DESIGN OF SECURE AND TRUSTWORTHY NETWORK-ON-CHIP ARCHITECTURES

By

SUBODHA CHARLES

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

2020

ACKNOWLEDGMENTS

I would like to express my sincere appreciation to my advisor, Prof. Prabhat Mishra, who provided the persistent support and guidance for my Ph.D. study. His patience, motivation and immense knowledge helped me in all aspects of research and writing this dissertation. He is the person who made my Ph.D. research and this dissertation come true.

Besides my advisor, I would like to thank the rest of my Ph.D. committee members (Prof. Sartaj Sahni, Prof. My Thai, and Prof. Swarup Bhunia) for their constructive recommendations and insightful critiques. Their diverse expertise and knowledge helped me improve the quality of my research and this dissertation.

I thank my fellow labmates: Yuanwen Huang, Farimah Farahmandi, Yangdi Lyu, Alif Ahmed, Zhixin Pan, Daniel Volya, Hasini Witharana, Jonathan Cruz, Nikhil Venkatesh and Megan Logan. It was my great pleasure to collaborate with them.

Last but not least, I sincerely acknowledge the support and great love of my family and friends. This dissertation would not be possible without their unconditional support and love. I dedicate this dissertation to them.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Abstract of Dissertation Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy

DESIGN OF SECURE AND TRUSTWORTHY NETWORK-ON-CHIP ARCHITECTURES

By

Subodha Charles

August 2020

Chair: Prabhat Mishra
Major: Computer Science

Recent advances in chip manufacturing technologies have enabled computer architects to integrate an increasing number of processor cores and other heterogeneous components on a System-on-Chip (SoC). Network-on-Chip (NoC) is a promising solution and is widely employed by multicore architectures to cater to their communication requirements. The increased usage of NoC and its distributed nature across the chip have made it a focal point of potential security attacks. Trustworthy NoC architectures need to maximize security without violating design constraints. Moreover, the security solutions should be applicable for diverse technologies such as electrical, optical and wireless NoCs.

This dissertation focuses on developing lightweight security countermeasures that can provide the desired communication security and privacy with a tolerable impact on area, power and performance. Specifically, my research makes fundamental contributions in three major areas: (1) accurate modeling and optimization of NoC-based SoCs, (2) development of design-for-security solutions for on-chip communication, and (3) runtime monitoring to detect security threats. I propose several lightweight security countermeasures including incremental cryptography, trust-aware routing, anonymous routing and anomaly detection to address a wide variety of attacks such as eavesdropping, spoofing, packet tampering, and denial-of-service. My work proposes a reconfigurable NoC security architecture as well as novel NoC security solutions utilizing machine learning. Experimental results demonstrate that the proposed approaches can lead to trustworthy on-chip communication in resource-constrained SoCs.

CHAPTER 1
INTRODUCTION

We are living in the era of Internet-of-Things (IoT), an era in which the number of connected smart computing devices exceeds the human population. Various reports suggest that we can expect over 50 billion devices to be deployed and mutually connected by 2025 [1], compared to about 500 million in 2003 [2]. In the past, computing devices like phones with a few custom applications represented the boundary of our imagination. Today, we are developing solutions ranging from smartwatches, smart cars, smart homes, all the way to smart cities. System-on-Chip (SoC) designs are at the heart of these computing devices, which range from simple IoT devices in smart homes to complex navigation systems in airplanes. As applications grow increasingly complex, so do the complexities of the SoCs. For example, a typical automotive SoC may include 100-200 diverse Intellectual Property (IP) blocks designed by multiple vendors. The ITRS (International Technology Roadmap for Semiconductors) 2015 roadmap projected that the increased demand for information processing will drive a 30-fold increase in the number of cores by 2029 [3]. Indeed, one of the most recent many-core processor architectures, Intel "Knights Landing" (KNL), features 64-72 Atom cores and 144 vector processing units [4]. The Intel Xeon Phi processor family, which implements the KNL architecture, is often integrated into workstations to serve machine learning applications. The 256-core CPU - MPPA2, launched by Kalray Corporation [5], is used in many data centers to speed up data processing.

The increasing number of cores demands the use of a scalable on-chip interconnection architecture, which is also known as Network-on-Chip (NoC). As shown in Figure 1-1, a typical SoC utilizes NoC to communicate between multiple IP cores including processor, memory, controllers, converters, input/output devices, peripherals, etc. NoC IPs are used in a wide variety of market segments such as mobile phones, tablets, automotive and general purpose processing leading to an exponential growth in NoC IP usage. A survey done by Gartner Inc. has revealed that NoC IP sales of Sonics, a privately-held Silicon Valley IP provider that

specializes in NoC and power-management technologies, is ranked number 7 in terms of design IP revenue with a profit growth of 44.8% compared to 2013 [6]. Therefore, it is evident that the NoC has become an increasingly important component in modern SoC designs.

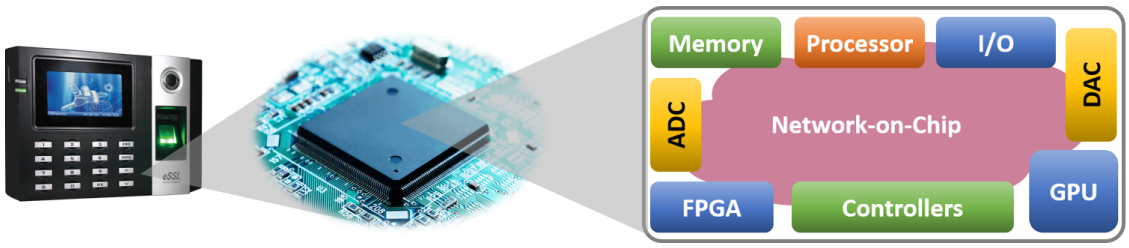


Figure 1-1. An example System-on-Chip (SoC) with Network-on-Chip (NoC) based communication fabric to interact with a wide variety of third-party Intellectual Property (IP) cores.

The drastic increase in SoC complexity has led to a significant increase in SoC design and validation complexity. Reusable hardware IP based SoC design has emerged as a pervasive design practice in the industry to dramatically reduce design and verification cost while meeting aggressive time-to-market constraints. Figure 1-2 shows the supply chain of a specific commercial SoC [7]. Growing reliance on these pre-verified hardware IPs, often gathered from untrusted third-party vendors, severely affects the security and trustworthiness of SoC computing platforms. These third-party IPs may come with deliberate malicious implants to incorporate undesired functionality (e.g. hardware Trojan), undocumented test/debug interfaces working as hidden backdoors, or other integrity issues. Based on Common Vulnerability Exposure estimates, if hardware-level vulnerabilities are removed, the overall system vulnerability will reduce by 43% [8, 9].

The security of emerging SoCs is becoming an increasingly important design concern. Beyond the traditional attacks from software on connected devices, attacks originating from or assisted by malicious components in hardware are becoming more common. For example, Quo Vadis Labs has reported backdoors in electronic chips that are used in weapon control systems and nuclear power plants [10], which can allow these chips to be compromised remotely. The well-publicized "Spectre" [11] and "Meltdown" [12] attacks highlight how sensitive data can

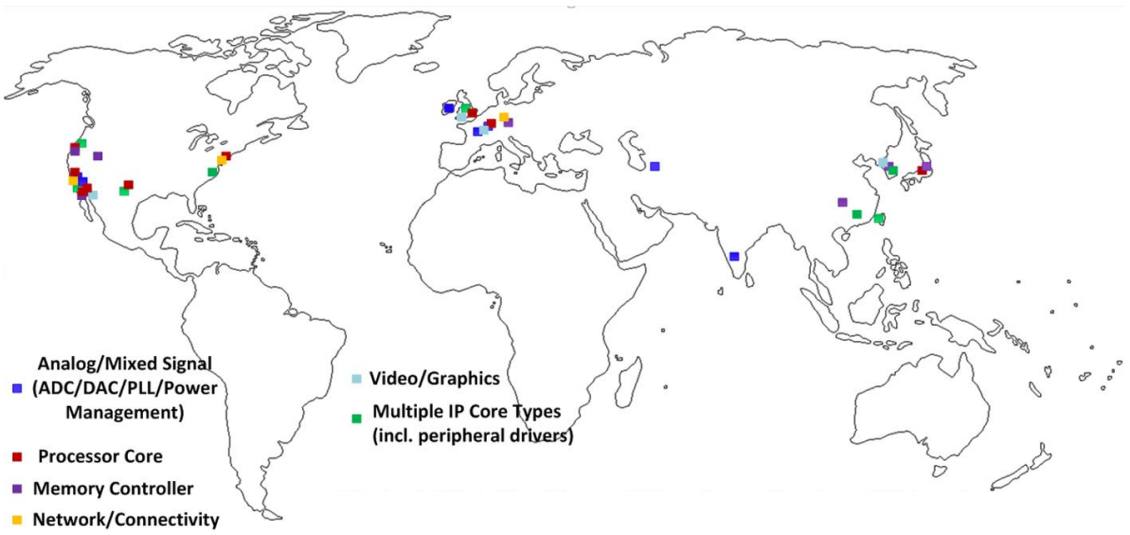


Figure 1-2. Supply chain of a commercial router SoC with components from multiple third-party companies across the globe.

be stolen from threads executing on multicore processors. It is widely acknowledged that all algorithmically secure cryptographic primitives and protocols rely on a hardware root-of-trust that is resilient to attacks to deliver the expected protections when implemented in software. Clearly, hardware platforms are at an elevated risk for security compromises in today's world.

In order to enable hardware-root-of-trust, we have to ensure that an SoC is trustworthy by ensuring security of computation, communication as well as storage. While the existing efforts have shown promising results in providing computation and storage related security solutions [7], there is limited effort in ensuring on-chip communication security. The ubiquity of devices using NoC-based SoCs has made NoC a focal point for security attacks as well as countermeasures. Therefore, in order to secure the cyberspace, it is vital to protect the NoC from potential security threats as well as leverage the advantages given by NoC to minimize security vulnerabilities of other system components.

A fundamental problem of NoC-based SoCs is ensuring security while preserving non-functional requirements such as performance, power and area. Due to the resource constrained nature of embedded and IoT devices, it may not be possible to implement traditional security measures such as encrypting text with the AES cipher and using SHA

hash functions. Thus, it is evident that considering security alone will not provide conclusive results. A more holistic approach is required that considers security among other non-functional requirements. In this dissertation, three main aspects of NoC-based SoCs are considered: (1) accurate modeling and optimization of NoC-based SoCs, (2) development of lightweight design-for-security solutions for on-chip communication, and (3) runtime monitoring to detect security threats.

This chapter is organized as follows. Section 1.1 provides an overview of NoC architectures. Section 1.2 describes the NoC security landscape. Section 1.3 highlights my research contributions. Finally, Section 1.4 outlines the organization of the dissertation.

1.1 Overview of Network-on-Chip (NoC) Architectures

Consider a designer who is responsible for designing the road network of a large city. Roads should be laid out giving easy access to all the offices, schools, houses, parks, etc. If all of the most common places are situated close to each other, it is inevitable that the roads in that area will get congested and other areas will be relatively empty. The designer should make sure that such instances do not occur and the traffic is uniformly distributed as much as possible. Alternatively, the roads should have more lanes and parking lots in such congested areas to cater to the requirement. In addition to accessibility and traffic distribution, the architect should also consider intersections, traffic lights, priority lanes, and potential detours due to occasional road maintenance. Moreover, self driving cars and drones that deliver various items might come into picture in the future as well. Analogous to this, the designer of an SoC faces a similar set of challenges when designing the communication infrastructure connecting all the cores.

The early SoCs employed bus and crossbar based architectures. Traditional bus architecture has dedicated point-to-point connections, with one wire dedicated to each component. When the number of cores in an SoC is low, buses are cost effective and simple to implement. Buses have been successfully implemented in many complex architectures. ARM's AMBA (Advanced Micro-controller Bus Architecture) bus [13] and IBM's CoreConnect [14] are

two popular examples. Figure 1-3 shows an overview of the ARM AMBA bus architecture [13]. Buses do not classify activities depending on their characteristics. For example, the general classification as transaction, transport and physical layer behavior are not distinguished by buses. This is one of the main reasons why they cannot adapt to changes in architecture or make use of advances in silicon process technology. Due to increasing SoC complexity coupled with increasing number of cores, buses often become the performance bottleneck in complex SoCs. This coupled with other drawbacks, such as non-scalability, increased power consumption, non-reusability, variable wire delay, and increased verification cost, motivated researchers to search for alternative solutions.

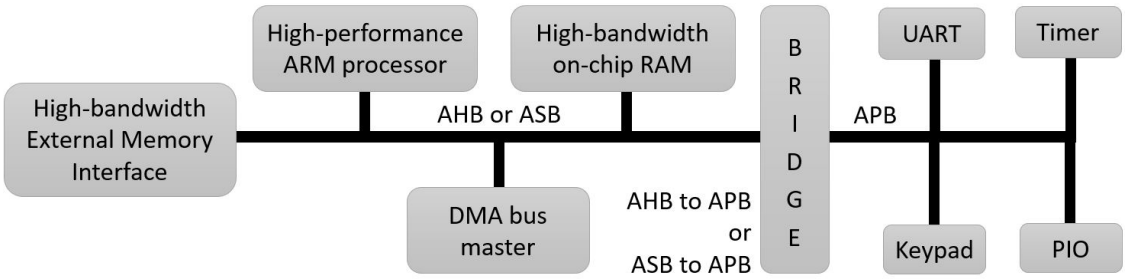


Figure 1-3. Overview of the ARM AMBA bus architecture.

The inspiration for network-on-chip (NoC) came from traditional networking solutions, more specifically, the Internet. The NoC, a miniature version of the wide area network with routers, packets and links, was proposed as the solution for on-chip communication [15, 16]. The new paradigm described a way of communicating between IPs including features such as routing protocols, flow control, switching, arbitration and buffering. With increased scalability, resource reuse, improved performance and reduced costs, NoC became the solution for the complex SoCs that required a scalable interconnection architecture. The remainder of this section covers various aspects of NoC architectures.

1.1.1 Network-on-Chip Architecture and Communication Protocol

Figure 1-4 shows an example NoC interconnection architecture consisting of several processing elements connected together via routers and regular sized wires (links). A processing element can be any component such as a microprocessor, an ASIC (application

specific integrated circuit), or an intellectual property block that performs a dedicated task as shown in Figure 1-1. Without loss of generality, in this dissertation, we call processing elements as IPs. IPs are connected to the routers via a network interface (NI). We call the combination of an IP, an NI and a router as a "node" in the NoC. It can be observed that words node and "tile" are used interchangeably in existing literature to refer to NoC components connected to one router [4, 17].

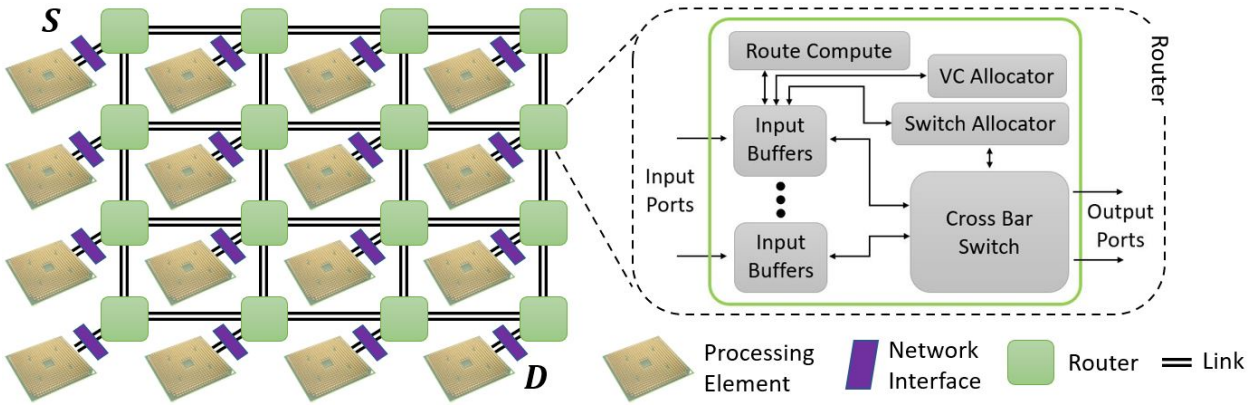


Figure 1-4. Example of an NoC connecting 16 IPs.

NoC interconnection architecture uses a packet-based communication approach. A request or response that goes to a cache or to off-chip memory is divided into packets, and subsequently to "flits", and injected to the network. A flit is the smallest unit of flow control in an NoC. A packet may consist of one or more flits. For example, assume S is a processor IP whereas node D is connected to an off-chip memory interface (memory controller). When a load instruction is executed at S, it first checks the private cache located in the same node and if it is a cache miss, the required data has to be fetched from the memory. Therefore, a memory fetch request message is created and sent on the appropriate virtual network to the NI. The network interface then converts it into network packets according to the packet format, fliticizes the packets and sends the flits into the network via the local router. The network is then responsible to route the flits to the destination, D. Flits are routed either along the same path or different paths depending on the routing protocol. The NI at D creates the packet from the received flits and forwards the request to D, which then initiates the memory

fetch request. The response message from the memory that contains the data block follows a similar process. Similarly, all IPs integrated in the SoC leverage the resources provided by the NoC to communicate with each other. Figure 1-5 shows an overview of this process. Figure 1-6 shows the format of a memory request packet and a response data packet used in the gem5 architectural simulator [18].

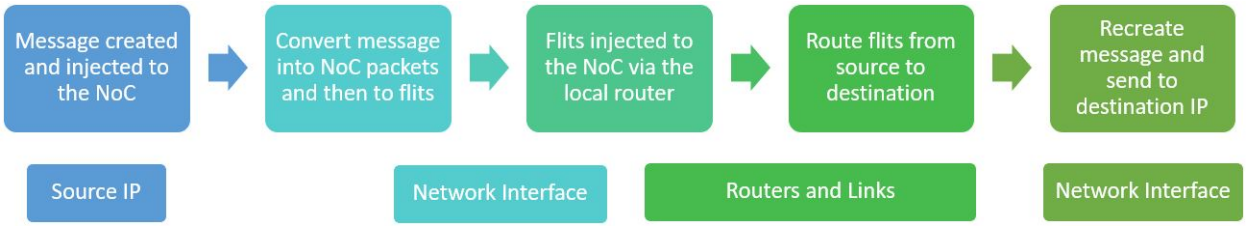
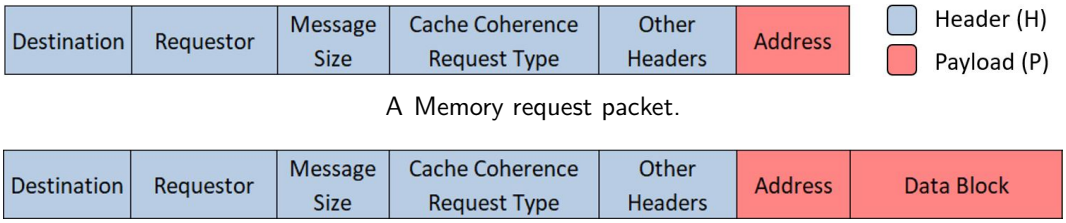


Figure 1-5. Overview of a NoC traversal.



A Memory request packet.

B Response data packet from memory.

Figure 1-6. NoC control (memory request) and data (response data) packet formats used in the gem5 simulator.

Previous works have proposed several NoC architectures such as Nostrum [19], SOCBUS [20], Proteo [21], Xpipes [22], Æthereal [23], etc. based on different requirements. The choice of the parameters in the architecture depends on the design requirements such as performance/power/area budgets, reliability, quality-of-service guarantees, scalability and implementation cost. Some of the existing NoC architectures have been surveyed in literature [24, 25]. NoC architecture design needs to consider two important factors - network topology and routing protocol. The next two subsections describe these aspects in detail.

1.1.1.1 Network topology

The topology defines the physical organization of IPs, routers and links of an interconnect. The organization in Figure 1-4 shows a mesh topology. Crossbar, point-to-point, tree, 3-D

mesh are few other commonly used topologies. Figure 1-7 shows some examples of them. The topology is chosen depending on the cost and performance requirements of an SoC. The topology directly impacts the communication latency when two IPs are communicating, since it affects the number of links and routers a flit has to traverse through to reach a given destination. A major trade-off when deciding the topology for a given requirement is between connectivity and cost. Higher connectivity (e.g., point-to-point) allows increased performance, but has higher area and power overhead. The 2-D mesh is the most common topology in NoC designs [4, 17]. Each link in a mesh has the same length leading to ease of design, and the area occupied by the mesh grows linearly with the number of nodes.

