloads. (Normalised to Baseline)

HT pairs due to more packets entering HT-induced cycles. At higher injection rates, HT pair (42, 43) has a similar impact on latency as (28, 29). Activating both pairs simultaneously results in a 25% increase in APL over the Baseline (Figure 4.9b), while three HT pairs can increase APL by over 35%, and four pairs by 39% (Figure 4.9c, Figure 4.9d).

When running SPEC 2006 benchmarks with three activated HT pairs, HT-NoC shows a 15.9% increase in APL over the Baseline, with high MPKI workloads like WH seeing more than a 20% increase. SECTAR mitigates HT effects with a 5% APL increase over the Baseline due to its HT bypass routing. In contrast, SecRC limits the increase to 1.7%. For workload WH, SecRC reduces APL by 15% compared to HT-NoC, outperforming SECTAR's 14% reduction. SecRC also achieves reductions of 12.4% and 12.1% in APL for WL and WM workloads, respectively. Overall, SecRC reduces APL by 9.9% for WHL, 15.7% for WHM, and 8.4% for WLM compared to HT-NoC, with a slight 1.7% increase over the Baseline due to non-minimal routing paths.

4.4.3 Impact on Maximum Packet Latency

We also study the Maximum Packet Latency (MPL) which is defined as the highest latency incurred by packets passing through HT-infected routers. We define MPL in the NoC architecture without any HT as the highest latency incurred by packets passing through routers that are chosen to deploy the HT. The analysis of maximum packet latency performed on various architectures is illustrated in Figure 4.11. Our analysis shows that HT-NoC has significantly high MPL across all workloads

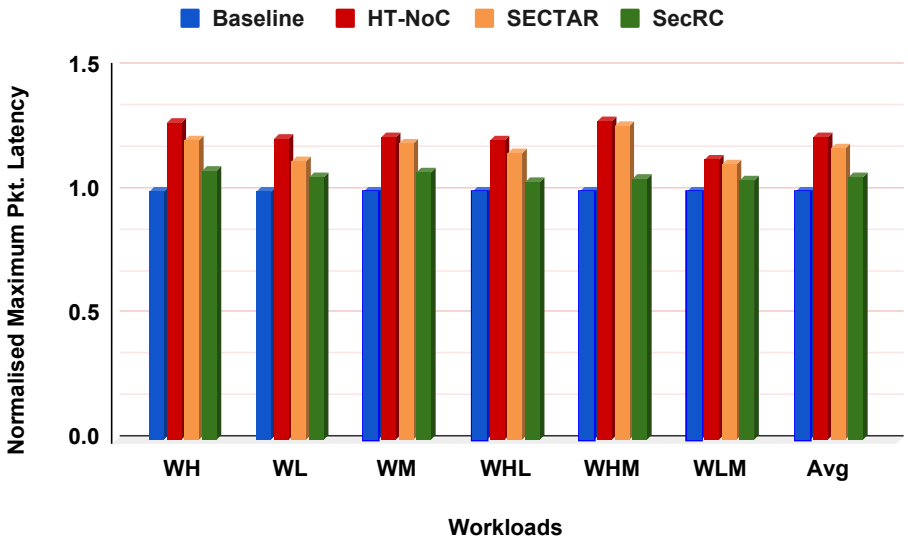


Figure 4.11: Comparison of maximum packet latency in real workloads consisting of SPEC CPU 2006 workloads. (Normalised to Baseline)

with an average of 22.4%. Similarly, in the case of SECTAR, due to the added cycles for packet traversal introduced by bypass routing, we observe an average MPL increase of 17.7%. In contrast, our proposed technique, SecRC, shows a slightly higher MPL (only 6.1%) than the Baseline across all workloads. SecRC shows an average reduction of 9.8% in MPL compared to SECTAR. This enhancement is attributed to the secure routing unit integrated into SecRC, which uses pre-computed output ports for HT mitigation, thereby eliminating the need for HT bypassing.

4.4.4 Impact on Processor Performance

To analyze the throughput of applications running in a TCMP, we examined the number of instructions executed per cycle (IPC). Figure 4.12 shows the normalized IPC for various architectural scenarios under different workloads. The HT-NoC architecture demonstrates an average IPC reduction of 19% compared to the Baseline, due to delays caused by packets traversing HT-induced cycles (as discussed in Section 4.4.2). These delays slow down instruction execution, leading to a decrease in IPC and overall application performance. Our analysis highlights the effectiveness of the proposed SecRC technique in mitigating HT impact. SecRC achieves IPC levels close to the Baseline, with only a 1% average reduction. The SECTAR technique, which reroutes packets to bypass HTs, results in a 6.1% average reduction in IPC compared to the Baseline. We also examined the impact of HT on the throughput of different applications running concurrently, as higher IPC generally contributes to increased throughput. Specifically, we analyzed the core-wise IPC of WHM,

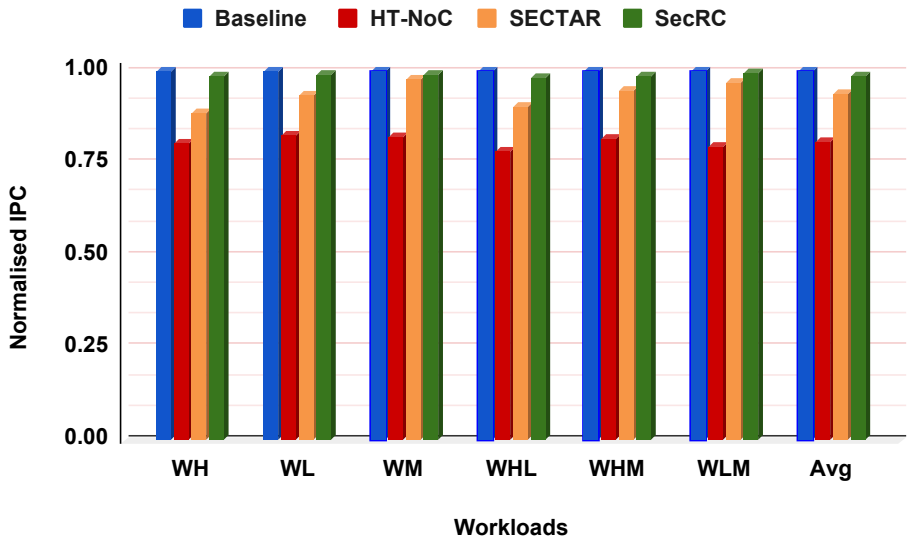


Figure 4.12: Comparison of IPC in real workloads consisting of SPEC CPU 2006 workloads.(Normalised to Baseline)

which includes High MPKI and Medium MPKI benchmarks. High MPKI benchmarks typically represent mission-critical or computationally intensive tasks, while Medium MPKI benchmarks are less critical but still important for system efficiency.

When HT is activated, High MPKI applications on cores 0 to 31 (Figure 4.13) experience an average IPC degradation of 18%, while Medium MPKI applications on cores 32 to 63 (Figure 4.14) show a 17% IPC drop, impacting throughput by slowing task execution. However, with the proposed SecRC technique, throughput is effectively maintained. High MPKI applications on cores 0 to 31 see only a 1.8% average IPC reduction compared to the Baseline, and Medium MPKI benchmarks on cores 32 to 63 experience a 2.6% drop. These results highlight SecRC's effectiveness in preserving performance in a TCMP environment with varying application criticality.

4.5 Area and Power Overhead Analysis

For the detection and mitigation of HT, additional circuitry is integrated into each NoC router. This circuitry includes a detection module with a 1-bit security flag to activate the Secure Routing Control (SecRC) unit, a 1-bit flag for the secure routing unit, a 5-bit counter for traffic monitoring, and a 15-bit register for storing the Packet Acceptance Criterion (PAC) threshold. The total storage overhead for this added circuitry amounts to 176 bytes (22 bits × 64 cores). This overhead is essential to support the enhanced security functionalities embedded within the router, ensuring that it can effectively detect and respond to HT-induced threats. To evaluate the impact of this additional

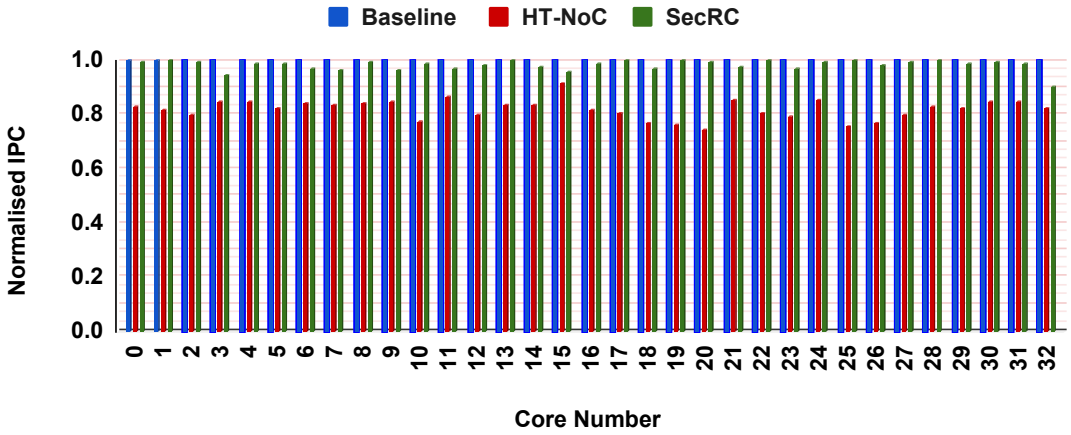


Figure 4.13: Core-wise comparison of IPC SPEC CPU 2006 workload WHM - core 0 to core 32

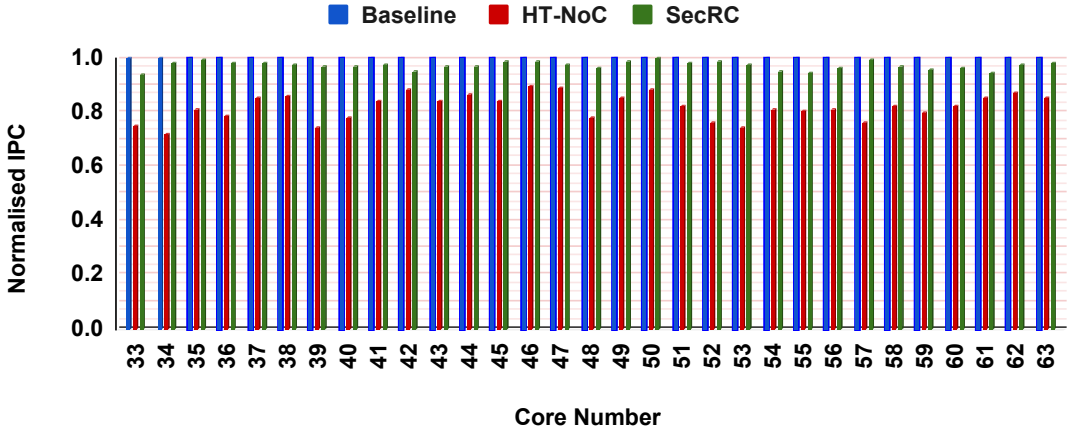


Figure 4.14: Core-wise comparison of IPC SPEC CPU 2006 workload WHM - core 33 to core 63

circuitry, both in terms of area and power consumption, we utilized parameters from McPAT [87] and conducted a comparative analysis against other security techniques, specifically RLAN and SECTAR. The results of this analysis are summarized in Table 4.3.