1.1.1.2 Router and routing protocol

The routers comprise of input buffers that accept packets from the local IP via the NI or from other routers connected to it. For example, in the mesh topology, except for the routers in the border, each router is connected to the local IP and four other routers. Based on the addresses in the packet header and the routing protocol, the crossbar switch routes data from the input buffers to the appropriate output port. Buffers are allocated for virtual channels which helps avoid deadlock. The switch allocator handles input port arbitration for output ports [26].

The routing protocol defines the path a flit should take in a given topology. Routing protocols can be broadly classified as deterministic and adaptive. In deterministic routing, each packet traversing from S to D follows the same path. X-Y routing is one common example of deterministic routing. In X-Y routing, packets use X-directional links first, before using Y-directional links [27]. An example including three paths taken by X-Y routing in a mesh NoC is shown in Figure 1-7. Adaptive routing takes network states such as congestion, security, and reliability into account, and sends the flits through different paths based on the current state of the network [28].

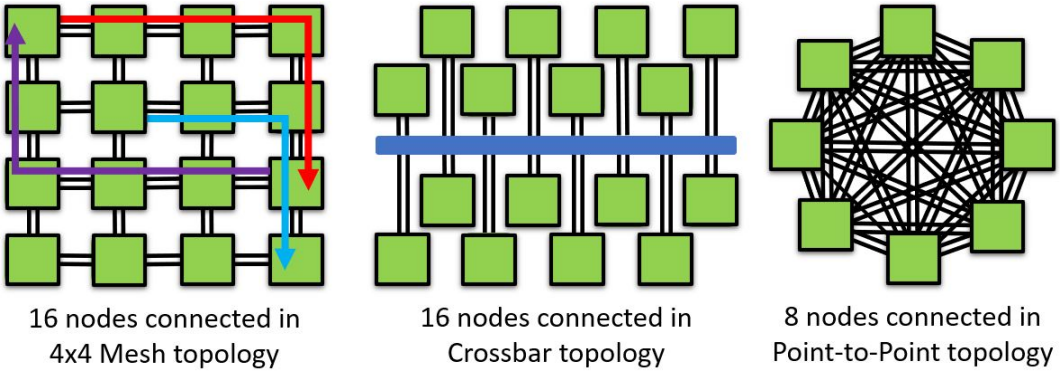


Figure 1-7. NoC topologies and an example of X-Y routing in a mesh NoC.

1.1.2 Emerging NoC Technologies

When NoC was first introduced, the focus was on electrical (copper) wires connecting NoC components together, referred to as "electrical NoC". However, recent advancements have demanded exploration of alternatives. With the advancement of manufacturing technologies, the computational power of IPs have increased significantly. As a result, the communication between SoC components have become the bottleneck. Irrespective of the architectural optimizations, electrical NoC exhibits inherent limitations due to the physical characteristics of electrical wires [29].

- The resistance of wires, and as a result, the resistance of NoC, is increasing significantly under combined effects of enhanced grain boundary scattering, surface scattering, and the presence of a highly resistive diffusion barrier layer [30, 31].
- Electrical NoC can contribute a significant portion of the on-chip capacitance. In some cases, about 70% of the total capacitance [32].
- The electrical NoC is a major source of power dissipation due to the above two factors.

Therefore, it is becoming increasingly difficult for electrical NoC to keep up with the delay, power, bandwidth, reliability and delay uncertainty requirements of state-of-the-art SoC architectures [33, 34]. These challenges can only intensify in future giga and tera-scale architectures. In fact, the International Technology Roadmap for Semiconductors (ITRS) has mentioned optical and wireless based on-chip interconnect innovation to be key to addressing these challenges [35].

Recent years has seen the introduction of emerging NoC technologies such as "wireless NoC" [36] and "optical NoC" [37]. While the focus of this dissertation is on security attacks and countermeasures in electrical NoCs, a majority of these security solutions are also applicable for wireless and optical NoCs. This is primarily due to the fact that they have inherent similarities in terms of network topology and routing protocols. For example, both electrical and optical NoCs represent similar topologies using wired connectivity. Similarly, wireless NoC always use one-hop routing, while optical and electrical NoCs utilize one-hop or muti-hop communication depending on the source and destination. Figure 1-8 shows an overview of how different NoC technologies can be used to connect heterogeneous SoC components. In the rest of the disseration, we use the word NoC to refer to electrical NoC unless otherwise specified as Wireless NoC or Optical NoC.

1.1.2.1 Wireless NoC

Wireless NoC was proposed as a solution to the latency experienced by electrical NoCs, which are based on metal interconnects and multi-hop communication. Wireless NoC integrates on-chip antennas and suitable transceivers that enable communication between two IPs without a wired medium. Silicon integrated antennas communicating using the millimeter wave range is shown to be a viable technology for on-chip communication [36].

1.1.2.2 Optical NoC

On the other hand, optical NoC, also known as photonic NoC, uses photo emitters, optical wave guides and transceivers for communication [38]. The major advantage over electrical NoC is that it is possible to physically intersect light beams with minimal crosstalk. This enables simplified routing and together with other properties, optical NoC can achieve bandwidths in the range of Gbps.

1.2 Security Landscape in NoC Based System-on-Chip

The widespread adaptation of NoCs has made it a focal point for security attacks as well as countermeasures. There is a growing interest in the industry to use the NoC to secure the SoC as evident from NoC-Lock [39] and FlexNoC resilience package [40]. On

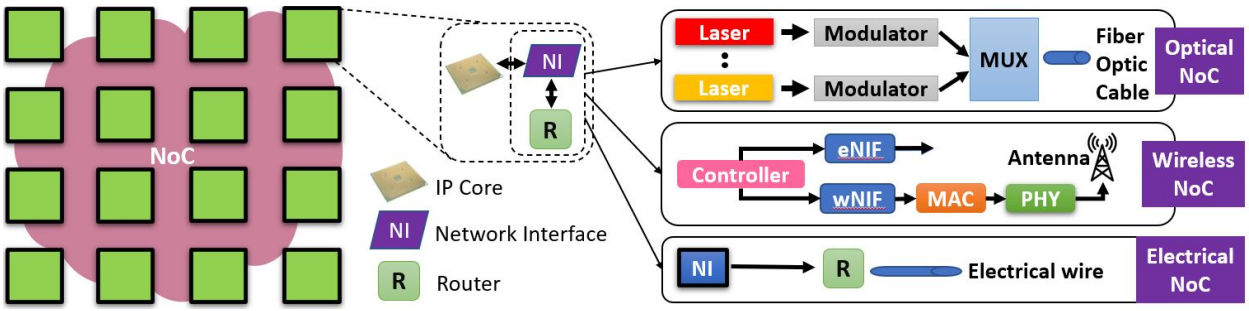


Figure 1-8. NoC enables communication between IPs. The network interface (NI), router (R) and links can be implemented using optical, wireless or electrical communication technologies.

the other hand, the NoC itself can be a threat when different IP blocks come from different vendors. A compromised NoC IP can corrupt data, degrade performance or even steal sensitive information. NoC security is crucial for three related reasons: i) NoC has access to all system data, ii) NoC spans across the entire SoC, and iii) NoC elements are repetitive in a way that any modification can be easily replicated. In the following subsections, we discuss how SoCs can become vulnerable to security threats (Section 1.2.1), why securing NoC based SoCs has become a hard problem (Section 1.2.2) and different threat models in existing literature related to NoC security (Section 1.2.3).

1.2.1 Security Vulnerabilities in SoCs

SoC complexity and tight time-to-market deadlines have shifted the in-house SoC manufacturing process to a global supply chain. SoC manufacturers outsource parts of the manufacturing process to third-party IP vendors. This globally distributed mechanism of design, validation and fabrication of IPs can lead to security vulnerabilities. Adversaries have the ability to implant malicious hardware/software components in the IPs. Existing literature has discussed three forms of vulnerabilities: (i) malicious implants, (ii) backdoor using test/debug interfaces, and (iii) unintentional vulnerabilities [41]. An adversary can utilize the malicious implants (hardware Trojans) to cause malfunction or facilitate information leakage [7]. An adversary can also exploit legitimate test and debug interfaces as a backdoor for information leakage [10]. Many security vulnerabilities can be created unintentionally by

design automation/computer-aided design (CAD) tools or by designers' mistakes [42]. These vulnerabilities can lead to untrusted (potentially malicious) IPs.

Attacks based on malicious implants, such as hardware Trojans, rely on Trojans being integrated in the SoC without being detected at the post-silicon verification stage or during runtime [7]. Hardware Trojans can be inserted into the design in several places such as by an untrusted CAD tool or designer or at the foundry via reverse engineering [43]. Even if all the IPs are tested before integration, hardware Trojans can still go undetected because of the complexity of designs with billions of transistors which make physical inspection or 100% coverage in design verification/validation a costly or even impossible target [44]. Furthermore, Trojans can mask their behavior as transient errors and can be activated only when a specific condition or a combination of conditions are satisfied [45]. A smart attacker can carefully craft the Trojan activation method so that it becomes difficult to detect. Previous work has discussed external/internal Trojan activation modes [44], software-hardware coalition [46] and triggers based on time, input sequence, traffic pattern, and even thermal conditions [45].

The usage of third party NoC IPs has grown rapidly over the years. Due to the widespread use of NoC IPs, outsourcing NoC IP fabrication has become a common practise. iSuppli, an independent market research firm, has concluded from their research that the FlexNoC on-chip interconnection architecture [40] is used by four out of the top five Chinese fabless semiconductor OEM (original equipment manufacturer) companies [47]. This has led to Arteris, the company that developed FlexNoC, achieve a sales growth of 1002% over a three year time period through IP licensing [48]. Therefore, there is ample opportunity for adversaries to attack the SoC through malicious implants in NoC IPs. Furthermore, due to the complexity of the design, NoC IPs are ideal candidates to insert hardware Trojans [49].

1.2.2 Unique Challenges in Securing NoC-based SoCs

The general problem of securing the interconnect has been well studied in the computer networks domain and other related areas [50–52]. However, implementation of security features introduce area, power and performance overhead. While computer networks domain can allow

for complex security countermeasures, the resource constrained nature of embedded and IoT devices pose additional unique challenges as outlined below.

1.2.2.1 Conflicting requirements

While enabling communication between IPs, NoCs need to satisfy a wide variety of requirements including security, privacy, energy efficiency, domain-specific requirements and real-time constraints. It is difficult to satisfy conflicting requirements such as security and energy efficiency. For example, it may not be possible to implement traditional security measures such as encrypting text with the AES cipher and using SHA hash functions in resource-constrained IoT devices. Similarly, security and domain-specific requirements may not be compatible. For example, in an automotive network, when a potential security breach is detected, pausing all systems to check the malfunction is not an option since the car is moving, and stopping it abruptly can lead to catastrophic consequences. Thus, there is a need for innovative solutions to secure NoCs with lightweight security mechanisms customized for application domains.

1.2.2.2 Increased complexity

The complexity of SoC designs have made exhaustive security validation an impossible task. Most IPs come as black boxes from vendors that do not reveal design details in order to maintain the competitive advantage in a niche market. As a result, the complete design is not visible to verification engineers. Modern verification tools often try to detect missing or erroneous functionality whereas security vulnerabilities can be hidden in dormant functions in large and complex designs that gets triggered only by a specific set of inputs as discussed in Section 1.2.1. Therefore, it is not feasible to capture all security vulnerabilities using security validation tools during design time.

1.2.2.3 Diverse technologies

While electrical communication is widely used in designing NoC based SoCs, emerging NoCs can also support chip-scale photonics (optical NoC) as well as wireless communication (wireless NoC) as shown in Figure 1-8. Security solutions for NoCs thus need to not only

address security over electrical wires, but also consider the emerging challenges from data transfers over photonic waveguides and wireless channels. While broadcast may be preferred for wireless NoCs, optical and electrical NoCs need to consider a wide variety of network topologies as well.

1.2.3 Threat Models

The intention of a hardware Trojan can vary from design to design. Commonly discussed threats include information leakage, denial-of-service and data corruption. A recent occurrence of a hardware Trojan spying on data, raised concerns across top US companies and authorities including Apple, Amazon and CIA [53]. In this section, we provide an overview of five classes of attacks on NoC based SoCs (Figure 1-9).

Figure 1-9. Five classes of security attacks discussed in existing literature.

These classes of attacks have been well studied in the computer networks domain and other related areas. However, implementation of security features introduce area, power and performance overhead. While computer networks domain can allow for complex security countermeasures, the resource constrained nature of embedded and IoT devices pose additional unique challenges as outlined above. To address this issue, in this dissertation, I present lightweight security countermeasures that can provide the desired security with tolerable impact on area, power and performance. In the remainder of this section, I provide an overview of attacks explored in NoC based SoCs (Chapter 2 will provide a detailed discussion on related efforts).

1.2.3.1 Eavesdropping attacks

Eavesdropping attack, also known as snooping/sniffing, refers to an attacker passively listening to on-chip communication in an attempt to steal sensitive information. The intention of the attacker is to leak information over long time periods without being detected. Recent

occurrences of hardware security breaches where hard-to-detect hardware components, that were not a part of the original design, integrated into the original design leaking information have attracted more attention to eavesdropping attacks [53].

1.2.3.2 Spoofing and data integrity attacks

SoC relies on the integrity of data communicated through the NoC for correct execution of tasks. If a malicious agent corrupts data intentionally, it can lead to erroneous execution of programs as well as system failures. On the other hand, spoofing is the act of disguising a communication from an unknown source as being from a known (trusted) source. Therefore, a malicious agent pretending to be a trusted source can inject new packets to the network causing system to malfunction. Spoofing can be used to bypass memory access protection by impersonating a core that has permission to read from (or write in) prohibited regions to steal sensitive information or disrupt execution. Spoofing may also be leveraged to respond to legitimate requests with wrong information to cause system failure. Spoofing can be achieved by an attacker replacing the source address of a packet by an address of a trusted IP.

1.2.3.3 Denial-of-service attacks

Denial-of-service (DoS) in a network is an attack that prevents legitimate users from accessing services and information. The most common example is an attacker flooding a network with information. When a user is trying to access a website, the request is sent to that web server to view the page. The server has a certain bandwidth and can only serve a limited number of requests at a time. If the attacker overloads the server with requests, it will not be able to process the user's legitimate request. This is "denial of service". In the context of an NoC, several threat models have been explored. In general, DoS in NoC based SoCs are attacks that overwhelm the network resources in an attempt to cause performance degradation, real-time guarantee violations and reduction of battery lifetime.

1.2.3.4 Buffer overflow and memory extraction attacks

The goal of a buffer overflow attack is to alter the function of a privileged program so that the attacker can gain access and execute his own code. A program with high privileges

(root programs) typically become the target of buffer overflow attacks. To accomplish this, the adversary has to insert malicious code and make the program execute it. "Code injection" is the first step to accomplish this where the malicious code is inserted into the privileged program's address space. This can be achieved by providing a string as input to the program which will be stored in the program buffer. The string will contain some root level instructions which the adversary wants the program to execute [54]. Then, the adversary creates an overflow in the program buffer to alter states of the program. For example, it can alter a return address of a function so that the program will jump to that location and start executing the malicious code [55]. This can be accomplished when buffers have weak or no bound checking. Buffer overflow attacks can also be used to read privileged memory locations from the address space. In an NoC context, the threat gets aggravated due to memory spaces being shared between multiple cores.

1.2.3.5 Side channel attacks

Side channel attacks exploit non-functional behavior such as time, power, electromagnetic radiation, heat and acoustic waveforms to attack a secure system [56]. The switching behavior of the CMOS (complementary metal oxide semiconductor) transistors can be analyzed to infer the underlying circuit functionality. Therefore, even a flawless implementation of a security mechanism can be vulnerable against side channel attacks. For example, Zhen et al. presented a method to implement a timing attack on Nvidia Kepler K40 GPU and successfully recovered the complete 128-bit AES encryption key [57]. In contrast, a paper published in 2012 showed that a brute-force attack on AES using a super computer can take 149 trillion years [58]. Even though computing resources have significantly improved since then, a brute-force attack on AES-128 is still not possible. Possibility of side channel attacks escalated, since in a realistic scenario, more constraints are imposed on the system such as performance and power. Even for systems with theoretically proven security bounds, revealing the secrets through these non-functional physical properties is a likely scenario.

1.3 Research Contributions

My research proposes novel techniques to address the security challenges in the following three broad categories.

1. **Design-for-Security:** I have proposed a set of security solutions that integrate security mechanisms to the chip during design time to mitigate threats rather than trying to detect attacks during runtime.

2. **Runtime Monitoring:** In addition to the attacks that can be mitigated by design-for-security solutions, runtime threat detection and localization techniques are required to further neutralize other threats.

3. **Modeling and Evaluation:** I have developed accurate models of commercial SoCs in architecture simulators to enable exploration of optimization opportunities as well as realistic evaluation of lightweight security countermeasures.

In other words, I propose to add security at two stages of the design cycle. The first set of proposed approaches would try to create design-for-security solutions that would make it hard or impossible for attackers to install malicious implants. The next step would be to perform runtime monitoring in case the first layer of defense was not enough or an attacker exploited specific runtime vulnerabilities.

Figure 1-10 outlines the major research contributions of the dissertation. The research listed in the left branch focuses on accurate modelling of commercial SoCs in architecture simulators and exploring potential optimization opportunities that will enable the exploration and evaluation of lightweight security countermeasures. The middle branch lists research focused on developing lightweight design for security solutions for NoC security. The right branch contains research that outlines runtime monitoring solutions to detect threats during SoC operation. The remainder of this section provides a brief overview of my contributions.

- **Accurately modeling of NoC architectures:** The NoC performance and power consumption depend critically on the traffic load. The network traffic itself is a function of not only the application, but also the cache coherence protocol, and memory controller/directory locations. To accurately measure the impact of any security mechanism or optimization technique, it is important to capture the traffic behavior accurately, and model them in architecture simulators. This dissertation shows that using unrealistic models in a widely used multiprocessor simulator produce misleading power

and performance predictions, whereas accurate modeling of NoC traffic behavior can produce results comparable with hardware platforms.

- **NoC-aware cache reconfiguration and exploration:** Dynamic cache reconfiguration (DCR) is an effective technique to optimize energy consumption in many-core architectures. While early work on DCR has shown promising energy saving opportunities, prior techniques are not suitable for NoC based SoCs since they do not consider the interactions and tight coupling between memory, caches and NoC traffic. In this dissertation, I propose an efficient cache reconfiguration framework in NoC based SoCs. My approach can reduce energy consumption significantly and the proposed machine learning based exploration framework can reduce the exploration time by an order of magnitude with negligible loss in accuracy.

- **Incremental cryptography for NoC communication:** On-chip communication has to be obfuscated and checked for integrity to ensure that malicious IPs do not eavesdrop/tamper with communication between IPs while meeting the desired power and performance targets (including real-time constraints). Traditional encryption (AES) and authentication (message authentication based on complex hash functions) schemes can incur significant performance and energy overhead, and as a result, may not be suitable in many scenarios. Therefore, it is crucial to develop a lightweight security architecture that can provide the desired security with acceptable overhead. In this dissertation, I propose a lightweight encryption scheme that leverages on the unique traffic characteristics of NoC. My approach uses the similarity between consecutive packets and uses an encryption scheme based on incremental encryption.

- **Lightweight encryption and anonymous routing:** While AEAD (Authenticated Encryption with Associated Data) schemes can protect sensitive data from eavesdroppers while ensuring the integrity of packets, the header fields, which are referred to as "associated data" is sent as plaintext. Attackers can use this header information to identify packets from the same information flow and launch more complex attacks such as linear and differential cryptanalysis to recover the plaintext from the encrypted portion. If the header field is encrypted, the intermediate routers have to decrypt headers to learn the next hop of the packet, which can lead to unacceptable performance and energy overhead. I propose a lightweight anonymous routing scheme that provides anonymity of source and destination as well as privacy of sensitive data in the packet. My approach uses a secret sharing based encryption scheme which allows us to eliminate traditional key-based encryption during packet transfer leading to better performance and energy efficiency.

- **Runtime detection & localization of DoS attacks:** Distributed denial-of-Service (DDoS) is an attack that is caused by one or more malicious IPs (MIPs) flooding the network with unnecessary packets causing significant performance degradation through NoC congestion. In this dissertation, I propose a lightweight and real-time DDoS attack detection mechanism. Once a potential attack has been flagged, my approach is also capable of localizing the MIPs using latency data gathered by NoC components. My

approach is capable of timely attack detection and localization while incurring minor area and power overhead.