The SecRC architecture incurs a 6.1% increase in area compared to the Baseline design, which is justified by its enhanced security features, allowing for effective detection and mitigation of hardware trojans (HTs) with minimal impact on chip size. When compared to SECTAR, SecRC's area increase is 3.4%, while it shows a 6.2% reduction in area compared to RLAN, underscoring its compact design. In terms of power, SecRC demonstrates a 4.9% increase in peak dynamic power over the Baseline, which is significantly lower than the 10.7% increase for RLAN and only slightly higher than SECTAR's 2.9%. For runtime dynamic power, SecRC's 4.7% increase is higher than SECTAR's 2.3% but much lower than RLAN's 11.6%. This slight rise is due to the continuous operation of additional monitoring components but remains efficient for practical use.

Table 4.3: Area and power overhead analysis

Metric	Baseline	RLAN	SECTAR	SecRC
Area (mm2)	1.14	1.29	1.17	1.21
Peak Dynamic (W)	1.02	1.13	1.05	1.07
Runtime Dynamic (W)	0.043	0.048	0.044	0.045

4.6 Chapter Summary

This chapter demonstrated how an HT modifying the routing unit of an NoC router can launch a DoS attack. Experimental results reveal that the proposed system effectively detects and mitigates HT effects with minimal area overhead and power consumption, while maintaining application QoS with controlled performance degradation.

CHAPTER

Fortifying NoC Security Against Trojan-Induced Packet Duplication Attacks

This chapter introduces an HT which is embedded within the NI of NoC and is capable of initiating packet duplication attacks. To counteract the adverse effects of HT-induced packet duplication, we present a comprehensive mitigation framework which is deployed within the NI and actively monitors the incoming and outgoing messages within the NoC. Through experimental evaluation, we demonstrate that the mitigation framework effectively suppresses the HT's impact, restoring system performance with minimal hardware overhead.

5.1 Introduction

Even though NoC is one of the front-end RTL IPs, the design team usually configures it at the end based on the requirements of the TCMP architecture. Any interference in NoC during the inter-core communication of critical applications can create damage in the system [9][20][21][92][94][95]. For instance, a malicious IP can exploit the NoC to extract data without the need to hack individual IPs [25]. In this context, HTs can be inserted into the NoC design by adversaries with access to the RTL or netlist of the NoC IP. These HTs are often triggered intermittently and selectively duplicate NoC packets, making them challenging to detect. Several known attacks have proposed data theft through NoC packet duplication [9][68][80]. To enhance security, manufacturers often implement encryption and decryption mechanisms for NoC packet communication. Since the NoC packet's header is crucial for routing and arbitration decisions at intermediate routers, it cannot be encrypted. We exploit this vulnerability to enable packet duplication by placing an HT in the NI, which can affect system performance [95]. Designing a countermeasure to mitigate HTs capable of executing

packet duplication and adversely affecting the system poses significant challenges. The placement of the mitigation framework becomes a critical concern, as its effectiveness depends on its location relative to the HT. For instance, the countermeasure may prove ineffective if the HT is embedded in the NI while the mitigation unit is kept in the router. The existing countermeasures discussed in the literature are HT-specific, addressing specific attack scenarios. However, these mitigation techniques often fall short when confronted with novel versions of HT duplication attacks in TCMPs. Therefore, the optimal solution involves designing a versatile mitigation framework capable of neutralizing current and potential future attacks of a similar nature. This work also attempts to design a ubiquitous mitigation framework for such packet duplication attacks.

The chapter initially focuses on the proposed HT model named LOKI[1], which can sit in NI and simultaneously create detrimental effects in different NoC components without hacking into them [95]. Then, to counter LOKI-like HTs, we propose a ubiquitous mitigation framework named HULK[2]. This chapter makes the following significant contributions:

1. We present how the proposed HT, LOKI, can selectively duplicate NoC control packets, which can degrade the system performance.

2. We show that LOKI can impact the packet latency, miss penalty and system speedup to degrade the overall TCMP performance. We also compare LOKI against a similar state-of-the-art HT and discuss how LOKI can bypass existing mitigation techniques and the need for a better solution.

3. We propose the framework, HULK, to mitigate LOKI-like HTs that attack TCMPs through packet duplication. HULK is integrated into the NI and monitors all the messages going in and out of the NoC. Hence, HULK can handle any anomalies happening in the NI, the routers or the links.

4. We evaluate HULK to show that it can neutralise LOKI's impact, thereby improving the system's performance. We also compare HULK against a similar state-of-the-art mitigation technique to show its effectiveness. With negligible hardware overhead, we propose that HULK could be employed as a ubiquitous framework to mitigate packet duplication attacks.

[1] God of Mischief in Norse Mythology and a Supervillain in Marvel Comics [96].
[2] A Superhero in Marvel Comics who defeats LOKI [96].

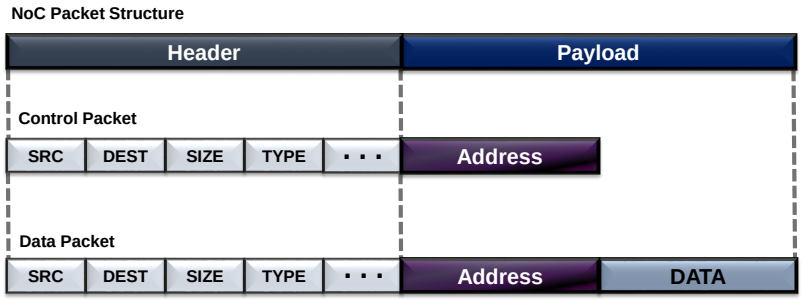


Figure 5.1: Structure of a NoC packet exchanged between IPs

The rest of the chapter is organized as follows. Section 5.2 outlines the proposed packet duplication HT threat model and the potential consequences of attacks on TCMPs. Section 5.3 describes the existing mitigation techniques to handle packet duplication attacks, while the proposed mitigation framework is discussed in Section 5.4. The experimental analysis, focusing on performance, is presented in Section 5.5. Subsequently, Section 5.6 presents the power and area overhead analysis of the system under evaluation. Finally, the chapter concludes in Section 5.7.

5.2 Threat Model: Trojan Design and its Impact

This section introduces the design of the proposed HT, LOKI, which includes a circuit diagram depicting a feasible method of integrating it into the NI. Subsequently, example scenarios are examined to showcase how LOKI operates and its impact on system performance.

5.2.1 Packet Duplication Trojan Design

IPs in TCMPs communicate with each other by exchanging NoC packets with each other, which consist of a header and a payload[3], as depicted in Figure 5.1. Here, the header packet contains essential information such as source (SRC), destination (DEST), message size (SIZE), message type (TYPE), etc., for packet traversal, while the payload carries data that is to be sent to the destination. Usually, these NoC packets are either control or data types, with control packets used to request data or send coherence messages and having a payload that carries the requested memory address [25]. When an IP source (IP_{SRC}) wants to communicate with its destination (IP_{DEST}), it sends the message to the NI (green IP node in Figure 5.2). NI then converts the message (purple) into a control or data packet (yellow) and sends it to the router to traverse the NoC and reach the destination. A similar process occurs in reverse to receive messages at the destination (blue IP node). To prevent

[3]Packet payload and HT payload are entirely different

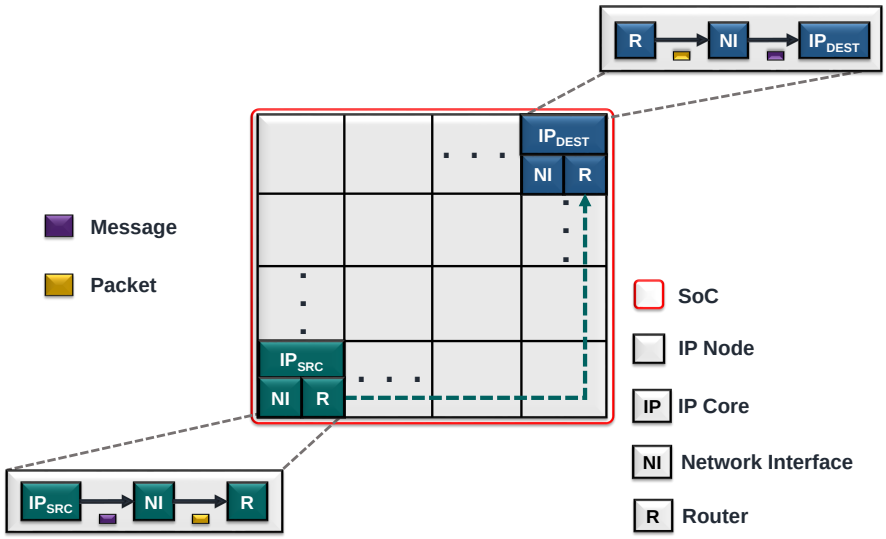


Figure 5.2: Communication between IPs in a TCMP

data stealing in modern TCMPs, various encryption techniques have been proposed that encrypt only the payload of the packet, as the header information is required for routing and arbitration decisions in the NoC. On the other hand, control packets, which do not carry any data, are not usually encrypted. LOKI takes advantage of this vulnerability by duplicating control packets in the NI to launch an attack on TCMPs. LOKI not only duplicates packets but also impacts the system performance with minimal resource usage. Raparti et al. [9] introduced an HT model similar to LOKI in their work, focusing on duplicating packets in the NI's flit queue to execute a data snooping attack. The significance of LOKI lies in being the first attempt to utilize control packets for deploying a packet duplication attack in TCMPs which can degrade the system performance.

Figure 5.3 represents a potential method for incorporating the proposed HT LOKI into an IP node's NI. Whenever an IP_{SRC} sends a message to the IP_{DEST}, the NI, as shown in Figure 5.2, converts a message to a packet and vice versa. Typically, the NI employs two modules: the Packetiser for the source IP node and the De-Packetiser for the destination IP node. In this case, LOKI is integrated into the Packetiser, so we have not included details about the De-Packetiser in Figure 5.3. When the NI receives a message from the source IP core, the Packet Generator sub-module converts it into a packet. Next, the Flit Generator sub-module converts the packet into one or more flits and stores them in the circular Flit Queue. Finally, the stored flits are inserted into the NoC router and sent to their respective destination IP nodes.LOKI's attack can be initiated by a backdoor Kill Switch (KS) [57]. The Packet Type (PT) indicates the message type (TYPE from Figure 5.1), and the Flit Type (FT) indicates the type of flit. To reduce storage overhead, we target read requests as they are control packets with only one head flit. LOKI uses a buffer of 1-flit size and can store only

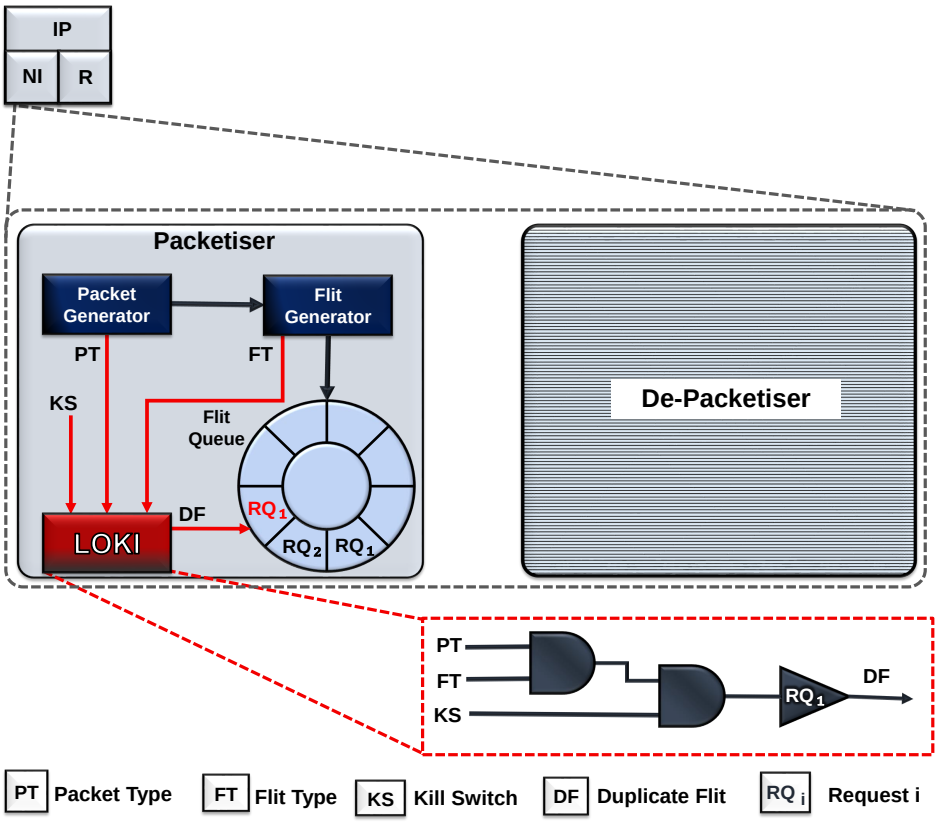


Figure 5.3: Insertion of the proposed packet duplication HT (LOKI) in NI

one duplicate read request packet at a time. As shown in Figure 5.3, LOKI is triggered only when PT=READ, FT=HEAD, and KS=EN[4]. LOKI then copies a head flit (read request packet) from the Flit Generator sub-module into its buffer and keeps inserting this duplicate flit into the Flit Queue until KS=DS. The duplicate flit is inserted only in the free locations to avoid interrupting the usual communication flow.

5.2.2 Impact of Packet Duplication Trojan Attack

Consider an SoC with a 2-level on-chip caching, as depicted in Figure 5.4. Each IP core has a private L1 Instruction cache (L1I) and Data cache (L1D), while the LLC is shared and distributed across the cores. An IP core searches for the instruction or data in the L1 cache, and upon a miss, the corresponding L1 cache Controller (L1 CTLR) sends a request message to the next-level LLC bank Controller (LLC CTLR). The requested cache block is then supplied to the IP core. Here, we assume that LOKI is mounted on the NI of the green IP node in Figure 5.4 and is triggered by the KS. We present two scenarios to illustrate how LOKI can launch an attack on NoC at once.

[4]READ = Read request, EN = Enable, and DS = Disable

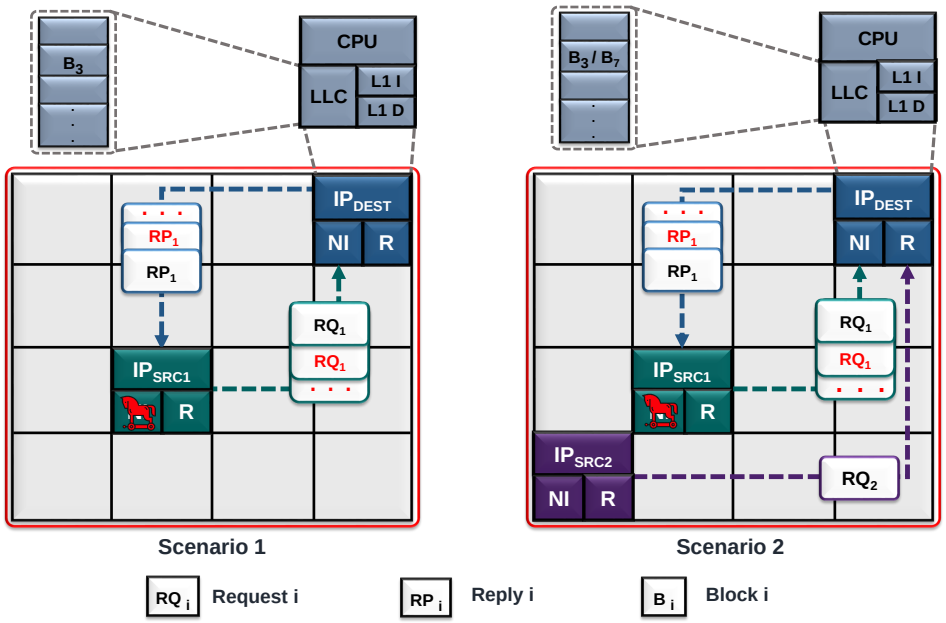


Figure 5.4: Demonstration of the LOKI attack.

5.2.2.1 Impact on NoC: Attack Scenario 1

IP_{SRC1} (green) requests a cache block B_3, leading to an L1 cache miss. The L1 CTLR sends a read request message RQ_1 to the NI for the destination LLC bank at IP_{DEST} (blue), where B_3 is cached. The LLC CTLR replies with the requested cache block in a reply message RP_1. When LOKI is active, it copies RQ_1 from the Flit Generator into its buffer and keeps inserting this duplicate request in free locations of the Flit Queue. These duplicate requests are treated like genuine requests by the LLC CTLR, leading to multiple replies with RP_1. Since IP_{SRC1} and IP_{DEST} are sufficiently spaced, the duplicate requests and replies (red) spread across the entire SoC, increasing contention in the routers and links and resulting in increased packet latency.

5.2.2.2 Impact on Cache: Attack Scenario 2

While Scenario 1 is in progress, IP_{SRC2} (purple) sends a read request message RQ_2 to IP_{DEST} for the cache block B_7. The destination LLC bank maps B_3 and B_7 to the same set due to the employed mapping strategy (modulo 4). Due to frequent requests for B_3 by duplicate RQ_1, B_7 could be evicted from the LLC bank, leading to a thrashing phase where useful blocks are evicted to service duplicate requests. Future requests for these evicted blocks encounter LLC miss, which increases the miss penalty (affecting the caches). As a result of the increased packet latency and miss penalty, instruction execution by sources like IP_{SRC2} will be delayed, leading to delayed commit and the delayed issue of new instructions. This decreases the system speedup, affecting the

core. Even though LOKI is mounted on a single NI, it can affect cache and processor performance without directly intruding on them.

5.3 Mitigating Packet Duplication Trojan Using Existing Techniques

We have seen how LOKI activation triggers read request duplication attacks and leads to performance impact at the cache and core levels. An attempt to isolate and prevent duplicate packets from entering the on-chip network is proposed in SeRA [61]. An NoC router has input buffers called VCs to store incoming packets while participating in routing and arbitration decisions. SeRA detects the VCs storing duplicate messages and masks them at runtime to deny entry into the network. However, VC isolation reduces their utilisation and hampers network performance. The situation worsens with LOKI, as the duplicate packets from the NI can be stored in any of the free VCs, rendering SeRA's mitigation approach ineffective. PLDU [74] is a proposal that assigns a unique identity to every packet and can detect duplicates. Here, the source and destination of a packet are tagged with a dynamic random value, and the tag is scrambled with the data payload. If an adversary alters either of the source, destination, data and tag, it gets detected at the destination IP core. But LOKI targets control packets without data payload and does not alter anything in the packet header. Hence, the duplicate packets by LOKI could become false negatives with PLDU. Similar proposals for duplicate packet detection, when integrated on routers, are ineffective as LOKI deploys attacks from the NI itself. SIM [9] is a duplicate packet detection approach that primarily attempts to block packets with invalid headers from entering the network. To validate packets, SIM relies on a unique key, which is a function of multiple variables, including the destination field of a packet. When an invalid packet is detected, SIM sends a signal to the Flit Queue at NI to discard duplicates. While SIM is ideal for detecting information leakage attacks through modified packet destinations, LOKI is not one of them. As LOKI does not alter anything in the packet header, the SIM approach may also generate false negatives.