- **Trust-aware routing in the presence of malicious IPs:** Even though authentication schemes can ensure data integrity, a MIP integrated on the NoC can tamper packets leading to performance and energy penalties. Continuous tampering can even lead to DoS attacks. While message authentication code (MAC)-based authentication schemes can detect tampering, if packets do not traverse through the MIPs, tampering can be avoided altogether. I develop a trust evaluation scheme where IPs can develop trust about neighbouring routers and decide which routing path will minimize the risk of getting a packet tampered. Trust values are effectively propagated to build local and global trust about the NoC components. My approach leads to less tampered packets, less retransmissions and as a result, improved performance and energy efficiency.

- **Reconfigurable network-on-chip security architecture:** In the early days, the IoT and embedded devices were intended for a single or very few use cases. The requirements and working conditions were well defined and predictable. Therefore, it was easy to make design choices to fit the requirements. In comparison to that, the devices manufactured today are intended to serve general purpose applications that are diverse and sometimes, not yet defined. Therefore, it is not possible to statically optimize the devices to fit each use case. In this dissertation, I plan on exploring a reconfigurable security architecture for NoC based SoCs that can be dynamically reconfigured depending on use-case scenarios. My approach seamlessly integrates several security primitives and proposes a mechanism to choose between them during runtime.

- **Digital watermarking for detecting malicious IPs:** Hardware security breaches due to third-party vendors aiming at industrial espionage have recently raised many concerns. Such eavesdropping attacks have been discussed in an NoC context as well. In particular, in an eavesdropping attack, a Trojan infected router copies packets transferred through the NoC and re-routes the duplicated packets to an accompanying malicious application running on another IP in an attempt to extract confidential information. While authenticated encryption can thwart such attacks, it incurs unacceptable overhead in resource-constrained SoCs. In this dissertation, I propose a lightweight alternative defense based on digital watermarking techniques. I develop theoretical models to provide security guarantees and validate them with experimental results. My approach can significantly outperform state-of-the-art methods.

- **Securing NoC using machine learning:** Machine learning (ML) techniques have proven to be effective at addressing security vulnerabilities. While computer networks domain has effectively utilized ML to secure networks, the intersection of ML and NoC security has never been explored before. In this dissertation, I propose an ML-based DoS attack detection framework that trains ML models during design time and uses the trained models to classify network traffic behavior as attack or normal execution. I extensively evaluate ML models and features that can be extracted/engineered from NoC traffic to provide high accuracy in DoS attack detection. My approach is capable

of adapting to several use cases with unpredictable NoC traffic patterns and detecting attacks with high accuracy.

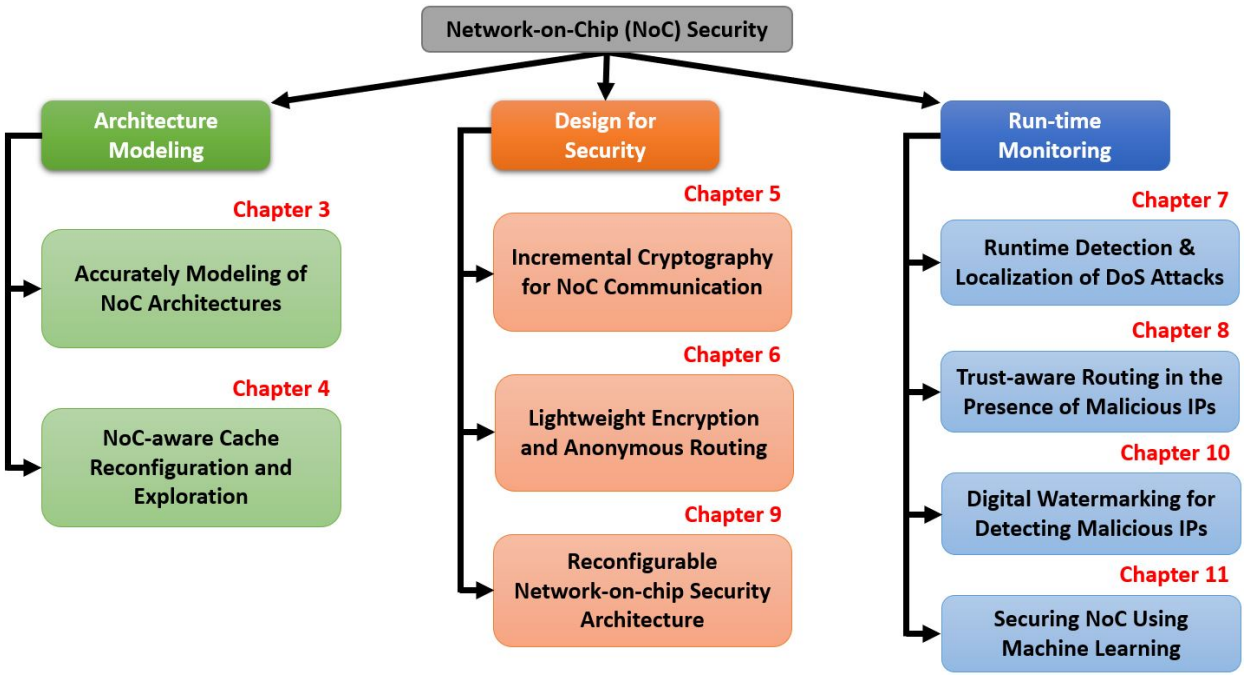


Figure 1-10. Dissertation Outline

1.4 Dissertation Organization

The rest of the dissertation is organized as follows. Chapter 2 surveys existing work related to NoC security. Chapter 3 presents the simulator model for accurately capturing NoC characteristics. Chapter 4 explores energy optimization opportunities using the accurate NoC model. Chapter 5 proposes a lightweight encryption mechanism based on incremental encryption. Chapter 6 illustrates an anonymous routing and encryption mechanism that leverages on secret sharing. Chapter 7 presents a real-time DoS attack detection and localization mechanism when MIPs launch DoS attacks on critical NoC components. Chapter 8 presents an effective trust-aware routing among NoC components to avoid MIPs. Chapter 9 presents a reconfigurable security framework that combines several security primitives and allows a choice between them depending on the security requirements. Chapter 10 presents a lightweight defense for eavesdropping attacks based on digital watermarking. Chapter 11 presents a machine learning based framework for detecting DoS attacks. Finally, Chapter 12 outlines future research directions and concludes this dissertation.

CHAPTER 2
RELATED WORK

Ever since the introduction of Network-on-chip (NoC) about two decades ago, researchers and industry have carried out research and development in several directions. In this chapter, I outline prior work related to NoC security. First, I outline previous work related to NoC traffic (Section 2.1) and SoC energy (Section 2.2) optimizations in an attempt to highlight the importance of NoC optimization for resource-constrained SoCs. In the subsequent five sections (Section 2.3 - Section 2.7), I present previous research efforts related to the five classes of security attacks on NoC-based SoCs introduced in Section 1.2.3.

2.1 NoC Traffic Optimization

Prior work on NoC traffic exploration and optimization motivates the need for better memory and processor placement to reduce contention and latency. Early work in this area suggests that the efficient distribution of memory traffic to provide quality-of-service guarantees [59]. Abts et al. [60] tackle the problem of optimum memory controller (MC) placement where m cores need to be placed with n MCs. The placement is decided by examining the variation in latency experienced by cores to access each MC. "Diamond" placement is found to be the best for an 8x8 mesh with 16 MCs, while further improvements are achieved by introducing a class-based deterministic routing algorithm. Xu et al. [61] leverage this idea to find an optimal placement for the same configuration. The minimum number of MCs and their placement required to achieve a given performance goal was explored by taking Intel SCC [62] as a case study [63]. Once the number of MCs are decided and placed, it creates opportunity for optimization by dynamically mapping workload data to appropriate MCs [64].

The effect of modeling the main memory access through the directory was discussed by Duraisamy et al. [65]. They explore the traffic patterns of two-level MESI directory protocol and AMD's Hammer-based HyperTransport (HT) [66] protocol to design an efficient multicast aware wireless NoC. Ros et al. analyzed area and traffic trade-offs associated with

cache coherence protocols [67]. To optimize power and performance, Schuchhardt et al. [68] proposed a method to place directories closer to their shared data and thereby eliminating many network traversals. Other coherence traffic-based optimization techniques include coherence protocol deactivation for private block accesses to reduce directory accesses [69], and a bloom filter mechanism for tagless coherence directory [70].

None of the prior efforts rigorously explore the affinity between IPs, MC and directory in a system running directory-based cache coherence and optimization with different cluster and memory modes. Hence, my proposed model in Chapter 3 is vital and crucial for emerging NoCs with wireless [65], optical [71] and 3D networks [72].

2.2 Energy Optimization of Many-core Architectures

Reconfigurable cache architectures have been extensively studied utilizing techniques, such as way shutdown, way management, cache partitioning and resizing [73–76]. Settle et al. [77] introduced a reconfigurable cache architecture specific for CMPs. These architectures were used to explore cache reconfiguration techniques in several orthogonal directions ranging from single core [78], multi core [79, 80], realtime [81], and embedded [82] systems. Cache partitioning (CP) techniques are mainly focused on improving performance of many-core systems [83, 84]. Beckmann et al. proposed JIGSAW, a cache organization method that addresses scalability and interference issues of shared caches. However, their work only discussed reconfiguration in L2 cache. As shown in [82] and [80], reconfiguring L1 would change the L2 traffic and therefore, requires simultaneous L1 and L2 reconfiguration. The existing L1/L2 reconfiguration methods are primarily based on static and/or dynamic analysis [81, 82]. These methods explore various configurations and decide which one to use depending on application characteristics. If application characteristics are known a priori, static analysis is beneficial since dynamic analysis can pose significant overhead and lead to unpredictable performance impact. However, dynamic analysis of a limited set of configurations is the only option when static profiling is not feasible. The above approaches can lead to unrealistic results

in NoC-based many-core architectures since they do not consider the energy impact of NoC traffic during exploration.

Studies on NoC traffic exploration proves the fact that NoC traffic largely affects the CMP PnP statistics [85]. Several studies were carried out in efficient distribution of memory traffic to provide quality-of-service guarantees [59], optimum memory controller placement [61] and task scheduling [68]. Combining the effects of NoC communication to overall energy consumption, Chen et al. proposed to dynamically turn on and off L2 cache banks to save energy in optical NoCs which has silicon-photonic links [86]. However, there has been limited effort on SoC energy optimization considering the NoC energy component and its variations due to other optimization techniques such as DCR/CP.

One of the major concerns in static and dynamic analysis of possible cache configurations is the exploration time. The exploration space grows significantly with the number of tunable cache parameters and levels of cache (L1, L2 etc.). In order to reduce the exploration complexity, heuristic-based approaches [78, 87] consider only a small set of potential configurations based on specific observations (such as independence or priority of specific parameters). While these approaches can reduce the exploration time, the quality of results can be far from the optimal. The approach presented in Chapter 4 utilizes machine learning to drastically reduce the exploration time with minor (acceptable) loss in the quality of results. My work provides a coherent framework that enables the exploration of optimum cache configurations in NoC-based many-core CMPs while addressing the exploration space complexity by a machine learning based static profiling technique.

2.3 Eavesdropping Attacks

As discussed in Chapter 1, IPs integrated on the same SoC use the NoC to communicate between each other using message passing as well as shared memory. Therefore, eavesdropping on the NoC allows an attacker to extract secret information without relying on memory access (either through on-chip cache or off-chip memory) or hacking into individual IPs. Bus based communication (e.g., broadcast in wireless NoCs) is inherently vulnerable to eavesdropping

attacks. Existing literature on NoC security has explored several variations of the eavesdropping attack.

One commonly explored threat model is where the malicious NoC IP colludes with an accompanying malicious application running on an another IP to launch an eavesdropping attack. I have described this threat model in detail in Chapter 10. It includes a Trojan-infected router copying packets passing through it and sending the duplicated packets to another IP running a malicious application in an attempt to steal confidential information. This threat model has been extensively used to study eavesdropping attacks specially since the attack is hard to detect [46, 88–91]. Trojans can also directly eavesdrop on the NoC communication without relying on re-routing duplicated packets to an accomplice application. This can be facilitated by external I/O pins attached to the NoC [92]. However, NoCs are generally more resistant against bus-probing attacks compared to the traditional bus-based architectures.

Similar to the malicious router and application colluding to launch the attack, a Trojan infected network interface and an application can work together to launch an eavesdropping attack [49]. In the threat model presented in [49], the hardware Trojan embedded in the NI can tamper with the flits in the circular flit queue, which is used to store flits before sending them to the corresponding router. When a flit is sent to the router, it waits in the queue until the next flit overwrites it. The Trojan keeps track of such outstanding flits, modifies the header flit with a new destination address and updates the header pointer so that it gets re-sent to the router. The duplicated flits are received by the malicious application. The area overhead of the Trojan is shown to be 1.3% [49].

Common countermeasures against eavesdropping attacks include packet encryption, authentication, additional validation checks during NoC traversal and information obfuscation. Encryption ensures that the plaintext of the secure information is not leaked and authentication detects any tampering with the packet including header information []. Several prior studies have tried to develop lightweight encryption and authentication schemes for on-chip data communication. Ancajas et al. [46] proposed a simple XoR cipher together with a packet

certification technique that calculates a tag and validates at the receiver. A configurable packet validation and authentication scheme was proposed by merging two robust error detection schemes, namely algebraic manipulation detection and cyclic redundancy check, in [91]. Intel's TinyCrypt - a cryptographic library with a small footprint is built for constrained IoT devices [93]. It provides basic functionality to build a secure system with very little overhead. It gives SHA-256 hash functions, message authentication, a psuedo-random number generator which can run using minimal memory, digital signatures, and encryption. It also has the basic cryptographic building blocks such as entropy sources, key exchange and the ability to create nonces and challenges. I have discussed encryption and authentication in detail and proposed new defense mechanisms in Chapters 5, 10 and 6. The duplicated packets in router-application combination as well as NI-application combination can be detected by additional validation checks. In [49], the authors implemented a snooping invalidator module (SIM) at the NI output queue to discard duplicate packets. On the other hand, information obfuscation can make the attack harder to initiate. For example, hiding the source and destination information of NoC packets can ensure that the malicious agents in the NoC are unable to select the target application to eavesdrop. Onion routing, a well known mechanism in the computer networks domain, can hide the origin and target of a network packet [94]. However, implementing such complex security mechanisms is not feasible in resource-constrained SoCs. Several previous studies tried to propose lightweight solutions that are compatible with the NoC context [46, 95].

2.4 Spoofing and Data Integrity Attacks

Spoofing and data integrity attacks intentially corrupt data transferred on the NoC to cause malfunction. Sepúlveda et al. presented "MalNoC", a Trojan infected NoC that can perform multiple attacks on NoC packets [90]. The infected MalNoC router copies packets arriving at a router, replaces the packet data with the content in a malicious register, modifies source and/or destination address in the header to the desired IP, and injects it back into the NoC. A control register within the router controls the Trojan operation. A similar threat model

that discussed eavesdropping, DoS and illegal packet forwarding, all of which utilized packet corruption at a router was presented in Section [88]. Kumar et al. [89] discussed a Trojan that corrupts flits arriving at the input buffers of a router.

Trojans can also be inserted in links to corrupt NoC packets. To avoid being detected, the Trojans change only the header flits causing deadlock, livelock and packet loss situations [96]. Even if hardware Trojans are not present, bit flipping can happen when packets are transferred through the links due to other reasons. Error correction codes are used to correct such bit flips. The Trojan in the link attempts to mask its malicious behavior as an error rather than a security attack to avoid being detected. The authors have explored the impact of Trojans embedded in different links (boundary links versus center links) in a 5×5 Mesh NoC [96].

Authenticated encryption schemes provide data confidentiality through encryption and data integrity through authentication [90, 97, 98]. If the authentication tag is calculated using the entire packet (header as well as payload), any packet corruption can be detected at the receiver's side when the packet is validated using authentication. Hussain et al. [88] argued that since the Trojan is rarely activated to avoid detection, authenticating each packet can lead to reduction in energy efficiency. In their work, they proposed an efficient Trojan detection design where the authentication gets activated only when the hardware Trojan has been triggered in the system. A combination of security modules placed at the IPs as well as at the routers provided attack detection as well as Trojan localization capabilities [88].

Error correcting codes (ECC) are widely used in the telecommunications domain [99]. ECCs have been used in NoCs to correct bit errors due to particle strikes, crosstalk and spurious voltage fluctuation in NoCs. Yu et al. introduced a method to detect Trojan induced errors using ECCs in [96]. Their method consisted of two main components. i) Link reshuffling: to avoid the Trojan from affecting the same bit in an attempt to create deadlocks/livelocks, the odd and even bits are switched in the retransmitted flit in case of an error detected by the ECC. This is effective for scenarios where the Trojan is triggered by specific flits. If the Trojan gets activated by a certain input, reshuffling the bits during the retransmission can make the

Trojan inactive again. ii) Link isolation: an algorithm to isolate links that are suspected to have Trojans. Trojans that are triggered by external signals can remain active for a long time. In such cases, wire isolation is used to reduce the number of retransmissions.

2.5 Denial of Service Attacks

Several threat models related to DoS attacks have been studied in prior work. One common threat model is where malicious IPs manipulating the availability of on-chip resources by flooding the NoC with packets. The performance of an SoC can heavily depend on few components. For example, a memory intensive application is likely to send many requests to memory controllers, and as a result, routers connected to them will experience heavy traffic. If a malicious IP targets the same node, the SoC performance will suffer significant degradation [85, 100–102]. This is known as a flooding-type DoS attack. We discuss flooding type of DoS attacks in detail in Chapters 7 and 11.

Continuous corruption of packets can also lead to a DoS attack [89, 103]. In [89], hardware Trojans tamper flits arriving at the input buffer of a router causing performance degradation. Performance degradation is caused by dropped packets, wastage of NoC resources such as buffer space, response delays and retransmissions. Boraten et al. [104] discussed a similar threat model where hardware Trojans influenced resource allocations and corrupted data to degrade performance. The same authors further explored possible DoS attacks in [105]. Compared to router-based packet corruption, they discussed a Trojan that performs deep packet inspection on links and inject faults when the target is identified. The injected faults trigger re-transmissions from the error correcting mechanism. Therefore, repeated injection of faults causes repeated re-transmission to starve network resources and create deadlocks capable of rendering single application to full chip failures.

Rajesh et al. [106] discussed a threat model where the packets are unfairly treated at the router to cause a DoS attack. The malicious NoC IP, once integrated on the SoC, picks a victim IP that is an important SoC component and manipulates the traffic flow to/from the victim IP. The traffic flow is manipulated by denying fair access to the allocator and arbiter

units in the router. The allocator is responsible for granting flits access to the crossbar. DoS is achieved by the allocator delaying packets to/from the victim IP. At the arbiter, the Trojan infected router gives least priority to the flits that have the victim IP as the source/destination. Both these scenarios lead to flits to/from one IP getting significantly delayed.

To address these different threat models, researchers proposed several solutions. As a countermeasure to denial-of-service through packet corruption, Kumar et al. proposed a bit shuffling method that makes flits less sensitive to the attack [89]. The authors proposed to shuffle the critical bit fields of the flits among themselves and others so that the Trojan is attacking on randomly shuffled data and not on the critical fields within the packets such as flit indication bits, source and destination addresses. While fuzzing can make the attack difficult, it does not guarantee prevention. Furthermore, the attack is not detected, and as a result, future attacks are not prevented either. Boraten et al.'s work was motivated by this, where they coupled switch-to-switch scrambling, inverting, shuffling and flit reordering with a heuristic-based fault detection model [105]. Their solution addresses the challenge of differentiating fault injections from transient and permanent faults. Another technique that exhibits similar defense characteristics as fuzzing - partitioning, tries to reduce interference of communication between different applications/packet types. As a result, overwhelming the NoC with DoS attacks becomes difficult [107].

Monitoring the traffic flow to detect abnormalities is another common defense against DoS attacks. Rajesh et al. [106] proposed a defense against their traffic flow manipulation threat model that is based on identifying the latency elongation of packets caused by the DoS attack. Their method relied on injecting additional packets to the network and observing their latencies. SoC firmware then examines the latencies of the injected packets. If two packets are injected at the same time and traverse paths with significant overlap, they are expected to exhibit comparable latencies. If not, it will be flagged as a potential threat. Similar methods that profiled normal behavior of traffic during design time and monitored NoC traffic to detect deviations from normal behavior were proposed in [101, 108]. Exploring another orthogonal

direction, work in [102–104, 109] proposed additional formal verification and runtime checks integrated in to the NoC to prevent and detect DoS attacks.

2.6 Buffer Overflow and Memory Extraction Attacks

Similar to the buffer overflow attacks in the computer networks domain, execution of malicious code can launch a buffer overflow attack in NoC-based SoCs. If a malicious IP writes on the stack and modifies the return address of a function to point at the malicious code, the malicious code will be executed. Return address modification in the stack is done by writing more data to a buffer located on the stack than what is actually allocated for that buffer. This is known as "smashing the stack" [110]. Even if the stack memory is made non-executable, or kept separate, it is possible to overwrite both the return address as well as the saved registers. Work done in [55] explored this threat model. Buffer overflow attacks pose a significant threat in NoC-based SoCs where the memory is shared among multiple cores.