5.4 HULK: Mitigation Framework for Packet Duplication HT Attacks

In Section 5.3, it is highlighted that current detection methods for packet duplication attacks primarily focus on safeguarding and verifying packet header information and how LOKI bypasses these existing detection techniques. To overcome these challenges, we introduce a security framework called HULK, which uses Packet Status Holding Registers (PSHR) to counter packet duplication attacks. Similar to how non-blocking cache controllers utilize Miss Status Holding Registers

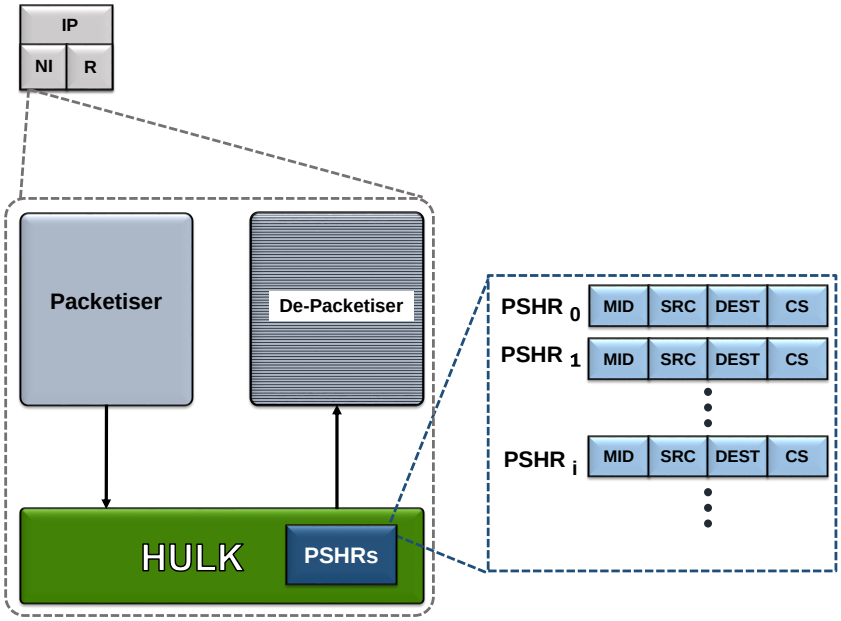


Figure 5.5: Packet Status Holding Registers (PSHRs) used in HULK

(MSHRs) to manage outstanding misses and prevent redundant requests for the same cache block, HULK employs PSHRs at the NIs to monitor in-flight request packets. This section describes the design of HULK and its working using the previous example from Figure 5.4.

HULK mainly uses message ID (MID) to filter out duplicate messages before processing them in NI. The simplest solution to avoid duplication is assigning a unique identity to each message (MID). However, trivial MIDs like monotonically increasing ones are easily predicted by adversaries looking to insert duplicates. Hence, HULK calculates MID as given below:

$$MID = SRC \oplus DEST \oplus NI_{SRC} \oplus KEY \tag{5.1}$$

Here, SRC and DEST represent the source and destination core ID and NI_{SRC} as the source NI. The KEY is an 8-bit key generated using a pseudo-random number generator. To incorporate MID into the packet, we modified the header field of the NoC packet given in Figure 5.1. Whenever the core generates a message, an MID is calculated at the Miss Status Holding Register (MSHR) of the L1 CTLR and added to the packet header as shown in Figure 5.1. To preserve the authenticity of the messages, HULK also generates a checksum (CS) before transmission and appends it to the message header. For CS calculation, we apply Alder-32 [26] on the MID. Alder-32 is one of the simplest cryptographic hash functions and trades reliability for speed with the Cyclic Redundancy Check (CRC) of the same length.

ALGORITHM 7: Working of PSHR

Input: Incoming request message

Output: Add or discard message

MID: ID of the incoming message

CS: Checksum of the incoming message

$PSHR_i$: An entry in the PSHR

$Alder32$: The cryptographic hash function

if $\exists PSHR_i \mid PSHR_i[MID] == MID$ **then**
    /* Duplicate message detected */
    Discard the message
else
    $CheckSum \leftarrow Alder32(MID)$
    if $Checksum == CS$ **then**
        Add an entry in the PSHR
    else
        /* Corrupted message detected */
        Discard the message

As detailed in Section 5.2.1, the Packetiser and De-Packetiser modules of the NI handle the injection and ejection of flits from routers, respectively. In our proposed framework, these modules are integrated with PSHRs, as illustrated in Figure 5.5. Once HULK is activated, all the in-flight messages to NI first pass through PSHR before it is forwarded to the network, as shown in Fig.5.5. Upon receiving a request message from a connected core, the NI creates a unique entry in the PSHRs. Subsequently, upon receiving a response packet, the corresponding PSHR entry is removed before the De-Packetiser begins processing. The size of the PSHR aligns with that of the MSHR, ensuring that the NI effectively manages the same set of request messages. As MID uniquely identifies the incoming messages, there is no possibility of in-flight messages having the same MID. Thus, if PSHR encounters a message with an already existing MID, it discards the message and continues processing other messages.

To understand how HULK mitigates packet duplication attacks, consider the same scenario from Figure 5.4. Once HULK is activated, before the read request RQ_1 is inserted into the router from the Flit Queue, a unique entry is added to the PSHR. As shown in Figure 5.6, HULK compares the MID of RQ_1 with all the PSHR entries. The possibility of having a match is none as every MID is unique. When LOKI is active, RQ_1 is duplicated and attempts are made to insert them into the router. With the proposed HULK in place, these duplicates will be detected and discarded. So, even

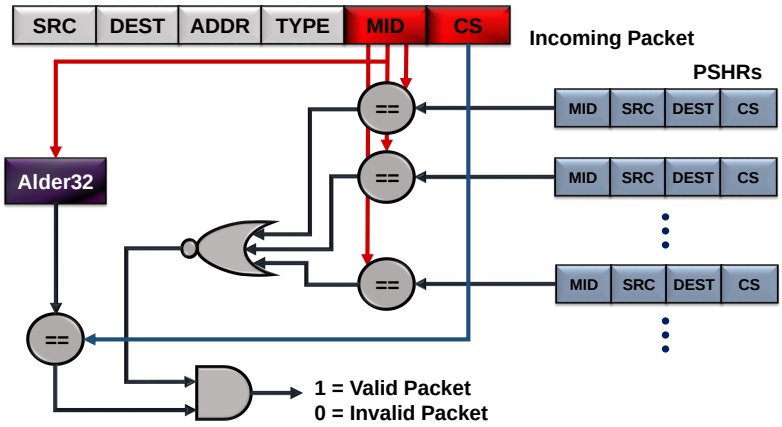


Figure 5.6: Working of the proposed HULK inside the NI

if LOKI successfully duplicates RQ_1, they will never enter the network. The working of HULK to add or discard an incoming message in the PSHR is presented in Algorithm 7. Similar steps are followed to delete or discard a response message. If LOKI-like adversaries try to outsmart HULK by using a new MID for RQ_1, the checksum test in line 10 of Algorithm 7 will fail. In fact, if an adversary tries to modify any of the MID, SRC, DEST and CS, HULK can detect and mitigate them. There could be a scenario where a duplicate RQ_1 is inserted only after the genuine one is serviced and its PSHR entry is deleted. In such a scenario, HULK cannot detect the duplicate RQ_1. LOKI can insert just one such duplicate at any given time. These duplicate insertions will be sufficiently spaced in time and, hence, will not impact the overall system performance. HULK can be viewed as a ubiquitous mitigation framework offering the combined benefits of SeRA, PLDU and SIM-like proposals, as it can detect duplicate and corrupted request messages.

The PSHR algorithm operates with a constant time complexity of O(1) for checking duplicates, calculating the checksum, and adding entries, assuming efficient implementations of data structures and constant-size inputs. The space complexity is O(n), where n represents the number of entries in the PSHR. This efficient performance ensures rapid processing and minimal memory overhead for managing incoming messages.

5.5 Results and Discussions

For performance evaluation, we assess the effectiveness of our proposed mitigation technique, HULK, by comparing it with SIM, a recent packet duplication mitigation approach [9]. Since the referenced work introduces an HT circuit similar to the one proposed in our study, assessing its performance in addressing LOKI-like HTs provides an insight into its effectiveness relative to our

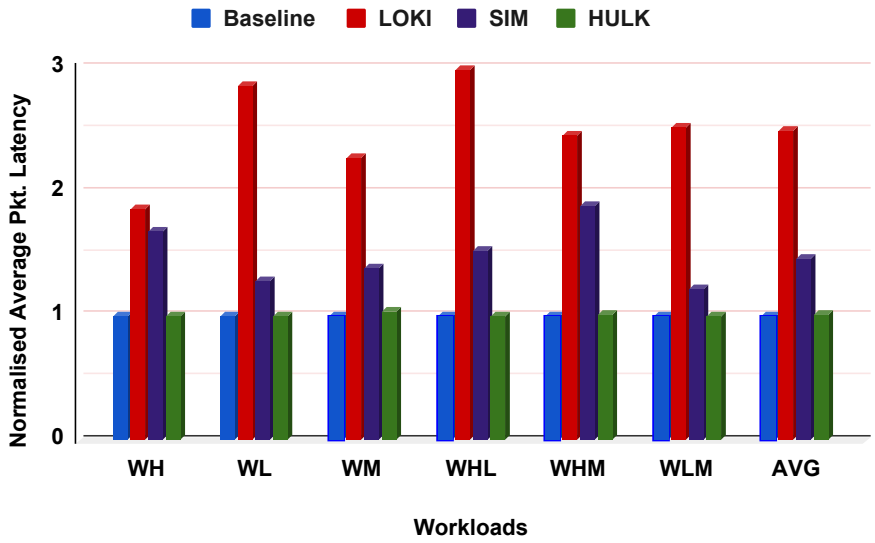


Figure 5.7: Comparison of average packet latency in real workloads consisting of SPEC CPU workloads (Normalised to Baseline)

mitigation technique, HULK. We consider the following architectures for evaluation:

- **Baseline:** A TCMP system without any HT.
- **LOKI:** A TCMP system with LOKI HT on NI.
- **SIM:** A TCMP system with the existing mitigation technique, SIM [9] on every NI to mitigate packet duplication.
- **HULK:** A TCMP system with the proposed HULK on every NI to mitigate packet duplication.