Kapoor et al. in their work considered some IPs on the SoC to contain confidential information (secure/trusted IP cores) and some untrusted IPs which can potentially carry hardware Trojans (non-secure/untrusted IP cores) [97]. The information inside secure IP cores should be protected from non-secure IP cores. Since all IPs are integrated on the same NoC, non-secure cores can communicate with secure cores. Non-secure cores can try to install Trojans in the secure cores and try to extract information. The confidential information in registers in the secure cores such as cryptographic keys, configuration register information and other secure data can be compromised in such an attack [97]. This threat model of non-secure IP cores trying to access secure-IP cores has been used in several other work as well [98, 101, 111–113].

Lukovic et al. proposed two methods to counter buffer overflow attacks. The first method focused on protecting the processing cores by embedding additional security in the network interface (NI) [55]. In their work, a data protection unit, which is similar to a firewall sits on the NI attached to the shared memory block. It secures the memory by filtering unauthorized memory access requests. A stack protection unit (SPU) is developed which protects the

stack from attacks that targets the return addresses. The SPU is developed as a part of the processor protection system which combines software and hardware units that replicate return addresses stored in the stack and protects it against code injection attacks. These countermeasures also stopped the attack from getting propagated to other parts of the NoC. Their second method extends the solutions proposed in [55] to a hierarchical security architecture [114]. The authors introduced four levels of security working at system level, NoC cluster level, per core, and in a layer specific to the attack (e.g., code injection). Similar to software protection mechanisms and the data protection unit in [55], many existing work provide access control by monitoring the incoming request [101, 111, 112]. For example, Saeed et al. introduced a method to mitigate buffer overflow attacks in an NoC based shared memory architecture by deploying an ID and address verification unit (IAV) [112]. This minimizes the threats caused by malicious IPs in the NoC because the IAV verifies each incoming packet by its ID and address.

Adding an extra layer of security to access authorization, commercial products such as Sonic SMART Interconnect [39] and ARM TrustZone [115] divide memory blocks into different protection regions and isolate secure and normal execution environments from each other. If the non-secure cores access secure cores, requests are validated by access authorization techniques [97, 98]. It is possible that security zones have to be modified due to task migration, new applications starting and ending. Therefore, security zones have to be created, modified and eliminated during runtime. Sepúlveda et al. [116] achieved this by using a partitioning method that used a lightweight Diffie-Hellman key-exchange protocol. The same authors proposed a method to create dynamic firewalls at the network interface to monitor and filter the NoC traffic [117]. The dynamic firewalls create "elastic security zones" by wrapping a desired set of components in a 3D NoC according to a trust policy. Porquet et al. [118] presented a method to co-host several secure applications running in parallel using the same shared memory space. Secure hardware implemented at the NI of the NoC enables secure and flexible partitioning of the shared memory space between multiple applications. Their approach

is similar to the operation of a virtualization hypervisor that protects code, data, exclusive peripheral device usage, etc., when multiple virtual machines are running on the same host machine [119].

2.7 Side Channel Attacks

Due to the difference in computation requirements, secure systems often take different times to perform different operations. By carefully measuring these time differences, it is possible to extract secret information from vulnerable systems. Reinbrecht et al. demonstrated a practical "Prime+Probe" timing attack on an NoC based SoC [120]. The target of their attack was the communication between an ARM Cortex-A9 core and a shared cache memory. Other studies carried out on timing attacks also used similar concepts on timing analysis of network traffic for attacks [121–124]. The threat model in [121] included four cores. Two of which are carrying out a secure communication and the other two, which lies on the secure communication path will be infected by the adversary. The two infected cores inject traffic to the network. Adversary is then able to observe latencies of maliciously injected traffic to infer information about timing, frequency and volume of the secure communication.

Wang et al. [124] in their work showed that the number of ones in the RSA [125] key can be inferred with a timing side channel attack on NoC, which can then be used to infer the entire key. A major part of the RSA algorithm is to do the modulo multiplication of two large (1024 or 2048-bit) numbers. The modulo multiplication is shown to be vulnerable to timing side channel attacks [126], mainly because the algorithm examines each bit in the RSA key and multiplies only if it is one. Wang et al's attack is based on observing the additional network traffic caused due to multiplications [124]. Similar to recovering the RSA key through timing attacks, existing work used the AES cipher as case studies as well. In 2010, Bogdanov et al. [127] proposed a differential cache collision attack on embedded systems. While their work did not consider an NoC based setup, in 2018, Reinbrecht et al. [128] showed that combining their previous work on NoC timing attacks [123] with Bogdanov et al's cache collision attack [127] can significantly enhance the AES key recovery effort.

Measuring the power consumption will give information about the process that is occurring inside the system. For example, if the processor is performing a simple addition versus executing an encryption instruction (Intel chips come with "AESENC" instruction that performs one round of AES encryption on a given plaintext), observing their power consumption can give reasonable information to differentiate the two operations. Similarly, many data encryption standard (DES) implementations have visible differences within permutations and shifts which can be utilized to break the security scheme [129]. Differential power analysis is a powerful attack technique based not only on power observations, but also on statistical analysis and noise filtering methods to gain more information about the underlying security scheme [130].

In addition to timing and power, existing work has explored thermal side channels. Similar to power, the SoC thermal characteristics are highly correlated to the SoC operation. Guo et al. [131] discussed two main thermal characteristics:

1. Spatial distribution: by observing the heatmap, attackers can identify active cores in the SoC.
2. Temporal variation: different instructions have different thermal profiles when executed. The temperature trace over time allows attackers to infer the executed instructions with a certain probability.

As a countermeasure to the "Prime+Probe" attack, the authors proposed "Gossip NoC" [120, 123] - a two stage security mechanism which first detects the attack and then protects the SoC. The detection process monitors the bandwidth and sends an alert message in case of a potential security breach. The protection mechanism gets triggered by this alert message which then alters the routing protocols to route packets avoiding the sensitive path that contains the malicious IP. The same route randomization concept was used as a mitigation technique in [121, 122]. Sepúlveda combined random arbitration with adaptive routing to dynamically allocate NoC resources, and as a result, minimized interference between secure packets and packets injected by the attacker [132].

As a solution to the thermal side channel attacks discussed in [131], the authors presented a task mapping scheme that minimized the thermal information leakage. In their work, a mathematical model was developed to quantify the security cost corresponding to a certain application mapping. A greedy optimization algorithm was then used to map application threads to cores such that the leakage is minimized. The optimization algorithm is implemented in the operating system and it receives SoC status from a hardware monitor. The security cost is then calculated according to the model for each core and a new application mapping is generated if required.

To avoid timing side channel attacks similar to the one introduced in [124], the same authors proposed to partition network traffic based on its security level. The basic idea is to make sure packets from applications running on secure IPs do not interfere with the packets from applications running on non-secure IPs. As a result, the communication latency and throughput of non-secure applications become independent of the dynamic behavior of secure application traffic. An obvious way to achieve this goal is to statically partition NoC resources (link bandwidth, buffers, etc.) spatially or temporally. However, it can lead to sub-optimal results causing performance degradation. Wang et al. introduce a priority-based arbitration technique for resources such as the router crossbar along with static allocation of virtual channels [124]. A similar principal was used in the "Secure Enhanced Router" architecture proposed by Sepúlveda et al. [133]. In their work, the router architecture included a shared buffer space and the number of virtual channels per input port was decided during runtime according to communication and security requirements. Similar to [124], the goal was to make the non-secure traffic flow oblivious of the secure traffic flow.

2.8 Summary

In this chapter, I have surveyed security vulnerabilities and defenses in NoC based SoCs. I have considered existing literature covering state-of-the-art attacks and defense mechanisms. The literature contains a significant amount of work related to on-chip network security. In

particular, I have discussed the research efforts under five classes of attacks highlighting their threat models and respective countermeasures.

CHAPTER 3
ACCURATE MODELING OF NOC ARCHITECTURES

The network-on-chip (NoC) performance and power consumption depend critically on the traffic load. The network traffic itself is a function of not only the application that causes packet injection on the NoC, but also the cache coherence protocol, and memory controller/directory locations. As discussed in Chapter 1, security has to be considered in the context of other non-functional requirements such as performance, power and area. To accurately measure the impact of any security mechanism or optimization technique, it is important to capture the traffic behavior accurately, and model them in architecture simulators. When modeling NoC behavior in architecture simulators, the following requirements should be satisfied.

- Availability of a NoC model that accurately captures NoC traffic behavior and corresponding performance, power and area statistics.
- Adequate visibility to NoC packets and NoC components to debug and measure the efficacy of security countermeasures at different abstraction levels.

In this chapter, I present an overview of how NoC has been utilized in commercial system-on-chips (SoC) together with their optimization techniques. I show that using unrealistic models in a widely used multiprocessor simulator produce misleading power and performance predictions. I introduce accurate models to capture the traffic behavior, which are comparable with results from hardware platforms.

The rest of the chapter is organized as follows. Section 3.1 provides a background on related concepts. Section 3.2 introduces Intel Knights Landing architecture including the memory and cluster modes. Section 3.3 presents the NoC modeling and exploration framework. Section 3.4 presents the experimental results. Finally, Section 3.5 summarizes the chapter.

3.1 Background

To facilitate memory transactions, modern chip multiprocessor (CMP) architectures comprise of a low-latency NoC that interconnects the cores with each other and a suite of integrated memory controllers (MC), which provide interfaces to multi-channel DRAM

(MCDRAM) and main memory (DDR) [17, 134, 135]. The memory hierarchy of CMPs commonly employ directory-based cache coherence protocols and multiple levels of cache. The first and second level caches (L1 and L2) are collocated with each core, while the last level cache (LLC) and tag directory are distributed throughout the chip, as illustrated in Figure 3-1. An L2 miss triggers a request to the directory that keeps track of the corresponding memory address. In case of a hit, the data is returned from (or written to) the LLC slice collocated with the directory. Otherwise, the request is forwarded to one of the MCs. Due to pin limitations and packaging constraints, the number of MCs is much less than the cores. For example, Intel Xeon Phi has 8 MCs interfacing MCDRAM and two MCs interfacing DRAM, while the system has 72 cores [4]. Similarly, AMD Opteron 6386 SE has 16 cores with 1 MC and 4 memory channels. Therefore, LLC-Memory communication exhibits a many-to-few communication pattern while Core-LLC communication is many-to-many. In other words, memory traffic is likely to introduce "hotspots", whereas Core-LLC traffic is relatively uniform. As a result, these hotspots and poor design choices can cause significant performance degradation [60].

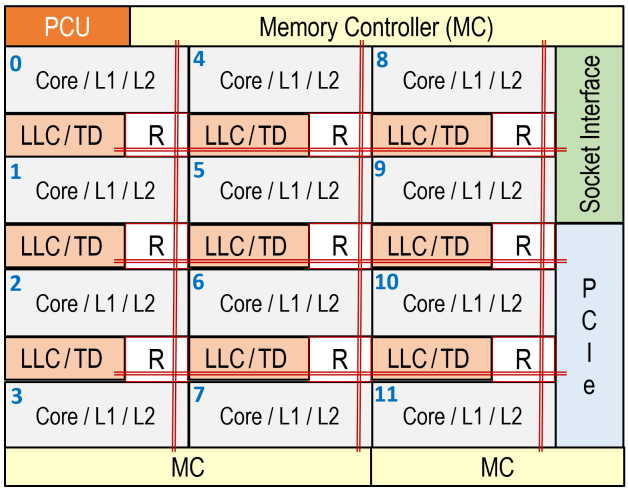


Figure 3-1. Representative illustration of a many-core CMP.

As an example, Intel Xeon Phi processor provides different cluster modes that define the affinity of directories to specific MCs due to the importance of LLC/directory to memory

traffic. In addition, the MCDRAM can be used as a cache, an extension to DDR or in a hybrid setup, giving three memory mode options. Each of these choices lead to a different NoC traffic pattern as a function of the workload. Analyzing the power consumption and performance impact of cluster and memory modes is important for two reasons. First, it enables us to use the existing platforms optimally. Second, it can help in making better architectural choices. This analysis is not feasible on existing hardware platforms, since the traffic between the cores and memory is not observable. Furthermore, there are no public simulators capable of performing this exploration. For example, gem5 [18], which is one of the most widely used architecture simulators, assumes that there is an interface from each tile to the main memory. Consequently, the memory access latency is modeled, but the actual traffic from LLC/directory to memory is not captured. This makes the default gem5 model unsuitable for cluster and memory mode exploration.

We propose a methodology for analyzing the impact of cluster and memory mode choices on the NoC traffic. We demonstrate that congestion on the NoC links affects not only the communication latency, but also power consumption and application execution time. We also show that any exploration that involves LLC/directory to memory traffic requires a simulation framework that models the cache coherence protocols accurately. We demonstrate both qualitatively and quantitatively that neglecting the LLC/memory traffic, as it is done in gem5, gives highly optimistic results in terms of the network load, latency and power consumption. We also show that this inaccuracy can lead to misleading conclusions in terms of optimal MC placement. Then, we describe how the LLC/directory to memory traffic, originating from directory-based cache coherence, can be modeled in architectural simulators. Using the corrected gem5 model, we evaluate the power consumption and performance impact of several cluster modes and memory modes which configure the directory-MC affinity. Our modified gem5 architecture enables the exploration of NoC optimization and security countermeasures proposed in subsequent chapters.

3.2 Memory and Cluster Modes in Modern CMPs

Knights Landing (KNL) is the codename for the second generation Xeon-Phi processor introduced by Intel which targets highly parallel workloads [4]. An overview of the KNL architecture is shown in Figure 3-2. It has 36 tiles arranged in a Mesh interconnect. It supports two types of memory - (i) multi-channel DRAM (MCDRAM) which is a 16 gigabyte high-bandwidth memory, and (ii) double data rate (DDR) memory which has a capacity of 384 gigabytes with less bandwidth. The architecture gives the option of configuring these two memories in several configurations which are called memory modes.

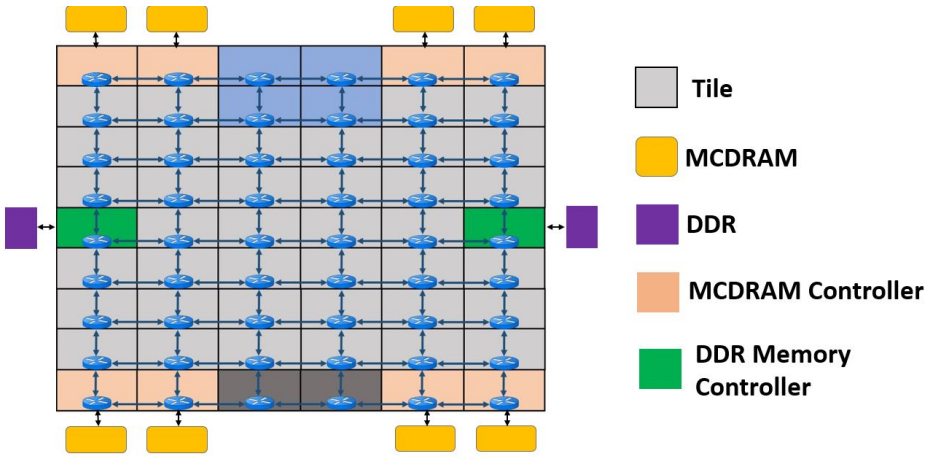


Figure 3-2. Overview of the KNL architecture.

3.2.1 Memory Modes in Xeon-Phi Architecture

Xeon-Phi architectures have a high-bandwidth MCDRAM memory and a larger low-bandwidth DDR memory [4]. These two memory types - MCDRAM and DDR memory, can be configured at boot time from the BIOS in different ways, as illustrated in Figure 3-3.

- **Flat Mode:** In the flat mode, both the MCDRAM and DDR memory are mapped in the same system address space. This mode is ideal for applications with data that can be separated into categories of a larger, low-bandwidth region, and a smaller, high-bandwidth region.
- **Cache Mode:** In the cache mode, MCDRAM acts as a last level cache which is placed in between the DDR memory and L2 cache. The cache is direct mapped with a cache line size of 64-bytes. All memory requests first go to the MCDRAM for a cache memory lookup, if there is a cache miss, they are sent to the DDR memory.

- **Hybrid Mode:** In the hybrid mode, part of MCDRAM (half or quarter) is used in cache mode while the rest is used as flat mode memory. The DDR memory will be served by the cache portion. This works well for a variety of applications that take advantage of storing frequently accessed data in flat memory while also benefiting from regular caching.

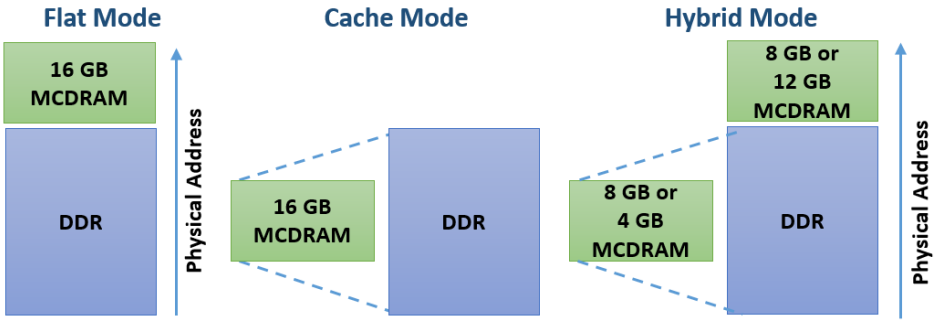


Figure 3-3. Three memory modes in Xeon-Phi architectures.

3.2.2 Cluster Modes in Xeon-Phi Architecture

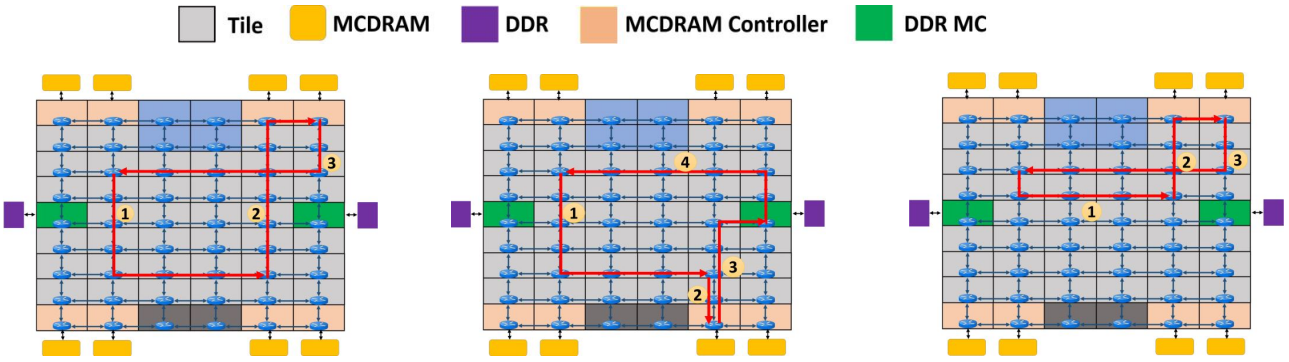
The mesh interconnect in KNL supports three cluster modes, which have significant impact on the NoC traffic behavior [4]. Similar to memory modes, cluster modes can also be selected from BIOS during boot time.

- **All-to-all mode:** In this mode, there is no affinity between the PE, MC and directory. That is, a memory request can go from any directory to any MC. As a result, this mode does not exploit locality, unlike the other two cluster modes.
- **Quadrant mode:** In the quadrant mode, the chip is divided into four quadrants. There is an affinity between the directories and MC in the same quadrant. However, there is no affinity between the processing element (PE) and directory, i.e, a processor can send the memory request to any directory, but the directory will always forward that request to an MC on the same quadrant.
- **Sub-NUMA mode:** This mode takes one more step forward by enforcing affinity between all three components - PE, MC and directory. A request from a PE lands on a directory on the same quadrant, and the directory can forward that request to an MC on the same quadrant.

The optimal combination of memory and cluster modes depends on the application characteristics and, largely affects the power and performance statistics. Applications whose threads and memory footprint fit to a single quadrant can take advantage of the strong locality

of quadrant and sub-NUMA modes. However, highly parallel applications with a large number of threads and memory footprint may benefit from all-to-all and flat memory modes.