5.5.1 Impact on Average Packet Latency

The time a packet takes to travel from its source to its destination is known as packet latency. Figure 5.7 depicts the average packet latency (normalised with respect to the Baseline) for various architectures under study. The duplication of request and reply packets by LOKI leads to NoC congestion, resulting in an average packet latency increase of 2.5x (red bars). While analysing HT's effect on workloads WL and WHL, we can see that, with HT in action, APL increases by an average of 2.9x times over the Baseline. Here, the duplicate packets in the network play a non-negligible part in network congestion, thus increasing the network delay. A similar trend can be seen in queuing delay, where genuine packets experience more queuing delay due to the injection of duplicate packets in the NI buffer. The effectiveness of the existing SIM mitigation framework on LOKI is very random (purple bars). We observe that SIM exhibits worse performance while

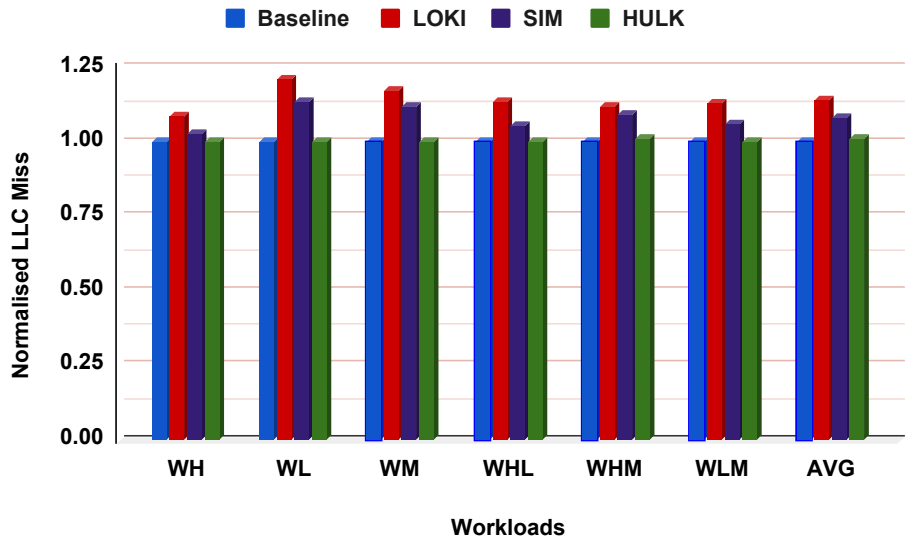


Figure 5.8: Comparison of LLC misses in real workloads consisting of SPEC CPU 2006 workloads (Normalised to Baseline)

running workload mixes consisting of High and Medium MPKI benchmarks due to the increase in queuing time associated with packet validation in the NI. In general, SIM shows an average packet latency increase of 1.4x (purple bars) compared to the Baseline. However, the proposed HULK mitigation framework shows steady performance, with an average packet latency decrease of 31% compared to SIM. Moreover, HULK offers average packet latency close to the Baseline across all the evaluated workloads (green bars).

5.5.2 Impact on LLC Miss

Figure 5.8 shows the normalized LLC misses in the presence of LOKI, SIM and HULK. We observe that the duplication of requests results in intermittent probing of the same data block in the LLC bank, leading to the eviction of a few blocks. As a result, LOKI shows an increase in LLC misses by an average of 14.3% across all the workloads (red bars) for LOKI. Workloads WL, WHL and WLM, consisting of Medium and Low MPKI applications, experience an average increase of LLC misses by 15.9%. This is attributed to these workloads' low packet injection rate, allowing LOKI to inject more duplicate requests into the Flit Buffer. The effectiveness of the existing SIM mitigation framework on LOKI is counter-productive (refer to purple bars). It can be noted that the SIM framework mainly detects the HT that modifies the destination of an NoC packet. Although such security-enhanced NI is ideal for detecting information leakage attacks, it may encounter false negatives while detecting LOKI in some instances, as the HT does not change any header information for packet duplication.

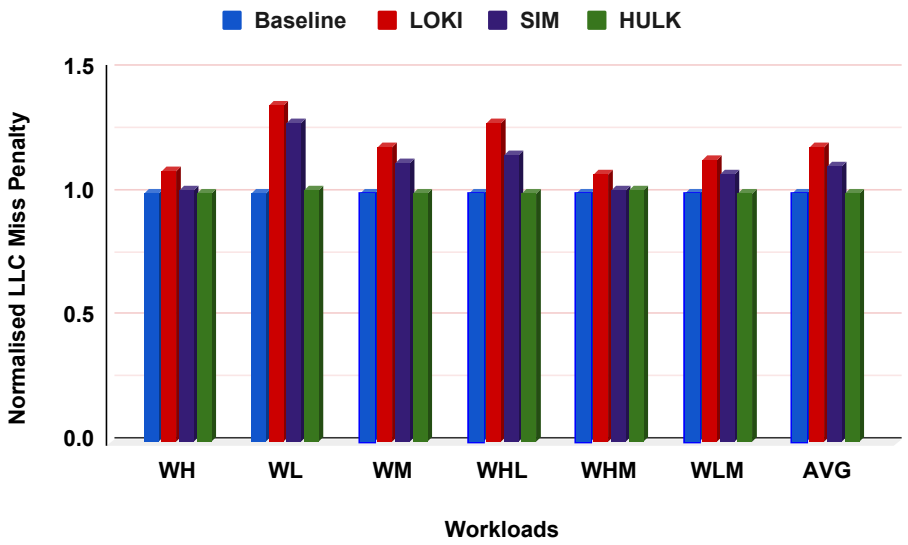


Figure 5.9: Comparison of L1 cache miss penalty in real workloads consisting of SPEC CPU 2006 workloads (Normalised to Baseline)

This makes it challenging for SIM to identify LOKI attacks in specific scenarios. Consequently, SIM exhibits an increase in LLC misses. In contrast, the proposed HULK mitigation framework demonstrates consistent response, with an average LLC misses decrease of 6.3% compared to SIM and very close to Baseline. Additionally, HULK can block duplicate requests, obtaining a similar Baseline trend of LLC misses in all the evaluated workloads (indicated by green bars).

5.5.3 Impact on L1 Cache Miss Penalty

The time taken to service an L1 cache miss by bringing in the requested data block is known as the L1 cache miss penalty. With the increase in LLC misses, the L1 cache miss penalty is also bound to increase, as the missed data blocks are fetched from the off-chip main memory. According to Figure 5.9, the presence of LOKI (represented by the red bars) leads to a maximum increase of up to 35% and an average increase of 19% on the L1 cache miss penalty. This indicates that LOKI affects the performance of L1 cache accesses. Even though SIM can reduce the LOKI-induced miss penalty by 9%, it still exhibits an average of 11% increase in miss penalty compared to the Baseline. This highlights that SIM provides some mitigation against the LOKI's effect on L1 cache performance but cannot completely eliminate the increase in L1 cache miss penalty. HULK demonstrates an L1 cache miss penalty behaviour that closely resembles the Baseline.

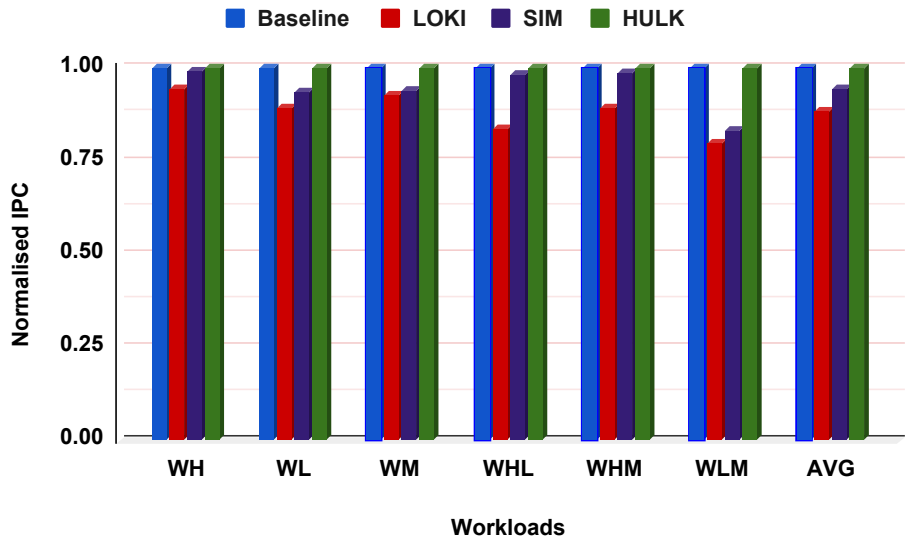


Figure 5.10: Comparison of system speedup in real workloads consisting of SPEC CPU 2006 workloads (Normalised to Baseline)

Table 5.1: Area and Power overhead analysis

Metric	Basleine	LOKI	SIM	HULK
Area (mm2)	1.14	1.16	1.23	1.19
Peak Dynamic (W)	1.02	1.05	1.09	1.06
Runtime Dynamic (W)	0.043	0.044	0.046	0.045

5.5.4 Impact on Processor Performance

Packet latency and L1 cache miss penalty have a direct impact on instruction execution time. To compare the system speedup between the Baseline and other architectures, we use Instructions Per Cycle (IPC). Figure 5.10 illustrates that LOKI is able to indirectly attack multiple components like NoC, cache, etc and decrease the system speedup. The IPC graph indicates a maximum reduction of up to 20% and an average reduction of 11.8% when comparing LOKI to the Baseline. This observation highlights how LOKI can bring down the system's overall performance. On the other hand, HULK achieves a similar IPC to the Baseline, indicating that it maintains a comparable level of performance. However, SIM exhibits a 5.5% reduction in IPC compared to the Baseline, suggesting a slight decrease in performance.

5.6 Area and Power Overhead Analysis

We evaluate the power overhead of NoC architectures using McPAT [87]. Table 5.1 shows the power overhead when LOKI is active, and various security schemes are applied. LOKI exhibits

minimal overhead, with a 1.75% increase in area, 2.94% in peak dynamic power, and 2.33% in runtime dynamic power, making it highly efficient at remaining undetected. To mitigate LOKI, SIM introduces the highest overhead, with an 8.33% area increase, 7.84% rise in peak dynamic power, and 7% in runtime dynamic power, making it more detectable. HULK offers a balanced solution with moderate increases: 4.39% in area, 3.92% in peak dynamic power, and 4.65% in runtime dynamic power.

5.7 Chapter Summary

This chapter discussed the threat model surrounding packet duplication HT attacks, which can have potential impacts on TCMPs. Here, we presented the design and implementation of HULK as a robust defence mechanism against packet duplication HT attacks. We also presented the experimental results obtained from evaluating HULK's performance, thus highlighting its strengths and comparing it with other state-of-the-art techniques. Furthermore, this chapter analyzed the overhead associated with implementing HULK, providing insights into the practical implications of deploying our defence strategy.