Figure 3-4 illustrates the traffic flow of these memory and cluster modes using examples. The quadrant and sub-NUMA clustering modes improve the locality of memory traffic. For instance, Figure 3-4D illustrates the quadrant mode in KNL [4]. The initial request from a core can go to any directory (1). However, each directory is associated with the MCs within the same quadrant. The memory request marked with (2) can go to integrated MC on the right side or to MCDRAM controllers at the upper right corner. This affinity helps in localizing the directory-memory traffic, which in turn improves memory access latency.



B Example of L2 miss in flat memory mode and all-to-all cluster mode: (1) L2 cache miss. Memory request injected on the network to check the tag directory, (2) request forwarded to any memory controller after miss in tag directory, (3) data read from memory and sent to the requester.

C Example of L2 and MCDRAM miss in cache memory mode and all-to-all cluster mode: (1) L2 cache miss. Memory request injected on the network to check the tag directory, (2) request forwarded to MCDRAM which acts as a cache after miss in tag directory, (3) request forwarded to memory after miss in MCDRAM, (4) data read from memory and sent to the requester.

D Example of L2 miss in flat memory mode and quadrant cluster mode: (1) L2 cache miss. Memory request injected on the network to check the tag directory, (2) request forwarded to memory controller on the same quadrant, (3) data read from memory and sent to the requester.

Figure 3-4. Traffic models in flat and cache memory modes and all-to-all and quadrant cluster modes in KNL architecture

3.3 Accurate Modeling of LLC/Directory to Memory Communication

In this section, we first describe how the transactions between core, LLC/directory and memory occur in modern CMPs. Then, we contrast it to the assumption made by gem5 and

highlight the consequences. Next, we present an accurate NoC modeling and implementation of cluster and memory modes in gem5. Finally, we demonstrate that our framework is vital to accurately model and explore modern CMPs.

3.3.1 Memory Controller Placement in CMPs

Due to pin limitations and package constraints, it is unrealistic to attach a memory controller to each core in a CMP. For example, Intel Core i7-900 processor has only one MC, and 27.3% of its total pins are dedicated to the MC [136]. Similarly, the Tilera Tile64 processor integrates 64 cores in an 8x8 mesh with four on-chip MCs [17]. This results in a core to MC ratio of 16:1. The total number of cores and memory controllers in several modern CMPs are summarized in Table 3-1.

Table 3-1. Comparison of cores and number of MCs in modern many-core CMPs.

Processor	No. of Cores	No. of Memory Controllers
Intel Xeon Phi 7210 [137]	64	8 MCDRAM & 2 DDR4
Tilera Tile64 [17]	64	4 DDR2 in 16 ports
Intel Xeon 8160M	24	2 DDR4, 6 channels
AMD Opteron 6386 SE	16	1 DDR3, 4 channels

Several studies have shown that relative placement of cores and MCs plays an important role in network traffic distribution [60, 61]. This impact is more significant in topologies, such as 2D Mesh, which do not have edge symmetry. Thus, it is evident that connecting MCs to every tile gives a highly optimistic estimate of the realistic scenario. Moreover, a large fraction of traffic in a CMP originates not from actual data transfers, but from communication between cores to maintain data coherence [68]. As soon as the directory component comes into play, the traffic distribution is not the same as processor to processor traffic or processor to memory traffic. Therefore, the affinity between the cores, directories and MCs affect the performance of architectures that employ a distributed directory-based cache coherence algorithm. Consequently, it is crucial to accurately account for the communication flow between the cores, tag directories and memory controllers.

Arguably, the most widely used architectural simulator - gem5 [18] makes an unrealistic assumption that there is an interface to main memory from every tile of the NoC. This eliminates the exploration of affinity between directory and MC. Furthermore, the effects of memory modes cannot be captured in the current gem5 setup.

3.3.2 LLC/Directory to Memory Communication in CMPs

A miss in the local cache triggers a sequence of transactions in many-core architectures with distributed directories, as demonstrated Figure 3-5a.

The order of these transactions are as follows:

1. The request is forwarded to the directory controller which contains the memory address information,
2. If data is not available in any of the caches, the request is forwarded to an MC,
3. The data is retrieved from the memory,
4. The MC forwards the data to the requester.

The last two steps are significant, since they introduce many-to-few communication pattern due to the smaller number of MCs, as summarized in Table 3-1. As a result, they not only increase the number of packets in flight, but also lead to hotspots which contribute to increased latency.

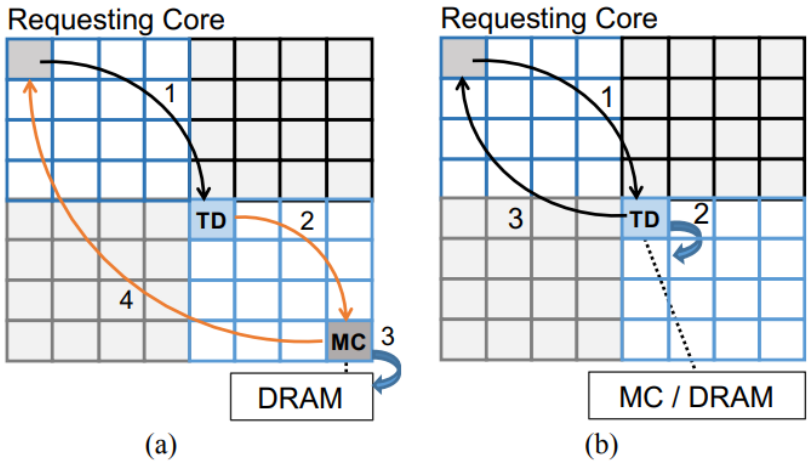


Figure 3-5. Life cycle of a memory request and resulting transactions in real distributed directory systems and in the gem5 simulator.

3.3.3 Unrealistic Assumptions in gem5 Traffic Model

gem5 is one of the most popular many-core architecture simulators [18]. Instead of following these steps given in Section 3.3.2, it models the memory accesses directly from the directory (home node) itself, as illustrated in Figure 3-5b. The first step is the same as shown in Figure 3-5a. That is, the request goes from the core to the tag directory responsible for the corresponding memory address. If there is an LLC miss, the data needs to be fetched from the memory, as expected. However, the memory access is modeled within the home directory (2), without explicitly modeling the traffic from the directory to memory controller. In other words, each "directory controller" implies both a directory (i.e. state) and an MC [18]. The model accounts for the delay to main memory, but it does not have a separate MC node in the NoC. Therefore, the NoC traffic to and from the memory controllers is not modeled. In contrast, the data is forwarded directly from the directory to the requester (3). Comparing the two scenarios, we can see that step 2 in Figure 3-5a does not exist in the current gem5 model. Moreover, the data (step 3) is sent from the directory, not from the MC as in the realistic model.

Impact on NoC Traffic: The modeling choice in Figure 3-5b essentially establishes a virtual link between the tag directory and memory controllers. Therefore, the request and data packets to and from MCs are completely missed in this simulation model. This affects not only the communication latency of a given transaction but also the utilization of the links and routers on the path. Consequently, the latency of "all the NoC traffic" that goes through those routers will be lower in simulation than their actual values. Hence, this will result in optimistic performance estimates.

3.3.4 Modeling and Exploration of Intel Xeon-Phi Architecture

An accurate NoC simulation model should explicitly capture PE to directory, directory to memory and memory to PE traffic.

Mapping Addresses to Memory Controllers: Since all the cores share the MCs, we need a mechanism to allocate different address ranges to the available MCs. To achieve this,

the physical address of a memory location is mapped to an MC according to the function shown in Listing 3.1. It allocates a certain set of bits from the address to select the MC by defining the range of bits from "small" to "big" and dividing the addresses uniformly among MCs. In this formulation, *addr* is the address to map, *small* is calculated as (*numa_high_bit* - *num_memories_bits* + *1*), and *big* is *numa_high_bit*. Here, "numa_high_bit" and "num_memories_bits" are calculated depending on the number of MCs. These expressions enable an even distribution of memory addresses among the MCs, which is similar to the decisions in a modern CMP [138].

Listing 3.1. Address hashing function used to map an address to a memory controller

```
Addr bitSelect(Addr addr, unsigned int small, unsigned int big){


    assert(big >= small);
    if(big >= ADDRESS_WIDTH-1){
        return (addr >> small);
    } else{
        Addr mask = ~((Addr)~0 << (big+1));
        Addr partial = (addr & mask);
        return (partial >> small);
    }
}
```

Simulation Framework: We employ a cycle-accurate full-system simulator - gem5 [18] and "GARNET2.0" [139] interconnection network model. The default gem5 model is modified to include separate MCs and to model PE to PE, PE to directory, directory to memory as well as memory to PE traffic. The gem5 implementation handles the traffic flow through coherence protocols. In a distributed cache coherence protocol, in case of a cache miss, the request is forwarded to the coherence protocol controller. It makes the necessary state transitions and pushes the message in the appropriate virtual network to the network interface. The network interface then converts the message into network packets and sends them to the network via

the connected router. The network then routes the flits to the destination node using X-Y deterministic routing protocol. When the home directory receives the packet, it checks its state machine to see if another cache shares that data. If yes, it forwards the packet to the owner and then to the requestor (PE) and if not, it initiates a memory fetch depending on which memory and cluster modes are being used.

If it is all-to-all and flat mode, addresses are uniformly distributed across MCDRAM and DDR memory spaces. Which MCDRAM/DDR memory controller to forward to is decided using the function in Listing 3.1. If it is quadrant and flat mode, only MCs in that quadrant are considered as candidates for forwarding the memory requests. In all-to-all and cache mode, MCDRAM space is treated as a last-level cache. Therefore, the request is sent to an MCDRAM controller for a cache lookup. If it is a miss, the memory request is again forwarded to the appropriate MC (selected using Listing 3.1 without considering MCDRAM controllers), and memory fetch request is placed through there. Once the requested data is fetched from either the DDR or MCDRAM memories, it is forwarded back to the PE after making the necessary coherence transitions.

We explicitly differentiate the behavior of MCDRAM memory in cache and flat modes. In cache mode, MCDRAM cache modules are instantiated and can be accessed only through the designated MCDRAM controller locations. In flat mode, this cache module is not used, and an MC similar to the MCs interfacing DDR memory is connected to the designated nodes.

We emphasize that without our modification of the gem5 model, it is not possible to explore the power consumption and performance impact of different cluster and memory modes. The next section highlights two important aspects of our exploration framework. Our proposed NoC model is realistic since the power and performance numbers are comparable with the results from the Xeon-Phi hardware board. Moreover, our framework can be used to accurately model and explore a wide variety of current and future NoC architectures.

3.4 Experiments

3.4.1 Experimental Setup

Architecture Model: In our studies, we use the Intel Xeon Phi 7210 platform [137] and model the same on gem5 [18]. It mainly targets high performance computing and other parallel computing segments. The architecture offers high memory bandwidth and massive parallelism options which enables it to run memory and processor intensive workloads with high throughput.

A 64-core CMP is modeled with gem5 using a mesh topology. Each tile is composed of a core that runs at 2 GHz, private L1 cache, tag directory and a router. Each cache is split into data and instruction caches with 16kB capacity each. GARNET2.0 [139], which leverages the routing infrastructure provided by ruby memory system, models a router with a crossbar switch, switch allocation, virtual circuit selection and 4 input buffers giving a 3-cycle pipeline. Each router is connected to four other routers with internal links and to an L1 cache and a directory controller through individual network interfaces via external links. The complete set of simulation parameters are summarized in Table 3-2.

NoC Power Model: Since dynamic power consumption of an NoC is a function of the traffic flow, we need to use an energy model that captures the changes in the traffic flow. We use the model in [140] to estimate the power consumption. According to the energy model, there are two main contributors to NoC power;

1. Number of packets injected into the network - this is directly related to the number of cache misses in L1 and L2 caches, and in cache memory mode, misses in MCDRAMs.
2. Average hops traversed by packets - depends on the relative placement of PE, MCs and directories. The affinity between these components which are configured using the cluster modes also contributes to the number of hops.

We feed the output statistics from gem5 to the McPAT power modeling framework [141]. Power consumption of other components - caches, processor, off-chip memory and directories, are estimated using the energy models in McPAT.

Benchmarks: We use benchmarks from SPLASH-2 [142] and MiBench [143] benchmark suites to run on gem5.

3.4.2 Parameters Used to Model KNL

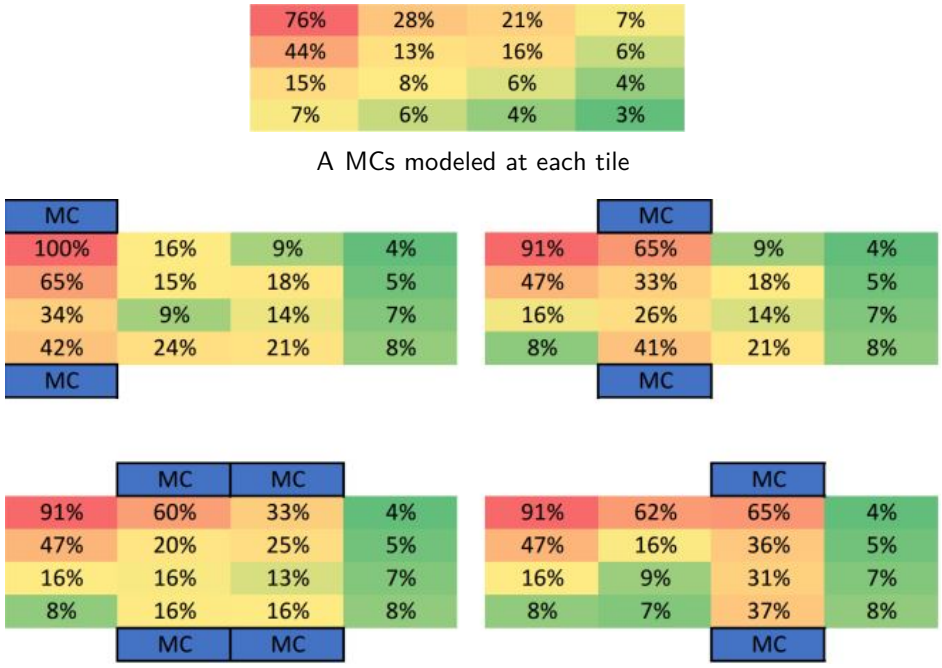
The number of cores in gem5 must be a power of 2. We have 32 tiles with cores similar to KNL. However, each tile contains a single core unlike KNL, since gem5 does not support tiles with two cores. To match the number of cores, we deactivate one core in each tile in our Xeon-Phi platform. We also place the MCs to match the KNL architecture shown in Figure 3-4. Moreover, we set the core frequency to 1.4 GHz when comparing the simulation results against the hardware measurements to match our Xeon-Phi platform frequency. The parameters used in implementing KNL are summarized in Table 3-2.

Table 3-2. System configuration parameters used in our simulations.

Parameter Class	Parameter	Value
Processor Configuration	Number of cores	64
	Core frequency	2 GHz
	Instruction set architecture	x86
Memory System Configuration	L1 cache	private, separate instruction and data cache. Each 16kB in size.
	Cache coherence	distributed directory-based protocol
	Memory size	4GB DDR
	Access latency	300 cycles
Interconnection Network Configuration	Topology	8x8 Mesh (formed by rings in rows and columns)
	Routing scheme	X-Y deterministic
	Router	4 port, 4 input buffer router with 3 cycle pipeline delay
	Link latency	1 cycle
Parameters that change when implementing KNL	Number of cores	32 (in 32 tiles each with one core)
	Core frequency	1.4 GHz
	L1 cache	private, separate instruction and data cache. Each 32kB in size.
	MCDRAM	shared, direct mapped cache

3.4.3 Network Traffic Analysis of Realistic and Unrealistic Models

To compare the effects of realistic (proposed approach) and unrealistic (default gem5) models, we observe the buffer utilization at each router as shown in Figure 3-6. Figure 3-6A shows a 4x4 mesh where the MCs are connected to each directory which is the default implementation of gem5. Traffic is uniform except for the tile 0 where the PE resides (tile numbers are as shown in Figure 3-1). Figure 3-6B shows a realistic scenario where every other parameter is kept the same, but MCs are connected to boundary routers. This does not display the uniform traffic distribution as shown in Figure 3-6A. Traffic patterns show hotspot columns due to MC placement which increases latency and saturates the throughput. The 4x4 mesh and MC placement configurations used in Figure 3-6 are for illustration only. Experiments are carried out using the parameters mentioned under section 3.4.1.



A MCs modeled at each tile

B MCs modeled only at places marked in blue.

Figure 3-6. Buffer utilization in routers when RADIX benchmark running on core 0.

As a result of this congestion and more packets being sent through the network, the realistic model shows a 54.9% more network flit latency on average across *FFT, FMM, RADIX and LU* benchmarks compared to the unrealistic model with a similar topology. A comparison

of network latency, NoC power usage and execution times with different benchmarks is shown in Figure 3-7. As stated before, the default gem5 model does not permit cluster mode exploration as MCs are collocated with directories at every tile. Even then, if the default model is used for exploration, the results in Figure 3-7 show that it gives highly optimistic results for NoC latencies and power.

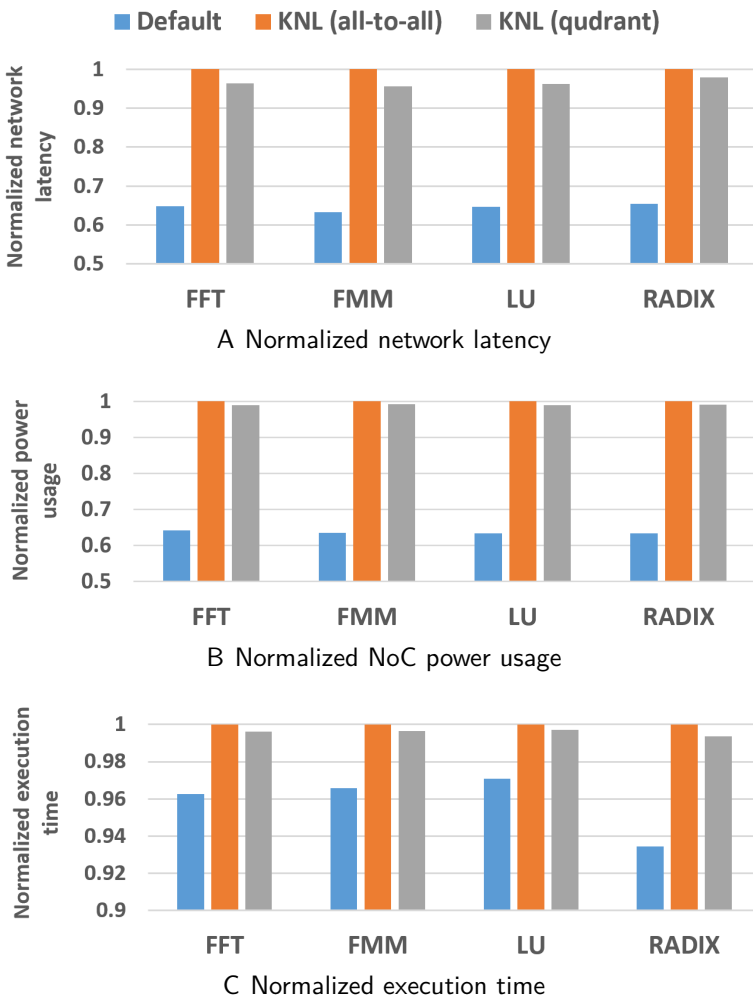


Figure 3-7. Power and performance comparison for different models.

3.4.4 Traffic Variation with Different Cache Coherence Protocols

Another factor that effects the NoC traffic behavior is the cache coherence protocol [144]. The default gem5 NoC implementation already captured these variations as it correctly implemented the PE to directory affinity. We explored the effects of different cache coherence

protocols - (1) MI, (2) MESI Two-Level, (3) MOESI CMP Directory, (4) MOESI Hammer [18]. Figure 3-8 shows traffic variation with different cache coherence protocols when RADIX benchmark is running on a 4x4 Mesh NoC. Figure 3-8A, show buffer utilization at each router when MI cache coherence protocol is used with the default gem5 implementation, which assumes a memory interface at each tile. Similar to the observation of Figure 3-6A, the traffic shows a uniform gradient across the routers without any congestion. Figures 3-8B shows the same results with our modified implementation. We observe that the patterns remain consistent across cache coherence protocols. As evident from Figure 3-8C & 3-8D, Figure 3-8E & 3-8F and Figure 3-8G & 3-8H pairs, this observation remains the same across other 3 cache coherence protocols as well. Therefore, the observations made in Section 3.4.3 hold irrespective of the cache coherence protocol. This is expected as our modification only affects the affinity between directory and MC. With increased sharing in cache coherence protocols, the traffic in NoC increases. But, the hotspot locations and the traffic distribution remain the same.

3.4.5 Network Latency Comparison for Different MC Placements

Xu et al. explored network traffic behavior with different MC placements (Column 0/7, Column 2/5, Diamond, Slash and Optimal) and concluded that the "Optimal" was best for similar benchmarks [61]. Figure 3-9 shows network flit latency when realistic MC placement configurations in [61] as well as gem5 default (unrealistic) model are tested across different benchmarks. As further evidence for the highly optimistic nature of the default gem5 model, we can see that the latency is significantly less when compared to realistic MC placement models.

In contrast to the conclusion in [61], "Optimal" is no longer the best placement, when the directory-based coherence is introduced. The results not only depend on MC placement, but also on PE placement and coherence protocol. Considering only the realistic placements described in [61], Column 2/5 configuration turns out to be the best by 9.0% compared to the worst configuration (Slash) when running *BASICMATH*. Column 2/5 also beats "Optimal" by 5.3% on average across all benchmarks. The traffic congestion caused by adjacent MCs in

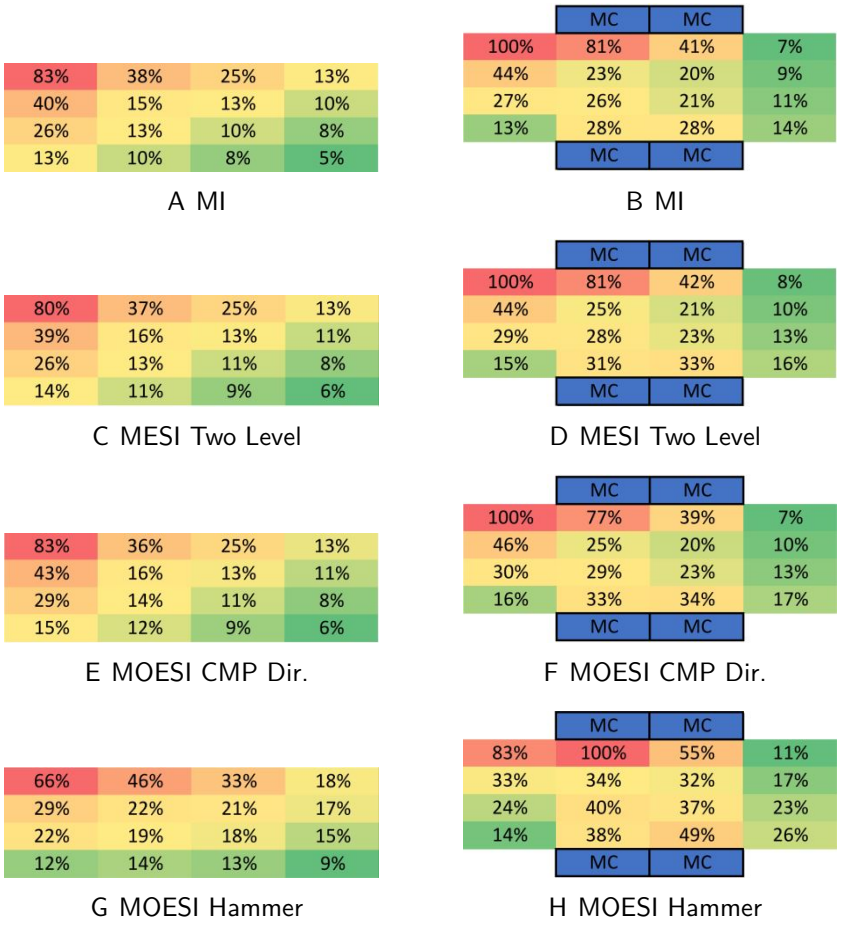


Figure 3-8. Buffer utilization in routers when RADIX benchmark is running on a 4x4 Mesh with different cache coherence protocols.

Column 2/5 configuration is compensated by the reduced hop counts, since it gives smallest average hop count.

3.4.6 Exploration of Memory and Cluster Modes and Validation with Results from the KNL Hardware Platform

As seen from results in Figure 3-9, the affinity between the PE, MC and directory plays a major role in network traffic behavior. To explore this further, we experimented with different cluster and memory modes available in the KNL architecture and validated the simulation results with Intel Xeon Phi 7210 platform. The results for both all-to-all and quadrant cluster modes as well as flat and cache memory modes are shown in Figure 3-10. Compared to all-to-all flat mode, all-to-all cache mode gives the highest benefit with 18.62% less execution

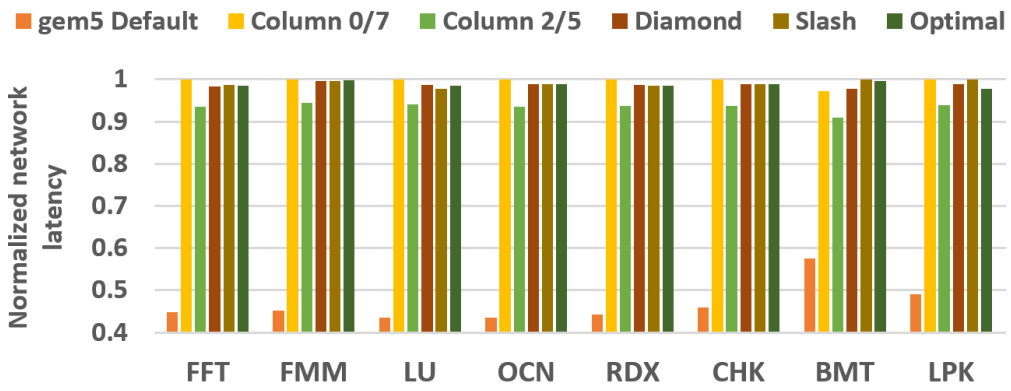


Figure 3-9. Normalized network latency with different MC placements.

time on average across all benchmarks. That is an average speedup of 1.23. The average speedup of quadrant flat mode over all-to-all flat is small (1.013). This is mainly because the benchmarks do not stress the platform too much and memory access latency hinders the savings of network flit latency. These observations are in agreement with the Intel Xeon Phi results [4], which further justifies the accuracy of our approach. Clearly, it is not possible to perform these memory and cluster mode explorations without the proposed NoC modeling framework.

Figure 3-10. Normalized execution times with KNL architecture modeled in gem5.

3.5 Summary

In this chapter, I explored how the network traffic behaves when directory-based cache coherence is introduced. The change in traffic behavior is not captured in the widely used gem5 simulator and thus it can lead to unrealistic conclusions. This chapter made three important contributions. First, I observed that the gem5 model is faulty as it models an MC at every tile, thus eliminating coherence traffic and also not practical due to pin limitations.

Next, I implemented an accurate and realistic model to explore MC placement in an 8x8 mesh with 16 MCs. The results showed that the previous conclusions made without considering coherence traffic are no longer valid. My studies showed that optimization methods should not only consider MC placements, but also PE placement and coherence protocol to come to realistic conclusions. Finally, experimental results demonstrated that the affinity between MC and directory controller can be manipulated with different cluster and memory modes such as quadrant, all-to-all, flat and cache modes introduced in Intel's KNL architecture to achieve better performance and power results. My proposed exploration framework is vital for emerging NoCs with wireless, optical and 3D networks. Furthermore, the modified gem5 architecture enables the exploration of NoC optimization and security countermeasures proposed in subsequent chapters.

CHAPTER 4
NOC-AWARE CACHE RECONFIGURATION AND EXPLORATION

Since security countermeasures can introduce overhead to network-on-chip (NoC) based system-on-chips (SoC), exploration of performance, power and area optimization opportunities is of utmost importance. Therefore, before discussing security attacks and countermeasures, in this chapter, I focus on improving the performance and energy efficiency of NoC-based SoCs. While existing research has addressed NoC optimization in several directions, I point out an important missing piece in NoC-based SoC optimization that has not been explored before.

Dynamic cache reconfiguration (DCR) has been well studied as an effective cache energy optimization technique [73, 81]. DCR allows runtime tuning of the cache parameters (e.g., cache size, associativity and line size) after deciding when and how to configure them using optimization algorithms. This enables the chip multiprocessor (CMP) to optimize energy consumption while maintaining the application's quality of service (QoS) standards. Dynamic tuning of multi-level caches is challenging since the exploration space is prohibitively large. Even with a small number of tunable cache parameters, the exploration space can grow exponentially making it impractical to do a simulation-based exhaustive exploration [78]. Several heuristics were proposed to reduce the exploration space by utilizing the independence between various cache parameters [78, 87]. Unfortunately, these approaches suffer from accuracy and inconsistency across different architectures. Previous works on DCR have not considered NoC traffic when calculating the overall system energy. Therefore, they are not suitable for making accurate architectural decisions. In this chapter, I present an exploration framework that is developed considering the complete memory hierarchy and NoC traffic. In order to explore the prohibitively large design space effectively, I propose and analyze a machine learning (ML) algorithm which predicts runtime and energy of applications with different cache configurations. This enables us to significantly reduce the exploration time, while maintaining the accuracy within an acceptable range.

In this chapter, I focus on the following four main areas.

1. **DCR-CP-NoC co-optimization:** Since DCR in level 1 (L1) cache and cache partitioning (CP) in shared level 2 (L2) are closely coupled, we explore DCR and CP together in an NoC based many-core architecture and compare it to previous studies on two-level cache configuration in bus-based architectures.

2. **Efficient static profiling:** We propose a machine learning algorithm to reduce the overall static profiling time compared to the exhaustive method by an order of magnitude with minor impact on energy savings. Results are compared with exhaustive as well as heuristic-based approaches.

3. **Dynamic programming based optimization:** We propose a dynamic programming (DP) based algorithm to find optimal L1 cache configurations for each application and L2 partition factors for each core. Effective utilization of DP allows this exploration to run with linear time complexity with respect to the number of cores.

4. **Extensive evaluation:** We accurately model the NoC traffic flow and energy consumption. Then, we evaluate our approach by running 14 benchmarks on a realistic Intel Xeon Phi configuration with 32 tiles [4] using the gem5 full-system simulator [18].

The remainder of the chapter is organized as follows. Section 4.1 presents some background information required to understand my approach. Section 4.2 motivates the need for this work. Section 4.3 describes the exploration framework. Section 4.4 presents the experimental results. Finally, Section 4.5 summarizes the chapter.

4.1 Background

In a typical CMP architecture, L1 caches are private for each core, whereas the L2 cache is shared across all cores. Such an arrangement introduces dependencies between the L1 and L2 caches as the configuration of one can affect the cache accesses of the other, and vice versa [82]. Therefore, DCR techniques are commonly used to optimize L1 caches. Similarly, CP improves performance by eliminating the inter-task interference on a shared cache [80]. Hence, it is employed to judiciously divide portions of L2 cache to each core.

Figure 4-1 shows a standard NoC-based many-core architecture with a shared L2 cache, private instruction (IL1) and data (DL1) caches. Both L1 and L2 caches are reconfigurable. L1 cache configuration can be changed by changing its capacity, line size and associativity. L2 cache is partitioned among all the cores and the partitions are decided depending on the application requirements.

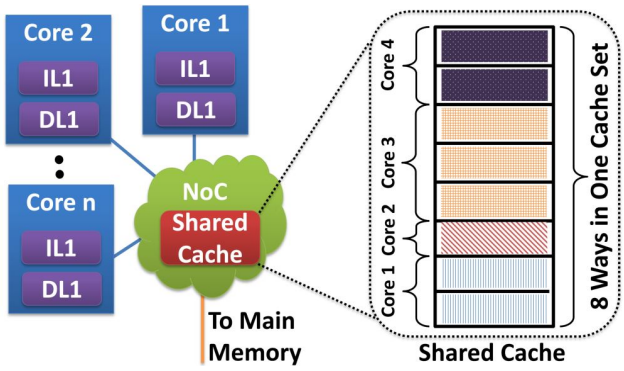


Figure 4-1. Many-core architecture with private instruction (IL1) and data (DL1) caches as well as shared L2 cache.

The cache architecture used in our work contains four separate banks operating as four separate ways. Figure 4-2 shows cache configurability of L1 caches with an illustrative example. If the base cache size is 4kB, we have four 1kB banks [145] (Figure 4-2a). Cache associativity can be reconfigured by concatenating neighboring ways (Figure 4-2b). To change cache sizes, gated V_{dd} is used to shutdown banks causing the effective cache size to shrink. A 4kB cache can have 4-way, 2-way and 1-way (direct mapped) associativity. However, if the cache size is reduced to 2kB, it can only have 2-way and 1-way associativity because a 4-way associativity will mean shutting down or concatenating half of the bank/way and that is not supported in the architecture (Figure 4-2c). Line size can be changed by changing the number of unit length blocks fetched during each cache access (Figure 4-2d).

This reconfigurable cache architecture has very little overhead and requires simple hardware changes [73]. Runtime re-configuration of L1 cache is done by using special configuration registers. The configuration registers inform the cache tuner which is a lightweight process implemented on hardware, to concatenate ways to change associativity. Similarly, the configuration registers can be configured to shut down ways causing the cache size to change. It is important to note that our contribution is an efficient technique that determines which cache configuration should be used for a given application. As explained in related work and following sections, run-time configuration of caches is a well studied problem

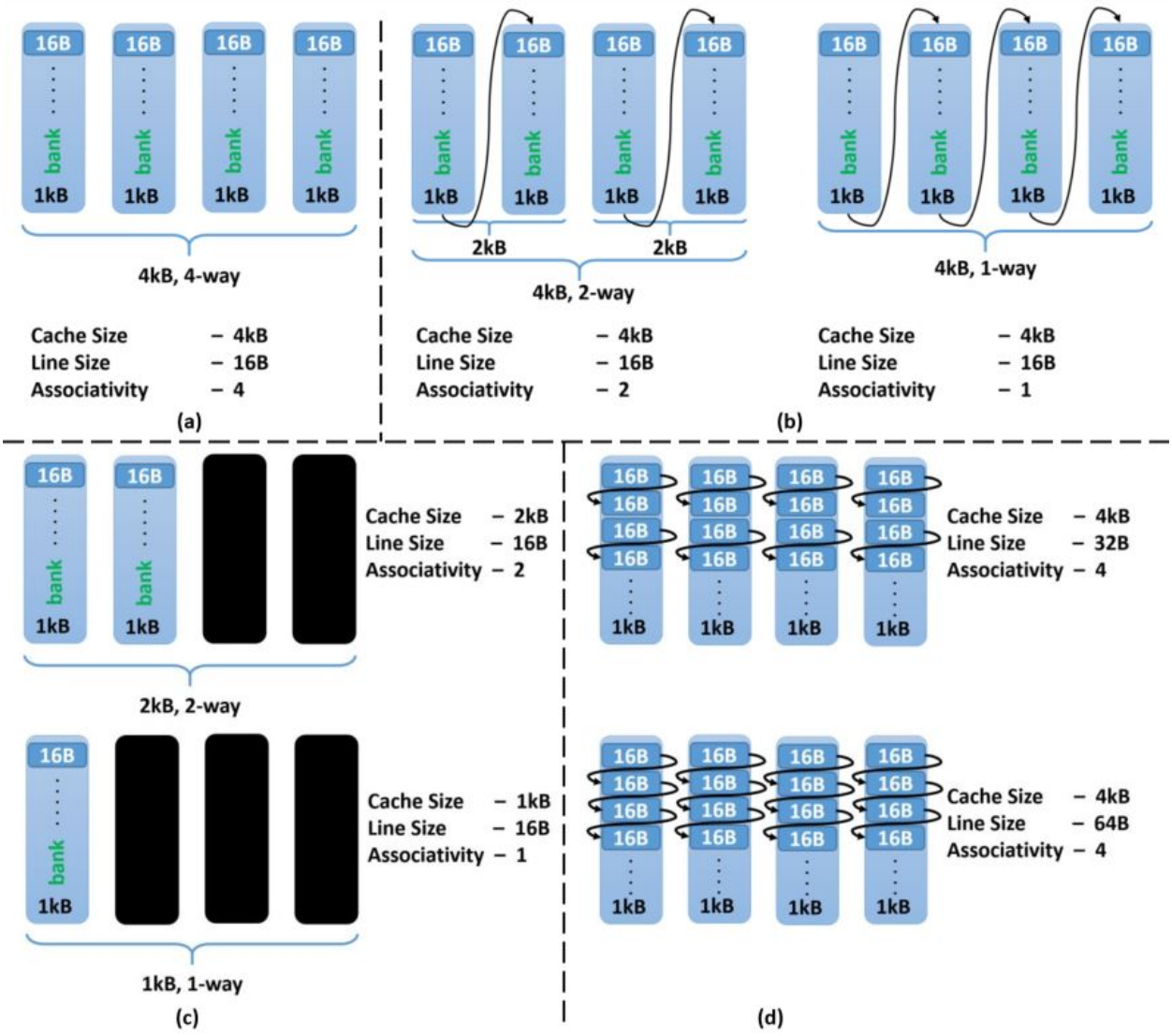


Figure 4-2. Cache configurability of a 4kB cache arranged as 4 banks.

and our architecture proposes to use existing mechanisms to tune the cache once an optimum configuration is found.

For the shared L2 cache, we use a way-based partitioning method that differs from traditional LRU replacement policy that implicitly partitions cache sets based on demand [77]. Figure 4-1 depicts a cache set with 8-way associativity which can be partitioned in the granularity of ways. Each core will access only the group of ways assigned to it in all the cache sets. LRU replacement is enforced in each individual group by maintaining a separate set of "age bits". Way-based partitioning is useful for exploiting energy efficiency. Number of

ways assigned to a core is referred to as the core's "partition factor. Core 1 in Figure 4-1 for example has an L2 partition factor of two. In our study we use static cache partitioning. In other words, each core's L2 partition factor is constant throughout system execution and is predetermined. Since L1 DCR has a major impact on L2 CP, the exploration framework should support tuning of all possible parameters simultaneously [78]. For example, the number of L2 accesses is dependent on the number of L1 misses. Also, the miss penalty of the L1 cache is dependent on the configuration of the L2 cache.

4.2 Motivation

4.2.1 Impact of DCR on Power and Performance

As an illustrative example, we ran FFT benchmark from the SPLASH-2 benchmark suite on an architecture model similar to Intel Knights Landing (KNL) introduced in Chapter 3, and recorded energy and runtime values for two DL1 configurations - 32K_2W_32B[1] and 8K_1W_32B [146]. IL1 configuration is the same for both executions. As shown in Figure 4-3, they give different runtime and energy values. If the performance of the system is not critical, using the configuration that gives the least energy consumption (8K_1W_32B) should be selected. If the performance target is to execute the application in $4ms$, we can observe in Figure 4-3C that 8K_1W_32B does not meet the requirement. In that case, 32K_2W_32B should be selected which meets the desired performance even though it consumes more energy.

In reality, the number of possible cache configurations is much larger than two. For example, Section 4.3.3 shows that there are 504 valid cache configurations in our specific exploration framework. When NoC power consumption is considered, the energy optimization problem becomes even more complex.

[1] In this chapter, we show cache configurations using three parameters. For example, 32K_2W_32B represents a cache with 32kB cache capacity, 2-way set associativity and 32 byte line size

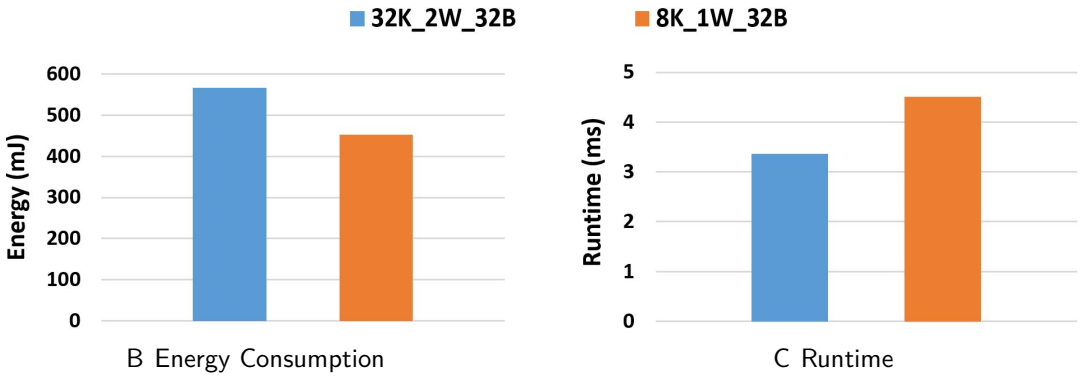


Figure 4-3. Energy consumption and runtime of two cache configurations running FFT.

With NoC being the most preferred interconnection technology in modern CMPs, it is imperative to account for the NoC while exploring the cache configurations. As shown in Figure 4-4, power consumption of the NoC portion of CMP is a function of the application executed, L1 and L2 configurations. We observe that with increased L1 cache size, NoC power consumption decreases. This is expected because increasing L1 cache size causes less L1 misses and as a result, fewer packets being injected to the network. From the NoC power model illustrated in Section 4.3.2, we can see that decreasing number of packets on the network decreases the NoC power. NoC power is shown in this figure instead of energy for comparisons across applications by eliminating the performance factor. In this study, both the applications are running on exactly the same CMP model.