CHAPTER

6

Conclusion and Future Directions

In this thesis, we explored potential HT threats in ICs, specifically focusing on integrating HTs in various locations within TCMPs, as detailed in Chapter 2. Primarily, we investigated major threats in NoC, including packet misrouting, packet looping, and packet duplication attacks that can deploy DoS attacks. Despite existing techniques that address the issues to some extent, we observed that these methods either incur significant hardware overhead or lack a comprehensive mitigation framework. Through this thesis, we presented the possibilities of Trojan models deployed in different locations in NoC routers. Additionally, we proposed detection and mitigation techniques with negligible hardware overhead. The thesis summary is outlined in the following section, while Section 6.2 offers insights into potential future works in a similar direction.

6.1 Summary of Thesis

Figure 6.1 summarises the contributions made within the thesis. The primary emphasis of the thesis is the modelling of HTs in NoC that deploys variants of DoS attacks. Through Thesis Contribution 1, we proposed an efficient detection and mitigation technique to address the impact of packet misrouting HT attacks in NoC. With the proposed technique, TAR, we demonstrated that the effects of HT can be mitigated using Trojan-aware routing algorithm. In Thesis Contribution 2, we discussed the mitigation of packet looping HT attacks on NoC by adapting the security wrapper module and implementing packet monitoring techniques. Also, this thesis explored the various effects of packet duplication HT attacks in NoC through Thesis Contribution 3. While existing works offer mitigation strategies for HT-induced packet duplication, they often fall within specific HT attack mitigation categories. Our thesis proposed a novel technique utilizing Packet Status Holding registers in the NI, which effectively mitigates HT-induced packet duplication across NoC.

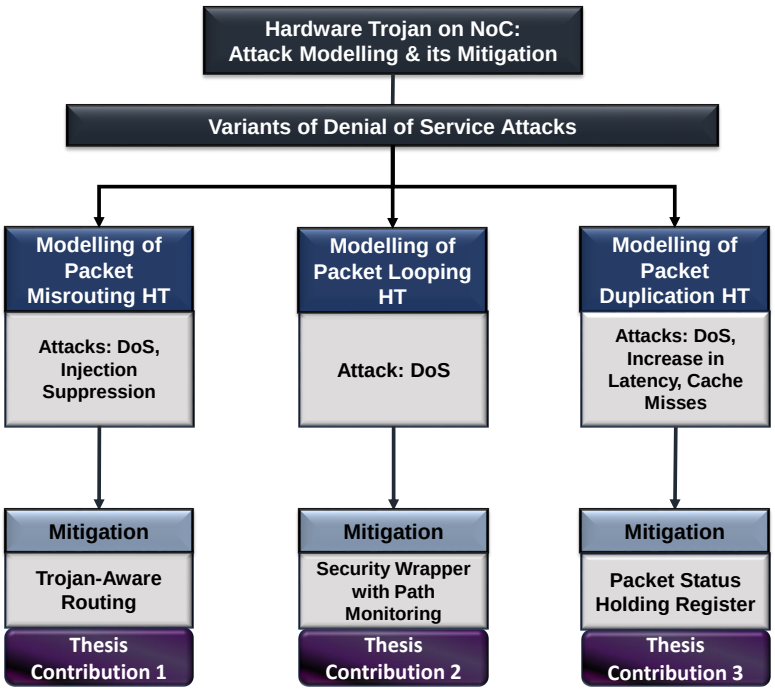


Figure 6.1: Summary of thesis

The experimental framework utilized to model each contribution is elaborated in Section 2.8.4. The major performance metrics of each contribution and the additional hardware requirements are shown in Figure 6.2. While analyzing the average packet latency, we observed that the misrouting HT incurs a significant 1.9x increase, packet looping HT shows a 1.5x increase and packet duplication HT demonstrates a substantial 2.5x increase compared to the Baseline. Furthermore, investigation into the mitigation techniques reveals that while HULK achieves performance similar to the Baseline, SecRC incurs a slight 1.7% increase, and TAR introduces a 7% increase in average packet latency compared to the Baseline. In terms of IPC, misrouting HT shows an average of 77% decrease, packet looping HT shows a 19% decrease, and packet duplication HT shows an average of 11.8% decrease compared to the Baseline. However, analysis of the proposed mitigation techniques shows that TAR exhibits a 3% decrease, and SecRC shows an average of 1% decrease over the Baseline. In contrast, HULK achieves remarkable performance similar to Baseline.

Regarding the area overhead, TAR incurs only a minimal 2.6% increase compared to the Baseline, while SecRC shows an 6.1% increase. However, HULK shows a 4.4% increase in area relative to the Baseline, thus indicating varying levels of resource utilization across the proposed systems. Further analysis of dynamics power overhead shows that TAR exhibits a moderate 2.6% increase compared to the Baseline, while SecRC results in a higher 4.8% increase. However, HULK

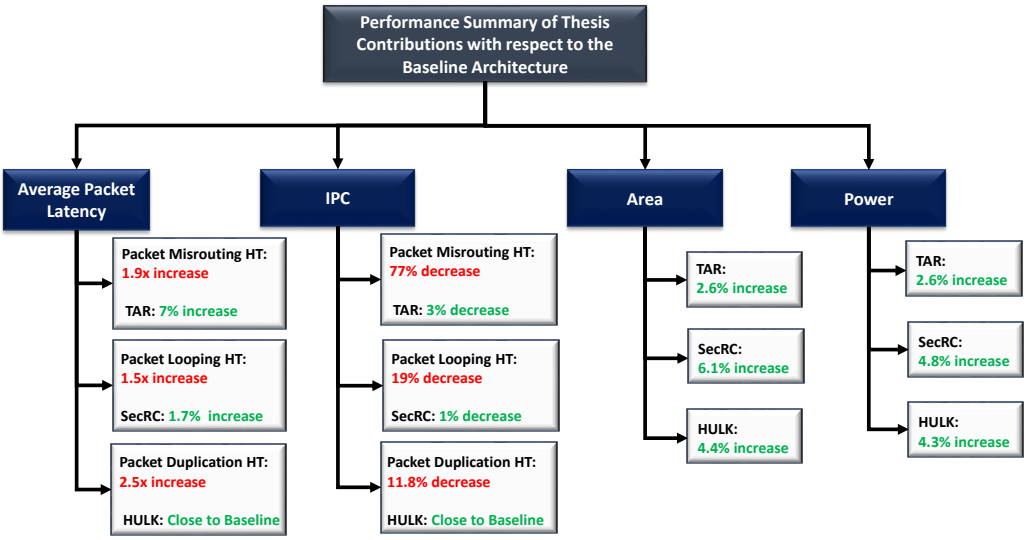


Figure 6.2: Summary of performance analysis

shows a comparatively lower 4.3% increase, suggesting different power consumption characteristics across the proposed solutions. It is observed that TAR, SecRC, and HULK utilize slightly more LUTs compared to the Baseline, indicating a moderate increase in resource consumption. Moreover, it is noted that all three thesis contributions ensure that the proposed solutions meet the timing requirements, ensuring that the enhanced NoC architectures can still maintain system performance.

6.2 Future Work

The contributions of the thesis can be extended in many ways. Some of the possible works that can be explored in future directions are summarized below.

- In Chapter 4, the work focuses on the activation of a misrouting HT within a specific NoC router at a particular time. However, there is potential for additional research regarding the positioning of misrouting HT instances that cooperate to trigger their attack across the NoC. Such an approach has the capacity to magnify the system's susceptibility to damage. As TAR deals with HT mitigation through the implementation of HT-induced router bypassing, it introduces a trade-off concerning packet latency. This aspect prompts further exploration to devise a solution that mitigates HT without necessitating the isolation of the HT.

- In Chapter 5, SecRC deals with the mitigation of HTs that modifies the selection strategy of odd-even routing. Further research can be done to study the impact of packet-looping HT attacks in other adaptive routing algorithms.
- In Chapter 6, HULK deals with the detection and mitigation of packet duplication attacks that degrade the system performance as a whole. Here, further research can be done to explore the integration of artificial intelligence and machine learning algorithms to enhance the HULK framework's ability to autonomously detect and respond to emerging threats in real-time.

Bibliography

[1] W. H. Wolf, A. A. Jerraya, and G. Martin, "Multiprocessor System-on-Chip (MPSoC) Technology," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, pp. 1701–1713, 2008.

[2] (2020) Arteris IP FlexNoC. [Online]. Available: https://www.arteris.com/products/non-coherent-noc-ip/flexnoc

[3] W. J. Dally and B. Towles, "Route packets, not wires: on-chip interconnection networks," in *Proceedings of the Design Automation Conference*, 2001, pp. 684–689.

[4] N. Potlapally, "Hardware security in practice: Challenges and opportunities," in *IEEE International Symposium on Hardware-Oriented Security and Trust*, 2011, pp. 93–98.

[5] F. Farahmandi, Y. Huang, P. Mishra, F. Saqib, and J. Plusquellic, *System-on-Chip Security: Validation and Verification*, 2019.

[6] A. P. D. Nath, S. Ray, A. Basak, and S. Bhunia, "System-on-chip security architecture and CAD framework for hardware patch," *Asia and South Pacific Design Automation Conference*, pp. 733–738, 2018.

[7] R. JayashankaraShridevi, D. M. Ancajas, K. Chakraborty, and S. Roy, "Security Measures Against a Rogue Network-on-Chip," *Journal of Hardware and Systems Security*, vol. 1, pp. 173–187, 2017.

[8] J. Coburn, S. Ravi, A. Raghunathan, and S. Chakradhar, "SECA: security-enhanced communication architecture," in *Proceedings of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, 2005, pp. 78–89.

[9] V. Y. Raparti and S. Pasricha, "Lightweight Mitigation of Hardware Trojan Attacks in NoC-based Manycore Computing," in *Proceedings of the Annual Design Automation Conference*, 2019, pp. 1–6.

[10] S. Hansman and R. Hunt, "A taxonomy of network and computer attacks," *Computers Security*, vol. 24, pp. 31–43, 2005.

[11] D. A. Osvik, A. Shamir, and E. Tromer, "Cache Attacks and Countermeasures: The Case of AES," in *Proceedings of the Topics in Cryptology: The Cryptographers' Track at the RSA Conference.*, 2006, pp. 1–20.

[12] Y. Yarom and K. Falkner, "FLUSH+ RELOAD: A high resolution, low noise, L3 cache side-channel attack," in *Proceedings of the USENIX Conference on Security Symposium*, 2014, pp. 719–732.

[13] (2019) Glitching: The hardware attack that can disrupt secure software. [Online]. Available: https://www.darkreading.com/edge/theedge/glitching-the-hardware-attack-that-can-disrupt-secure-software-/b/d-id/1336119

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

[15] M. M. Ahmed, A. Dhavlle, N. Mansoor, S. M. P. Dinakarrao, K. Basu, and A. Ganguly, "What Can a Remote Access Hardware Trojan do to a Network-on-Chip?" in *IEEE International Symposium on Circuits and Systems*, 2021, pp. 1–5.

[16] E. Choi and S. Chang, "A consumer tracking estimator for vehicles in GPS-free environments," *IEEE Transactions on Consumer Electronics*, vol. 63, no. 4, pp. 450–458, 2017.

[17] U. Guin, K. Huang, D. DiMase, J. M. Carulli, M. Tehranipoor, and Y. Makris, "Counterfeit Integrated Circuits: A Rising Threat in the Global Semiconductor Supply Chain," *Proceedings of the IEEE*, vol. 102, pp. 1207–1228, 2014.

[18] (2021) Mobileye SuperVision™ for Hands-free ADAS. [Online]. Available: https://www.mobileye.com/super-vision/

[19] R. Torrance and D. James, "The State-of-the-Art in IC Reverse Engineering," in *International Workshop on Cryptographic Hardware and Embedded Systems*, 2009, pp. 363–381.

[20] S. Charles, V. Bindschaedler, and P. Mishra, "Digital Watermarking for Detecting Malicious Intellectual Property Cores in NoC Architectures," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 30, pp. 952–965, 2022.

[21] P. Mishra and S. Charles, *Network-on-Chip Security and Privacy*, 2021.

[22] G. Ascia, V. Catania, M. Palesi, and D. Patti, "Implementation and Analysis of a New Selection Strategy for Adaptive Routing in Networks-on-Chip," *IEEE Transactions on Computers*, vol. 57, pp. 809–820, 2008.

[23] P. T. Huang and W. Hwang, "An adaptive congestion-aware routing algorithm for mesh network-on-chip platform," in *IEEE International SOC Conference*, 2009, pp. 375–378.

[24] L. Liu, Y. Sun, Z. Zhu, and Y. Yang, "A congestion-aware OE router employing fair arbitration for network-on-chip," *Journal of Semiconductors*, vol. 39, p. 125006, 2018.

[25] S. Charles and P. Mishra, "A Survey of Network-on-Chip Security Attacks and Countermeasures," *ACM Computing Surveys*, vol. 54, pp. 1–36, 05 2021.

[26] P. Deutsch and J.-L. Gailly, "Rfc1950: Zlib compressed data format specification version 3.3," Tech. Rep., 1996.

[27] O. Wallach. (2021) Visualizing the global semiconductor supply chain. [Online]. Available: https://www.visualcapitalist.com/sp/visualizing-the-global-semiconductor-supply-chain/

[28] M. Rostami, F. Koushanfar, and R. Karri, "A Primer on Hardware Security: Models, Methods, and Metrics," *Proceedings of the IEEE*, vol. 102, pp. 1283–1295, 2014.

[29] R. Saleh, S. Wilton, S. Mirabbasi, A. Hu, M. Greenstreet, G. Lemieux, P. Pande, C. Grecu, and A. Ivanov, "System-on-Chip: Reuse and Integration," *Proceedings of the IEEE*, vol. 94, pp. 1050–1069, 2006.

[30] (2023) Arteris ip. [Online]. Available: https://www.arteris.com/

[31] T. Feng, H. Pei, Z. Jin, and X. Wu, "A survey and perspective on electronic design automation tools for ensuring SoC security," in *International SoC Design Conference*, 2022, pp. 215–216.

[32] (2023) Qflow. [Online]. Available: http://opencircuitdesign.com/qflow//

[33] "Ngspice:Open Source Mixed Mode, Mixed Level Circuit Simulator." Based on Berkeley's Spice3f5, 1992.

[34] J. Verley, E. R. Keiter, and H. K. Thornquist, "Xyce: Open Source Simulation for Large-Scale Circuits." Sandia National Lab, Tech. Rep., 2018.

[35] C. Wolf., "Yosys open synthesis suite." Tech. Rep., 2018.

[36] Y. Lin, Z. Jiang, J. Gu, W. Li, S. Dhar, H. Ren, B. Khailany, and D. Z. Pan, "DREAMPlace: Deep Learning Toolkit-Enabled GPU Acceleration for Modern VLSI Placement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, pp. 748–761, 2021.

[37] T. Ajayi and D. Blaauw, "OpenROAD: Toward a Self-Driving, Open-Source Digital Layout Implementation Tool Chain," in *Proceedings of Government Microcircuit Applications and Critical Technology Conference*, 2019.

[38] C. Rooney, A. Seeam, and X. Bellekens, "Creation and Detection of Hardware Trojans Using Non-Invasive Off-The-Shelf Technologies," *Electronics*, vol. 7, p. 124, 2018.

[39] S. Bhunia, M. S. Hsiao, M. Banga, and S. Narasimhan, "Hardware Trojan attacks: Threat analysis and countermeasures," *Proceedings of the IEEE*, vol. 102, pp. 1229–1247, 2014.

[40] M. Areno, "Supply Chain Threats Against Integrated Circuits," *Intel Whitepaper*, 2020.

[41] T. Perez, M. Imran, P. Vaz, and S. Pagliarini, "Side-Channel Trojan Insertion - A Practical Foundry-Side Attack via ECO," in *IEEE International Symposium on Circuits and Systems*, 2021, pp. 1–5.

[42] F. Almeida, M. Imran, J. Raik, and S. Pagliarini, "Ransomware Attack as Hardware Trojan: A Feasibility and Demonstration Study," *IEEE Access*, vol. 10, pp. 44 827–44 839, 2022.

[43] Y. Liu, Y. Jin, A. Nosratinia, and Y. Makris, "Silicon Demonstration of Hardware Trojan Design and Detection in Wireless Cryptographic ICs," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 25, pp. 1506–1519, 2017.

[44] M. Xue, C. Gu, W. Liu, S. Yu, and M. O'Neill, "Ten years of hardware Trojans: a survey from the attacker's perspective," *IET Computers & Digital Techniques*, vol. 14, pp. 231–246, 2020.

[45] T. Zhang, J. Park, M. Tehranipoor, and F. Farahmandi, "PSC-TG: RTL Power Side-Channel Leakage Assessment with Test Pattern Generation," in *ACM/IEEE Design Automation Conference*, 2021, pp. 709–714.

[46] H. Pearce, V. R. Surabhi, P. Krishnamurthy, J. Trujillo, R. Karri, and F. Khorrami, "Detecting hardware trojans in pcbs using side channel loopbacks," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 30, pp. 926–937, 2022.

[47] C. Pilato, K. Basu, M. Shayan, F. Regazzoni, and R. Karri, "High-Level Synthesis of Benevolent Trojans," in *Design, Automation & Test in Europe Conference Exhibition*, 2019, pp. 1124–1129.

[48] K. Basu, S. M. Saeed, C. Pilato, M. Ashraf, M. T. Nabeel, K. Chakrabarty, and R. Karri, "CAD-Base: An Attack Vector into the Electronics Supply Chain," *ACM Transactions on Design Automation of Electronic Systems*, vol. 24, pp. 1–30, 2019.

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

[50] M. B. Taylor, W. Lee, J. E. Miller, D. Wentzlaff, I. Bratt, B. Greenwald, H. Hoffmann, P. R. Johnson, J. S. Kim, J. Psota, A. Saraf, N. Shnidman, V. Strumpen, M. I. Frank, S. Amarasinghe, and A. Agarwal, *Tiled multicore processors*, 2009.

[51] T. Bjerregaard and S. Mahadevan, "A survey of research and practices of Network-on-chip," *ACM Computing Surveys*, vol. 38, pp. 1–es, 2006.

[52] A. Agarwal, C. Iskander, and R. Shankar, "Survey of Network on Chip (NoC) Architectures Contributions," *Journal of Engineering, Computing and Architecture*, vol. 3, pp. 21–27, 2009.

[53] W. Dally, "Virtual-channel flow control," *IEEE Transactions on Parallel and Distributed Systems*, vol. 3, pp. 194–205, 1992.

[54] Y. Lyu and P. Mishra, "Scalable Activation of Rare Triggers in Hardware Trojans by Repeated Maximal Clique Sampling," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, pp. 1287–1300, 2020.

[55] R. JS, K. Chakraborty, and S. Roy, "Hardware Trojan Attacks in SoC and NoC," *The Hardware Trojan War: Attacks, Myths, and Defenses*, pp. 55–74, 2018.

[56] S. Charles, Y. Lyu, and P. Mishra, "Real-time Detection and Localization of DoS Attacks in NoC based SoCs," in *Design, Automation Test in Europe Conference Exhibition*, 2019, pp. 1160–1165.

[57] T. Boraten and A. K. Kodi, "Mitigation of Denial of Service Attack with Hardware Trojans in NoC Architectures," in *International Parallel and Distributed Processing Symposium*, 2016, pp. 1091–1100.

[58] V. J. Kulkarni, R. Manju, R. Gupta, J. Jose, and S. Nandi, "Packet Header Attack by Hardware Trojan in NoC based TCMP and its Impact Analysis," in *IEEE/ACM International Symposium on Networks-on-Chip*, 2021, pp. 21–28.

[59] M. H. Khan, R. Gupta, J. Jose, and S. Nandi, "Dead Flit Attack on NoC by Hardware Trojan and Its Impact Analysis," in *Proceedings of the International Workshop on Network on Chip Architectures*, 2021, pp. 10–15.

[60] L. Daoud and N. Rafla, "Routing Aware and Runtime Detection for Infected Network-on-Chip Routers," in *International Midwest Symposium on Circuits and Systems*, 2018, pp. 775–778.

[61] N. Prasad, R. Karmakar, S. Chattopadhyay, and I. Chakrabarti, "Runtime mitigation of illegal packet request attacks in Networks-on-chip," in *IEEE International Symposium on Circuits and Systems*, 2017, pp. 1–4.