Figure 4-5 illustrates NoC energy consumption as a percentage of total energy. NoC energy, core energy and energy contribution from other components (last level cache, memory, etc.) are shown for comparison. As expected, the NoC energy consumption decreases with increasing cache size and associativity. More precisely, NoC energy consumption percentage reduces by more than half from 22% to 10% when DL1 cache size is doubled. This is expected since the number of requests going to off-chip memory is reduced. It further reduces to 3% with further increase in DL1 size and IL1 associativity. A fixed line size of 64B and a fixed partition factor of 2 are used for all three experiments. Irrespective of the choice of partition factor, our results show that NoC is an important contributor to overall energy consumption.

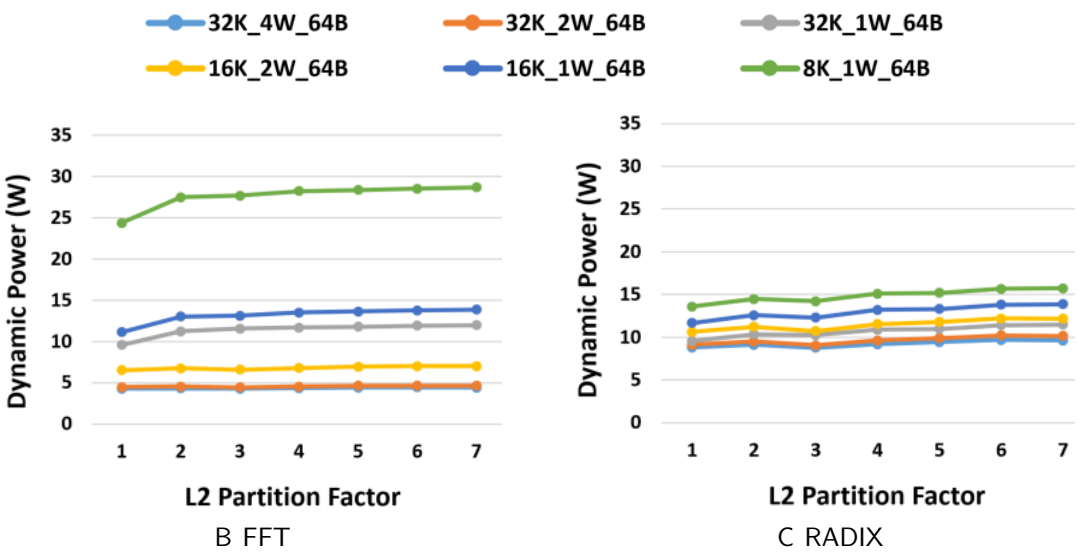


Figure 4-4. NoC dynamic power consumption with different DL1 configurations and L2 partition factors. IL1 cache is fixed at 32K_2W_64B.

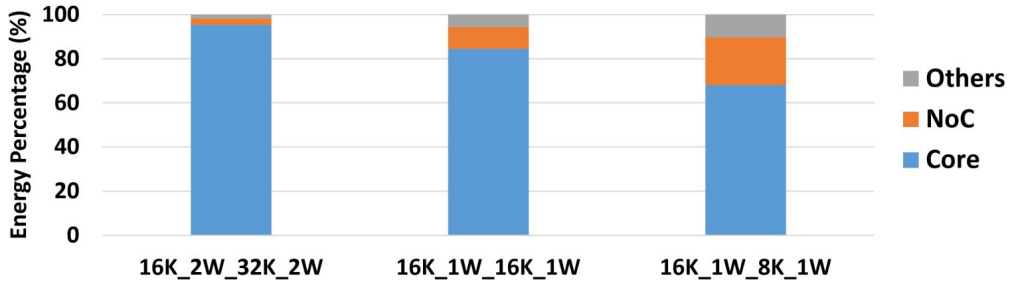


Figure 4-5. Core and NoC energy as a percentage of total CMP energy consumption.

Given the above observations, it is evident that ignoring the energy contribution from NoC when exploring L1 DCR and L2 CP trade-offs in an NoC based many-core architectures can lead to misleading conclusions. Therefore, it is important to model NoC traffic as well as its energy accurately.

4.2.2 Impact of Memory Modes in KNL Architecture

To further emphasize the importance of cache reconfiguration exploration in many-core architectures, we ran some experiments on Intel Xeon Phi 7210 hardware platform which implements the KNL architecture [137]. The selection between different memory modes is essentially a cache reconfiguration as the total amount of memory is fixed and it is divided among shared cache and main memory to fit the application characteristics. The two extreme

configurations out of all the possibilities are Flat and Cache modes since Cache mode allocates all 16GB of MCDRAM memory as cache and Flat mode allocates it as main memory. Since the Hybrid configurations fall in the middle, we ran tests using these extreme configurations to illustrate their effects on application runtime.

Figure 4-6 shows execution time of 6 complex benchmarks running on KNL Flat and Cache modes. All these applications show benefits in the Cache mode. Yet, the percentage speedup varies drastically between a minimum of 2.3% for LINPACK benchmark to a maximum of 77.1% for LBS3D. This behavior is expected when executed on Xeon Phi hardware board since the Cache mode is able to exploit the memory access patterns in LBS3D benchmark, and as a result, provided significant performance improvement compared to Flat mode. On the other hand, Cache mode is slightly better than Flat mode in case of LINPACK memory access patterns. It is important to note that the execution time for each application is normalized with respect to the execution time in Flat mode. Therefore, the comparison shows how much Cache mode will benefit over Flat mode for a given application.

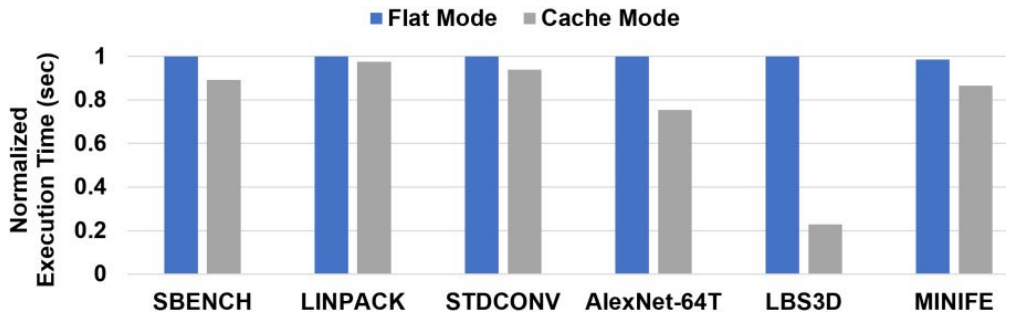


Figure 4-6. Execution time variation in Cache vs Flat memory modes in Intel Xeon Phi 7210 processor.

This proves that state-of-the-art CMPs support different cache configurations and as a result, application power and performance can vary drastically. However, instead of the limited number of possible cache configurations introduced in the KNL architecture, it is beneficial to have more tunable cache parameters so that more optimization opportunities are available. This has not been possible due to the lack of simulation frameworks that capture all the significant components in a CMP including NoC, caches and main memory. The simulation

framework should be accompanied by an efficient cache reconfiguration framework which selects the best cache configuration for a given application.

4.2.3 Design Space of Possible Cache Configurations

Even though having more tunable cache parameters gives more optimization opportunities, it can cause the number of possible cache configurations (design space) to grow exponentially. This makes it infeasible to run all possible cache configurations exhaustively and come to a conclusion on the best cache configuration for a given application. The existing solutions for this as explained in Chapter 2, suffer from loss of accuracy and inconsistency across architectures. Therefore, we propose a machine learning based approach that achieves high accuracy and drastically reduces the exhaustive exploration time. We provide a detailed calculation for the motivation of our approach in Section 4.3.3.1 after we have defined the terminology used in this chapter in section 4.3.1.

4.3 Efficient Cache-NoC-Memory Co-Exploration

Figure 4-7 shows an overview of our cache reconfiguration framework. It consists of two parts: (i) static profiling of application programs, and (ii) runtime (dynamic) inter-application cache reconfiguration. The static profiling is used to determine the most profitable cache configuration for a given application under various design constraints. This is done by reducing the exploration space using a machine learning based approach (Section 4.3.3) and running a dynamic programming based optimization algorithm to find the optimum cache configurations (Section 4.3.4 and Section 4.3.5). An optimum cache configuration table is created at this stage - each row of the table contains the most profitable cache configuration for an application. When an application starts execution, the cache tuner changes the cache based on the configuration stored in the table at runtime. This cache tuning is possible according to the reconfigurable cache architecture described in Section 4.1. The goal of this chapter is to develop a framework that efficiently constructs the optimum cache configuration table. Using the configurations in the table to tune the cache during runtime is beyond the scope of

this chapter. Runtime configuration is a well studied problem and existing mechanisms can be applied to deal with it.

This section is organized as follows. We first provide the problem formulation (Section 4.3.1) followed by the power/energy models (Section 4.3.2). The next three subsections describe the three important steps in our static profiling framework: (i) machine learning based static profiling (Section 4.3.3), (ii) dynamic programming based per-core optimization (Section 4.3.4), and (iii) optimization across all cores (Section 4.3.5).

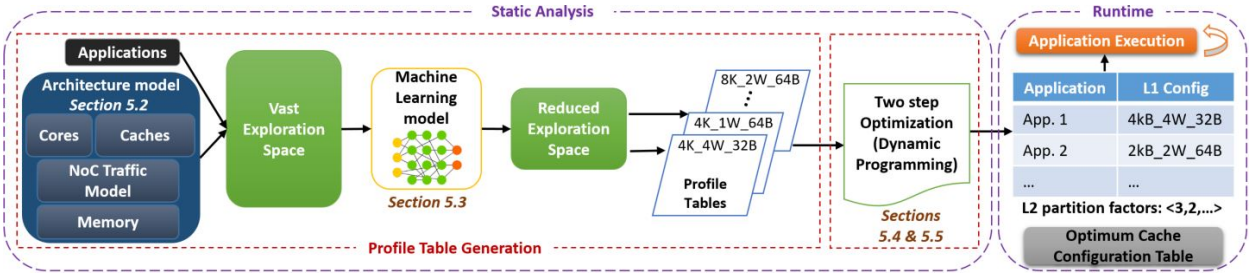


Figure 4-7. Overview and main steps in our proposed approach.

4.3.1 Problem Formulation

We model the many-core system with the following parameters and convert the exploration into a minimization problem.

- The CMP consists of n cores. The set of all cores is denoted by $\mathbb{P}$: $\{p_1, p_2, ..., p_n\}$.
- Each core consists of private IL1 and DL1 caches. Each private cache can be configured into r different configurations as explained in Section 4.1. Set of all cache configurations are - $\mathbb{C}$: $\{c_1, c_2, ..., c_r\}$.
- The L2 cache, which is shared among all n cores, is α-way associative with way-based partitioning support.
- m independent applications $\mathbb{T}$: $\{\tau_1, \tau_2, ..., \tau_m\}$ are executed on the CMP model without violating quality of service (QoS) requirements quantified by a "Performance Target" D. In a given application mix, each application can finish at different times, but there is a common soft deadline by which all applications should be completed. Violations of this performance target deteriorate the QoS. The illustrative example in Section 4.2 introduces the usage of D. How to select D for a given application set is described in more detail in Section 4.4.2.

The assumption of a common soft deadline for an application set, which is represented by the performance target, is made to generate results that are comparable with existing approaches. In many classes of embedded systems, the tasks can be divided into multiple sets of tasks, where the priority as well as the deadline is same for the tasks in each set. In other words, the priority as well as the deadline will be different for tasks in two different sets. In the extreme, each set may contain only one task, thereby, enabling individual deadlines for each task. Our approach is applicable for tasks with individual deadlines without any changes to the proposed algorithm.

The goal of our optimization algorithm is to find a reconfiguration scheme $\mathbf{R}$ for IL1 and DL1 and a partition scheme $\mathbf{\Pi}$ for the L2 cache such that the assigned applications run with minimal energy E without violating QoS standards where;

$$E = E_{cores} + E_{caches} + E_{noc} + E_{memory} + E_{directories} \tag{4-1}$$

The inputs to the proposed algorithm are as follows:

- Set of all possible L1 cache configuration schemes $\mathbf{R}$ which assigns a cache configuration to each IL1 and DL1 cache for each application - $\mathbf{R} : \mathbb{T} \rightarrow C_I, C_D$.
- Set of all possible L2 cache partitioning schemes $\mathbf{\Pi}$: $\{w_1, w_2, ..., w_n\}$ which assigns w_k ways to core k.
- An application mapping scheme $\mathbf{M}$: $\mathbb{T} \rightarrow \mathbb{P}$ which maps the m applications to the available cores. The application mapping is beyond the scope of this chapter and $\mathbf{M}$ is assumed to be available. Here, δ_k denotes the number of applications mapped to core k.

Let $\tau_{k,i}$ denote the i^{th} application running on core k. Similarly, $\epsilon_{k,i}(c_I, c_D, w_k)$ is the total CMP energy consumption for $\tau_{k,i}$ with c_I, c_D as IL1 and DL1 cache configurations, respectively, and w_k partition factor. In other words, $\epsilon_{k,i}(c_I, c_D, w_k)$ denotes the energy contribution to total energy E from τ_i. Here, $t_{k,i}(c_I, c_D, w_k)$ represents the time spent by $\tau_{k,i}$ with the said cache configurations. Then, the optimization problem can be expressed as the minimization of:

$$E = \sum_{k=1}^{n} \sum_{i=1}^{\delta_k} \epsilon_{k,i}(c_I, c_D, w_k) \tag{4-2}$$

subject to:

$$\max_{k=1..n}(\sum_{i=1}^{\delta_k} t_{k,i}(c_I, c_D, w_k)) \leq D \tag{4-3}$$

$$\sum_{k=1}^{n} w_k = \alpha; w_k \geq 1, \forall k \in [1, n] \tag{4-4}$$

As shown in Equations (4-1) and (4-2), E consists of energy consumption in cores, caches, NoC, off-chip memory and directories. The constraint in Equation (4-3) guarantees that all applications will meet the required QoS standards quantified by the performance target D whereas Equation (4-4) verifies that the partitioning scheme is valid.

4.3.2 Cache Coherent Traffic Flow and Energy Models

This section describes the traffic flow and energy model used in the NoC and Cache of our architecture model.

4.3.2.1 NoC traffic and energy model

Since dynamic power consumption of an NoC is a function of the traffic flow, we need to model a realistic traffic flow and use an energy model that captures the changes in the traffic flow. For this purpose, we model the traffic flow of Cache memory mode in KNL architecture. An example is shown in Figure 4-8 to illustrate the traffic behavior of a memory request in Cache mode. When multiple cores are active and send many packets to the network, their contention for resources is captured by the model presented in [147]. We used the same model in our experiments that includes a credit-based flow control mechanism, buffers, arbiters, etc. to capture packet behavior accurately.

In a typical mesh network where a router is connected to each processing element, energy consumption for sending one bit of data from a source tile (t_s) to a destination tile (t_d) can be calculated using the Manhattan distance between them [148]. The "bit energy metric" defined by Ye et al. [149] defines the dynamic part of the communication energy as

$$E_{bit} = E_F + E_L + E_B \tag{4-5}$$

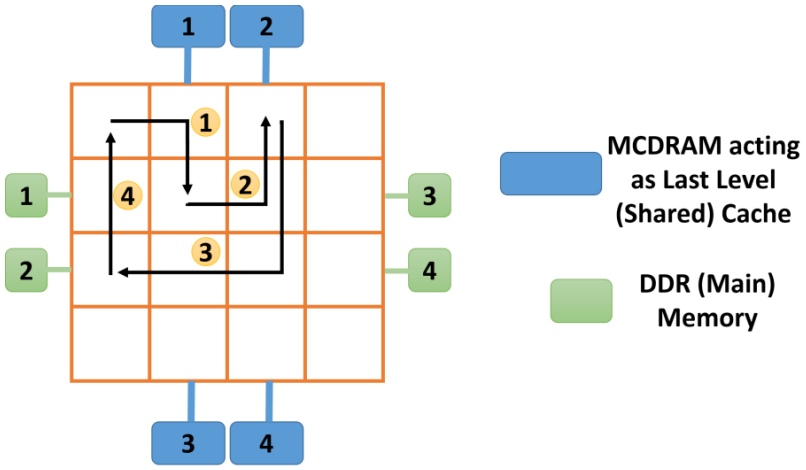


Figure 4-8. Traffic model in Cache memory mode in KNL architecture in case of an L1 and MCDRAM miss.

where E_F, E_L and E_B represent the energy consumption per bit by the switch fabric, link and buffer, respectively. If V_i represents the supply voltage for tile i and bit energy values are measured at V_{DD}, the energy needed to transmit one bit through P tiles can be abstracted by [140];

$$E_{bit}^{t_s,t_d} = \sum_{i \in P} (E_F(i) + E_L(i) + E_B(i)) \cdot \frac{V_i^2}{V_{DD}^2} \tag{4-6}$$

The links connecting the routers consume power because of their switching activity (toggle between logic 0 and logic 1). Power consumption in a traditional router with four pipeline stages (routing computation, virtual channel allocation, switch allocation and switch traversal) is a combination of power consumed in buffers, arbiter, allocator and crossbar switch. This is captured by E_B and E_F.

Assuming that the architecture is fixed, the energy model in Equation 4-6 shows two main contributors to NoC power:

1. Number of packets injected to the network - this is directly related to the number of cache misses in L1 and L2 caches.
2. Average hops traversed by packets - depends on the placement of NoC components such as memory controllers, home directories and last level cache.

This NoC energy model was adopted from the work done by Ogras et al. [135] and it is validated with both simulations and real hardware data.

4.3.2.2 Cache energy model

The total energy consumption of cache (E_{cache}) is not only the energy consumed by the cache memory (E_{array}), but also the energy consumed by the memory addressing path ($E_{address\_path}$) and the I/O path (E_{I/O_path}) [150]. Therefore, the total cache energy consumption can be calculated as;

$$E_{cache} = E_{array} + E_{address\_path} + E_{I/O\_path} \tag{4-7}$$

$E_{address\_path}$ is determined by the switching activity of the address bus. The tag and data memory arrays usually dominate the total cache energy consumption E_{array}. The energy model used in our approach is based on dynamic logic where bit lines are pre-charged on every access. Therefore, the energy consumed by the tag and data arrays will be determined by the number of accesses. The I/O path includes I/O pads as well as address and data buses connected to it. Out of these components, the switching activity of the I/O pads usually dominate that energy component (E_{I/O_path}). Therefore, the three main components of E_{cache} can be computed as follows;

$$E_{array} = \alpha \cdot tag\_access + \beta \cdot blk\_access \tag{4-8}$$

$$E_{address\_path} = \gamma \cdot bsr\_addr\_bus \tag{4-9}$$

$$E_{I/O\_path} = \delta \cdot bsr\_addr\_pad + \epsilon \cdot bsr\_data\_pad \tag{4-10}$$

where;

tag_access - access rates of bit-lines in cache tag arrays

blk_access - access rates of bit-lines in cache block arrays

bsr_addr_bus - bit switching rates of address bus

bsr_addr_pad - bit switching rates of address pads

bsr_data_pad - bit switching rates of data pads

$\alpha, \beta, \delta, \gamma, \epsilon$ - constants depending on VLSI implementation.

This model is similar to the cache energy model implemented in the widely used McPAT simulator and has been verified with hardware data [151]. We used the default energy models available in the McPAT simulator for other components in the SoC.

4.3.3 Efficient Static Profiling Using Machine Learning

We need a profile table including the runtime and energy consumption of each application when running with all valid cache configurations to give as input to the optimization algorithm (second and third steps of Algorithm 2).

4.3.3.1 Design space of possible cache configurations

According to the reconfigurable cache architecture described in Section 4.1, both IL1 and DL1 have $6(= 3 + 2 + 1)$ possible configurations each. When two possible line sizes are used (64B and 32B), changing IL1 and DL1 at the same time, it gives $72(= 6 \times 6 \times 2)$ candidates for IL1 and DL1. It is in-feasible to profile application with all possible L1 reconfiguration schemes $\mathbf{R}$, all possible L2 partition schemes $\mathbf{\Pi}$, for the whole application set $\mathbb{T}$ and all possible application mappings schemes $\mathbf{M}$ [82].

As a solution for this, Wang et al. [78] proposed to reduce the exploration time significantly based on the following observations:

- All the applications in Table 4-3 are independent with no inter-application data sharing. An application can always start and complete on the assigned core without any migrations happening during runtime.
- The L1 cache is private for each core and the configuration of L1 in one core doesn't have any effect on the other core's configuration as there are no multi-threaded applications that map to two cores in our application set.
- With L2 partitioning, each application uses an independent portion of the shared cache which makes it a logical private cache.