[62] J. Rajesh, D. M. Ancajas, K. Chakraborty, and S. Roy, "Runtime Detection of a Bandwidth Denial Attack from a Rogue Network-on-Chip," in *Proceedings of the International Symposium on Networks-on-Chip*, 2015, pp. 1–8.

[63] M. Hussain, A. Malekpour, H. Guo, and S. Parameswaran, "EETD: An Energy Efficient Design for Runtime Hardware Trojan Detection in Untrusted Network-on-Chip," in *IEEE Computer Society Annual Symposium on VLSI*, 2018, pp. 345–350.

[64] R. Gupta, V. J. Kulkarni, J. Jose, and S. Nandi, "Securing On-Chip Interconnect against Delay Trojan Using Dynamic Adaptive Caging," in *Proceedings of the Great Lakes Symposium on VLSI*, 2022, pp. 411–416.

[65] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, "Flush+ Flush: A Fast and Stealthy Cache Attack," in *Detection of Intrusions and Malware, and Vulnerability Assessment: International Conference*, 2016, pp. 279–299.

[66] G. Saileshwar, C. W. Fletcher, and M. K. Qureshi, "Streamline: A Fast, Flushless Cache Covert-Channel Attack by Enabling Asynchronous Collusion," in *ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021, pp. 1077–1090.

[67] C. Reinbrecht, A. Susin, L. Bossuet, G. Sigl, and J. Sepúlveda, "Side channel attack on NoC-based MPSoCs are practical: NoC Prime+ Probe attack," in *Symposium on Integrated Circuits and Systems Design*, 2016, pp. 1–6.

[68] D. M. Ancajas, K. Chakraborty, and S. Roy, "Fort-NoCs: Mitigating the threat of a compromised NoC," in *ACM/EDAC/IEEE Design Automation Conference*, 2014, pp. 1–6.

[69] C. Sudusinghe, S. Charles, S. Ahangama, and P. Mishra, "Eavesdropping Attack Detection Using Machine Learning in Network-on-Chip Architectures," *IEEE Design Test*, vol. 39, pp. 28–38, 2022.

[70] S. Charles and P. Mishra, "Lightweight and Trust-Aware Routing in NoC-Based SoCs," in *IEEE Computer Society Annual Symposium on VLSI*, 2020, pp. 160–167.

[71] S. Charles and P. Mishra, "Securing Network-on-Chip Using Incremental Cryptography," in *IEEE Computer Society Annual Symposium on VLSI*, 2020, pp. 168–175.

[72] Y. Wang and G. E. Suh, "Efficient Timing Channel Protection for On-Chip Networks," in *IEEE/ACM International Symposium on Networks-on-Chip*, 2012, pp. 142–151.

[73] T. Boraten and A. Kodi, "Mitigation of Hardware Trojan based Denial-of-Service attack for secure NoCs," *Journal of Parallel and Distributed Computing*, vol. 111, pp. 24–38, 2018.

[74] M. Hussain and H. Guo, "Packet Leak Detection on Hardware-Trojan Infected NoCs for MPSoC Systems," in *Proceedings of the International Conference on Cryptography, Security and Privacy*, 2017, pp. 85–90.

[75] A. K. Biswas, S. Nandy, and R. Narayan, "Router Attack toward NoC-enabled MPSoC and Monitoring Countermeasures against such Threat," *Circuits, Systems, and Signal Processing*, vol. 34, pp. 3241–3290, 2015.

[76] X. Cui, E. Koopahi, K. Wu, and R. Karri, "Hardware Trojan Detection Using the Order of Path Delay," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 14, pp. 1–23, 2018.

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

[78] M. H. Khan, R. Gupta, V. J. Kulkarni, J. Jose, and S. Nandi, "Hardware Trojan Mitigation for Securing On-chip Networks from Dead Flit Attacks," in *IEEE International Conference on Very Large Scale Integration*, 2022, pp. 1–6.

[79] Q. Yu and J. Frey, "Exploiting error control approaches for Hardware Trojans on Network-on-Chip links," in *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*, 2013, pp. 266–271.

[80] S. Charles, M. Logan, and P. Mishra, "Lightweight Anonymous Routing in NoC based SoCs," in *Design, Automation Test in Europe Conference Exhibition*, 2020, pp. 334–337.

[81] M. Werner, T. Unterluggauer, L. Giner, M. Schwarz, D. Gruss, and S. Mangard, "ScatterCache: Thwarting Cache Attacks via Cache Set Randomization," in *Proceedings of the USENIX Conference on Security Symposium*, 2019, pp. 675–692.

[82] S. Lukovic and N. Christianos, "Enhancing network-on-chip components to support security of processing elements," in *Proceedings of the Workshop on Embedded Systems Security*, 2010, pp. 1–9.

[83] S. Charles, Y. Lyu, and P. Mishra, "Real-Time Detection and Localization of Distributed DoS Attacks in NoC-based SoCs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, pp. 4510–4523, 2020.

[84] N. Binkert, R. Dreslinski, L. Hsu, K. Lim, A. Saidi, and S. Reinhardt, "The M5 Simulator: Modeling Networked Systems," *IEEE Micro*, vol. 26, pp. 52–60, 2006.

[85] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The Gem5 Simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, pp. 1–7, 2011.

[86] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood, "Multifacet's General Execution-Driven Multiprocessor Simulator Toolset," *SIGARCH Computer Architecture News*, vol. 33, pp. 92–99, 2005.

[87] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures," in *Annual IEEE/ACM International Symposium on Microarchitecture*, 2009, pp. 469–480.

[88] J. L. Henning, "SPEC CPU2006 Benchmark Descriptions," *SIGARCH Computer Architecture News*, vol. 34, pp. 1–17, 2006.

[89] J. Duato, S. Yalamanchili, and L. Ni, *CHAPTER 4 - Routing Algorithms*. Morgan Kaufmann, 2003.

[90] C. Glass and L. Ni, "The Turn Model for Adaptive Routing," in *International Symposium on Computer Architecture*, 1992, pp. 278–287.

[91] A. Das, S. Babu, J. Jose, S. Jose, and M. Palesi, "Critical Packet Prioritisation by Slack-Aware Re-Routing in On-Chip Networks," in *IEEE/ACM International Symposium on Networks-on-Chip*, 2018, pp. 1–8.

[92] R. Manju, A. Das, J. Jose, and P. Mishra, "SECTAR: Secure NoC using Trojan Aware Routing," in *IEEE/ACM International Symposium on Networks-on-Chip*, 2020, pp. 1–8.

[93] J. Lecler, Jean and G. Baillieu, "Application driven network-on-chip architecture exploration & refinement for a complex SoC," *Design Automation for Embedded Systems*, vol. 15, pp. 133–158, 2011.

[94] R. Manju, M. Choksey, and J. Jose, "Runtime Detection of Time-Delay Security Attack in System-an-Chip," in *IEEE/ACM International Workshop on Network on Chip Architectures*, 2022, pp. 1–6.

[95] R. Manju, A. Das, and J. Jose, "LOKI: A Hardware Trojan Affecting Multiple Components of an SoC," in *IEEE Computer Society Annual Symposium on VLSI*, 2022, pp. 176–181.

[96] (1961) Marvel comics. [Online]. Available: https://www.marvel.com/

LIST OF PUBLICATIONS

PUBLICATIONS FROM THESIS WORK

Book Chapter:

1. **Manju Rajan**, Abhijit Das, John Jose, and Prabhat Mishra, "Trojan-Aware Network-on-Chip Routing ", *Network-on-Chip Security and Privacy (NSP)*, Springer International Publishing, January 2021.
   DOI: 10.1007/9783030691318_11

Refereed Journals:

2. **Manju Rajan**, Mayank Choksey, and John Jose, "Secure Routing Framework for Mitigating Time-Delay Trojan Attack in System-on-Chip ", *Elsevier Journal of Systems Architecture (JSA)*, Volume 144, Issue 1, November 2023.
   DOI: 10.1016/j.sysarc.2023.103

3. **Manju Rajan**, Abhijit Das, John Jose "Fortifying System-on-Chip Security Against Trojan-Induced Packet Duplication Threats", *ACM Transactions on Design Automation and Electronics Systems (ACM TODAES). (Submitted).*

Refereed Conferences:

4. **Manju Rajan**, Mayank Choksey, and John Jose, "Runtime Detection of Time-Delay Security Attack in System-on-Chip ", *2022 15th IEEE/ACM International Workshop on Network on Chip Architectures (NoCArc)*, 2022, pp. 1-6.
   DOI: *10.1109/NoCArc57472.2022.9911380*

5. **Manju Rajan**, Abhijit Das, and John Jose, "LOKI: A Hardware Trojan Affecting Multiple Components of an SoC ", *2022 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), 2022, pp. 176-181.*
   DOI: *10.1109/ISVLSI54635.2022.00043*

6. **Manju Rajan**, Abhijit Das, and John Jose, "SECTAR: Secure NoC using Trojan Aware Routing ", *2020 14th IEEE/ACM International Symposium on Networks-on-Chip (NOCS), 2020, pp. 1-8.*
   DOI: *10.1109/NOCS50636.2020.9241711*

PUBLICATIONS OTHER THAN THESIS WORK

1. Vedika J Kulkarni, **Manju Rajan**, Ruchika Gupta, and John Jose, "Packet Header Attack by Hardware Trojan in NoC based TCMP and its Impact Analysis ", *2021 15th IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*, pp. 21-28, October 2021.

VITAE

Manju R commenced her Ph.D. journey in the Department of Computer Science and Engineering at the Indian Institute of Technology Guwahati, India, in July 2018. Throughout her tenure at IIT Guwahati, she was actively engaged with the Multi-core ARchitecture and Systems Lab (MARS Lab) within the Department of Computer Science and Engineering. She earned her Master of Technology in Computer Science and Engineering from Dr. M.G. R. Educational & Research Institute, Chennai, in 2006. Prior to this, she accomplished her Bachelor of Technology in Information Technology from Manonmaniam Sundarnar University, Tamil Nadu, in 2003. Her Ph.D. research is focused on tackling Hardware Trojan attacks in Network-on-Chip-based MPSoCs. Her work involves modeling various Trojan scenarios that could be integrated into NoC, along with the development of methods to detect and mitigate the impact of these Trojans, ultimately enhancing system performance. Beyond her primary focus, her research interests extend to multicore architecture and Quantum Computing.

Contact Information

E-mail: manju18@iitg.ac.in
manjurajanv@gmail.com

Phone: +91-9961330220

Website: https://www.iitg.ac.in/stud/186101012/