With the applications and all the caches being independent, we can profile each application as it was running on a uni-processor with a w_i-way associative L2 cache with the capacity equal to $\frac{w_i}{w} \times original\ cache\ size$. In this case, the total number of simulations required would be $|\mathbf{R}| \times (\alpha - 1) \cdot m$. Thus, it takes $72 \times 7 \times 14 = 7056$ simulations for

14 application. Even after this reduction, the simulations take approximately one month to complete. Since the number of simulations can grow exponentially with the number of tunable cache parameters and applications, this exhaustive exploration becomes in-feasible, when the design space becomes arbitrarily large. As a solution, we propose a machine learning based approach which runs only few simulations and uses that as training data to tune a neural network model which then predicts the rest of the profile table. With this method, time required to build the full profile table would be much lower. Table 4-1 shows profile table entries for five L1 cache configurations (out of 72 possible configurations) created by machine learning.

Table 4-1. A portion of the profile table generated from the machine learning algorithm for "stringsearch" benchmark.

	L2 Partition Factor													
IL1 and DL1 Configuration	Time	Energy	Time	Energy	Time	Energy	Time	Energy	Time	Energy	Time	Energy	Time	Energy
32kB_2W_64B_16kB_1W_64B	3424	744	2744	674	2704	669	2608	659	2592	658	2560	655	2536	652
16kB_1W_64B_8kB_1W_64B	5216	1159	4528	1087	4488	1082	4400	1074	4376	1072	4352	1071	4328	1067
8kB_1W_64B_32kB_2W_64B	2016	576	1976	573	2032	578	1960	571	1904	565	1888	564	1888	562
32kB_4W_64B_32kB_2W_64B	2016	576	1976	573	2032	578	1960	571	1904	565	1888	564	1888	562
32kB_2W_64B_16kB_2W_64B	2232	606	2120	594	2192	601	2112	594	2040	587	2024	585	2024	585

4.3.3.2 Algorithm

First, a portion of the profile table entries are filled up by simulating an application on different configurations. Next, several models are trained with the collected data, and the model with least error is selected. If the error is within a threshold, then we predict the rest of the profile table entries using that model. Otherwise, we collect additional training data by running more simulations, and repeat the procedure until the error threshold criteria is satisfied. Algorithm 1 describes our machine learning based approach. Here, X_{all} denotes the set of all possible configurations for an application. X_{sim} is the set of configurations on which we already simulated the application and have the corresponding table entries Y_{sim}. The remaining configurations are in X_{pred} and needs to be predicted. It is evident that $X_{all} = X_{sim} + X_{pred}$. Initially, all the table entries are empty. Thus, $X_{pred} = X_{all}$ and $X_{sim} = \emptyset$ [line 2-3]. In the subsequent iterations, some of the configurations (X_{sel}) are randomly selected from X_{pred} for simulation [line 6]. After simulating with X_{sel} configurations, Y_{sel} entries are put into the

profile table. Consequently, X_{sel} configurations are removed from X_{pred} and added to X_{sim} [line 7-9]. Size of X_{sel} determines the number of simulations carried out in each iteration. Next, we train multiple models with the filled table entries Y_{sim} and their configurations X_{sim}. For model building purpose, these entries and configurations are divided into three groups - training (X_{train}, Y_{train}), cross-validation (X_{cv}, Y_{cv}) and testing (X_{test}, Y_{test}) [line 10-11]. This essentially means that $X_{sim} = X_{train} + X_{cv} + X_{test}$ and $Y_{sim} = Y_{train} + Y_{cv} + Y_{test}$. Training set is used for training the model. Cross-validation is used for hyper-parameter tuning such as learning rate or regularization parameter. Test set is used for determining the prediction accuracy of models. A standard split is used in our experiments - 70% for training, 15% for cross-validation, and 15% for testing. These models are further tuned by parameter sweeping [line 13-19]. In our experiments, we trained a shallow neural network, and changed the number of nodes in the hidden layer during the parameter sweep. If the prediction error is larger than the error threshold ϵ, we collect more simulation data and repeat the model building procedure. Alternatively, if error is within the threshold, then the model is used for predicting the rest of the profile table. Input to the model will be the configuration set X_{pred} and output will be the predicted energy and runtime values $(\hat{Y}_{pred})$ [line 24]. Full profile table is built by combining predicted data $\hat{Y}_{pred}$ and simulated data Y_{sim} [line 25].

To calculate the error threshold, Normalized Root Mean Square Error (NRMSE) is used (Equation 4-11). Using NRMSE is advantageous, since it is a relative measurement. So the same threshold can be used for all applications.

$$NRMSE = 100\% * \sqrt{MSE}/\bar{Y} \tag{4-11}$$

where, MSE is the mean squared error and $\bar{Y}$ is the average value. NRMSE is measured over the test data set, Y_{test} and $\hat{Y}_{test}$ while calculating the threshold. As experimental results demonstrate, our machine learning framework can give a speedup of 7.76 times when the error threshold is set at 5%.

Algorithm 1 Profile table generation using machine learning

for each application **do** ▷ Model Building
  $X_{pred} = X_{all}$
  $X_{sim} = \emptyset$
  $min\_error = \infty$
  while $X_{pred} != \emptyset$ **and** $min\_error > \epsilon$ **do**
    $X_{sel} = randomSample(X_{pred})$
    $X_{pred} = X_{pred} - X_{sel}$
    $X_{sim} = X_{sim} + X_{sel}$
    $Y_{sel} = simulate(X_{sel})$
    $[X_{train}, X_{cv}, X_{test}] = distribute(X_{sel})$
    $[Y_{train}, Y_{cv}, Y_{test}] = distribute(Y_{sel})$
    for each regression algorithm **do**
      for each param **do** ▷ Parameter sweep
        $model = train(X_{train}, Y_{train}, X_{cv}, Y_{cv}, param)$
        $\hat{Y}_{test} = predict(X_{test}, model)$
        $error = nrmse(Y_{test}, \hat{Y}_{test})$
        if $error < min\_error$ **then**
          $min\_error = error$
          $sel\_model = model$
        end if
      end for
    end for
  end while
  $\hat{Y}_{pred} = predict(X_{pred}, sel\_model)$ ▷ Profile table entry prediction
  $Y_{all} = Y_{sim} + \hat{Y}_{pred}$
end for

4.3.4 Per-Core Optimization

We tackle the optimization problem in two steps. First we find the optimum L1 cache configuration for each core and then optimize across all cores to find the best L2 partition scheme. This subsection illustrates optimizing each core with best L1 cache configuration. Since static partitioning is used, applications running on one core will have the same partition factor - w_k. Thus, we find best $\mathbf{R}$ under different L2 partition factors. Mathematically, the goal is to find L1 configuration $\mathbf{R}$ to minimize $E_k(w_k) = \sum_{i=1}^{\delta_k} \epsilon_{k,i}(c_I, c_D, w_k)$ constrained by $\sum_{i=1}^{\delta_k} t_{k,i}(c_I, c_D, w_k) \leq D$, with k and w_k fixed $\forall k \in [1, n]$ and $\forall w_k \in [1, \alpha - 1]$.

To find minimum energy consumption, this can be discretized to simplify the problem and we use a dynamic programming (DP) algorithm on that. Let $\epsilon_k^{min}(w_k) = \sum_{i=1}^{\delta_k} min\{\epsilon_{k,i}(c_I, c_D, w_k)\}$

and $\epsilon_k^{max}(w_k) = \sum_{i=1}^{\delta_k} max\{\epsilon_{k,i}(c_I, c_D, w_k)\}$ denote minimum and maximum possible energy on core k. Thus, the energy consumption $E_k(w_k)$ of core k is bounded by these min and max values. Let Φ_i^E be the current solution for the first i applications where E is the cumulative energy consumption achieving best runtime. Runtime $T[i][E]$ for Φ_i^E is stored in a two-dimensional table T and the solution for Φ_i^E is updated whenever the runtime can be improved. The recursive formula used in our DP approach is shown in Figure 4-9.

$$\textbf{If } (T[i][E] > T[i-1][E - \epsilon_{k,i}(c_I, c_D, w_k)] + t_{k,i}(c_I, c_D, w_k)) \{$$
$$T[i][E] = T[i-1][E - \epsilon_{k,i}(c_I, c_D, w_k)] + t_{k,i}(c_I, c_D, w_k)\}$$

Figure 4-9. Recursive formula for dynamic programming

The final optimal energy consumption $E_k^*(w_k)$ is found by;

$$E_k^*(w_k) = \min\{E_k \mid T[\delta_k][E_k] \leq D\} \tag{4-12}$$

At the end of the DP algorithm, Equation 4-12 provides the solution for core k with partition factor w_k, which has minimum energy consumption with QoS constraints satisfied.

4.3.5 Optimizing the Entire CMP

Now that we have the optimal energy $E_k^*(w_k)$ calculated for a given partition factor w_k on core k, we combine the solutions across all cores to find the minimum total CMP energy consumption E^* within all partition schemes $\mathbf{\Pi}$ as;

$$E^* = min\{\sum_{k=1}^{n} E_k^*(w_k)\}, \quad \forall\{w_1, w_2, ..., w_n\} \in \mathbf{\Pi} \tag{4-13}$$

Our approach across all the steps starting from building the profile table is summarized in Algorithm 2. Optimizing for each core to find best L1 configuration is shown in step 2. At each iteration (lines 2 to 25), all discretized energy values (ϵ) and L1 cache configurations for current application $\tau_{k,i}$ are examined. The recursive DP algorithm given in Figure 4-9 is included in lines 4 to 22 in two parts - initially for the first application and then from application 2 to δ_k.

Time complexity for step 2 is $O(n \cdot \alpha \cdot \delta_k \cdot |\mathbf{R}| \cdot (\epsilon^{max} - \epsilon^{min})/q_t)$, where $\epsilon^{max} - \epsilon^{min}$ is the energy range and q_t is discretization interval. We have used a constant q_t in the DP algorithm throughout our exploration. Step 3 iterates through all valid $\mathbf{\Pi}$ to find the final solution with time complexity $O(n \cdot |\mathbf{\Pi}|)$. Since cache parameters are constant for the given architecture model, time complexity for both step 2 and step 3 is linear with respect to n.

When calculating the exact runtime without neglecting the constant factors, it is important to note that $|\mathbf{\Pi}|$ depends on the constraint in Equation 4. Therefore, this becomes the classic "stars and bars" problem in combinatorics. A theorem in combinatorics states that for any pair of positive integers n and k, the number of k-tuples of positive integers whose sum is n, is equal to the number of $(k-1)$-element subsets of a set with $n-1$ elements. Therefore, according to our notation, $|\mathbf{\Pi}| = \binom{\alpha-1}{n-1}$. However, this calculation is not required since in reality, the number of ways (α) is as small as 8 or 16 (8-way or 16-way). Given that n represents the number of active cores, If n is greater than α, the cache partitions will have to be shared among cores. If n is less than or equal to α, $|\mathbf{\Pi}|$ would be small (e.g., if $\alpha = 16$ and $n = 12$, $|\mathbf{\Pi}| = 1365$). Therefore, this calculation can be done in constant or linear time for most of the scenarios.

4.4 Experiments

4.4.1 Experimental Setup

We used a cycle-accurate full-system simulator - gem5 [152] and "GARNET2.0" [147] NoC model that is integrated with gem5, to model the multi-core architecture. Our goal was to model a realistic architecture that included reconfigurable L1, L2 caches, and an accurate NoC traffic model. Our previous work modeled the KNL architecture on the gem5 simulator by modifying the default gem5 source to capture the behavior of Memory and Cluster modes in KNL [153]. There are other multi-core architectures such as Tilera TILE64 [154] and Kalray's MPPA-256 [155], which allows for predictable data transfers and composition of memory accesses. However, we did our experiments on the gem5 KNL model since we had validated the

Algorithm 2 Selection of optimal cache configurations

Run Algorithm 1 ▷ **1st step**: Building profile table (Section 4.3.3)
for k = 1 to n **do** ▷ **2nd step**: Optimize on each core (Section 4.3.4)
    for $w_k = 1$ to $\alpha - 1$ **do**
        for $\epsilon = \epsilon_k^{min}(w_k)$ to $\epsilon_k^{max}(w_k)$ **do**
            for $c_I, c_D \in \mathbb{C}$ **do**
                if $\epsilon_{k,1}(c_I, c_D, w_k) == \epsilon$ **then**
                    if $t_{k,1}(c_I, c_D, w_k) < T[1][\epsilon]$ **then**
                        $T[1][\epsilon] = t_{k,1}(c_I, c_D, w_k)$
                    end if
                end if
            end for
        end for
        for i = 2 to δ_k **do**
            for $\epsilon = \epsilon_k^{min}(w_k)$ to $\epsilon_k^{max}(w_k)$ **do**
                for $c_I, c_D \in \mathbb{C}$ **do**
                    $\epsilon' = \epsilon - \epsilon_{k,i}(c_I, c_D, w_k)$
                    if $T[i-1][\epsilon'] + t_{k,i}(c_I, c_D, w_k) < T[i][\epsilon]$ **then**
                        $T[i][\epsilon] = T[i-1][\epsilon'] + t_{k,i}(c_I, c_D, w_k)$
                    end if
                end for
            end for
        end for
        $E_k^*(w_k) = \min\{\epsilon_k \,|\, T[\delta_k][\epsilon_k] \leq D\}$
    end for
end for
for all $\Pi_j = \{w_1, w_2, ..., w_n\} \in \mathbf{\Pi}$ **do** ▷ **3rd step**: Optimize across all cores (Section 4.3.5)
    $E_j^* = \sum_{k=1}^{n} E_k^*(w_k)$
    $E^* = min(E^*, E_j^*)$
end for
return E^*

gem5 simulator model with results from real hardware (Intel Xeon Phi 7210 hardware board) in our previous work [153].

However, the full KNL implementation is quite complex to be modeled on gem5. For example, gem5 does not support tiles with 2 cores. Hence, there is one core in each tile in our experiments. This mimics the scenario where one core in each tile is switched off in the hardware board. Furthermore, KNL runs AVX512 instructions whereas our gem5 KNL model runs X86 instructions. Cache sizes were chosen such that the applications we used get a

realistic hit percentage of around 95% in L1 cache. If we used a larger cache size, the L1 hit rate would be 100%, and any discussion about cache reconfiguration will be meaningless. Modeling the entire KNL architecture in a simulator is beyond the scope of this chapter. Our goal was to model a realistic NoC traffic model. Even though the absolute values are not the same, the relative advantages/disadvantages of reconfiguration are accurately captured as shown in our previous work as well [153]. The core contributions of this chapter - cache reconfiguration mechanism and machine learning-based exploration space reduction remains intact irrespective of the architecture.

The complete set of simulation parameters are summarized in Table 4-2. Power results were obtained by feeding the gem5 output statistics to McPAT power modeling framework [151].

Table 4-2. System configuration parameters.

Parameter Class	Parameter	Value
Processor Configuration	Number of cores	32
	Core frequency	1.4 GHz
	Instruction set architecture	x86
Memory System Configuration	L1 cache	private, reconfigurable, separate instruction and data cache. Each 32kB in size.
	L2 Cache	reconfigurable, shared cache. 512kB in size.
	Cache coherence	MESI Two-Level directory-based cache coherence protocol
	Memory Size	4GB DDR
Interconnection Network Configuration	Topology	8x4 Mesh
	Routing scheme	X-Y deterministic
	Router	4 port, 4 input buffer router with 3 cycle pipeline delay
	Link latency	1 cycle

To evaluate the effectiveness of our approach, we use 14 benchmarks selected from MiBench [156] - bitcnts, crc, dijkstra, patricia, qsort, sha, stringsearch and SPLASH-2 [146] - FFT, Radix, Lu, FMM, Cholesky, Water-Nsquared, Barnes benchmark suites. In order to

make the size of MiBench benchmarks comparable with SPLASH-2, we use reduced (but well verified) input sets. Table 4-3 lists the application sets which are combinations of the selected benchmarks used in our experiments. We choose 3 application sets where each core contains 2 benchmarks, 2 application sets where each core contains 3 benchmarks and 1 application set where each core contains 4 benchmarks.

Out of the 32 cores in the 8x4 Mesh interconnect, we chose 16 cores just for experimental purposes. Simulation time can be prohibitive with larger number of cores. This is evident from the results in Figure 4-14. Besides being practical, this core configuration enables us to mimic that about 50% of the cores from a chip will be typically utilized at a given time. As mentioned in Section 4.3.1, mapping of applications to cores is beyond the scope of this chapter and the application mapping is assumed to be given as an input. The task mapping problem in soft real-time system has been studied before [157]. For our experimental results, to get the application mapping, we ran each application with the base cache configuration and grouped them so that the total execution time for each set of applications is comparable. We cannot get the execution time to be exactly the same since it depends on cache configuration. However, the intent is to have a fair comparison and therefore not pair up a task with quick execution time with a task with very long execution time. This was done to make sure behaviors such as NoC congestion will be captured throughout application runtime. Otherwise, if most cores finish their tasks and only some are running, the experiments will give results that do not capture traffic congestion in NoC. If a task is parallelized, it can be viewed as multiple tasks and mapping can be performed (a smart mapping algorithm is likely to map them to the cores in a cluster). The performance target D is set in a way that there is a feasible L1 cache assignment for every partition factor in every core. In other words, all possible L2 partition schemes can be used.

4.4.2 Performance Target Selection

Equation (4-3) gives the performance target as a measure of providing the expected QoS for the application sets. Figure 4-10 shows the optimal energy consumption variation

Table 4-3. Application sets from the MiBench and SPLASH-2 benchmarks.

Application Set	Cores 1, 2, 3, 4	Cores 5, 6, 7, 8	Cores 9, 10, 11, 12	Cores 13, 14, 15, 16
Set 1	stringsearch, sha	FFT, Barnes	stringsearch, Lu	sha, FFT
Set 2	crc, Barnes	Radix, Lu	Cholesky, sha	qsort, FMM
Set 3	bitcnts, stringsearch	FMM, Water-Nsquared	Barnes, Lu	Cholesky, sha
Set 4	Radix, Lu, FFT	crc,sha,stringsearch	qsort, Barnes, Water-Nsquared	bitcnts, FMM, stringsearch
Set 5	patricia, Water-Nsquared, Barnes	dijkstra, bitcnts, qsort	Cholesky, Radix, crc	patricia, sha, Lu
Set 6	Lu, stringsearch, FFT, patricia	Water-Nsquared, FMM, qsort, dijkstra	FFT, Cholesky, dijkstra, sha	crc, Radix, qsort, bitcnts

as a function of the performance target D for application set 1 in Table 4-3. We swept the target from $8100ms$ to $8600ms$. When the target is shorter than $8180ms$, there was no feasible solution. As the target was increased, it converged to the optimal which was $5324\ mJ$. Therefore, it is clear that the performance target will affect the best possible energy calculated by our approach. In our experiments, we selected the target such that each core can converge to an optimal energy under the base configuration. In this example, the selected target was $8400ms$.

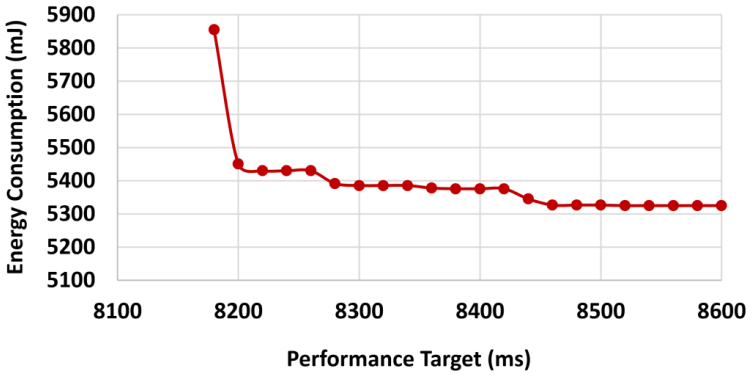


Figure 4-10. Energy consumption variation with performance target. A relaxed target leads to more energy savings.

4.4.3 Accuracy of Profile Tables Built with Machine Learning

Section 4.3.3 describes our machine learning algorithm for building the profile table with less number of simulations. Figure 4-11 shows the total amount of training data required for all benchmarks over different error thresholds. Here, training data is expressed as a percentage value. This is the ratio of profile table entries filled using simulations and total number of profile table entries, for all 14 benchmarks. Error threshold is expressed using NRMSE. As expected, the more training data we use, lower the error threshold is. Error is less than 6% even with only 10% of training data. This essentially means approximately an order-of-magnitude speedup in profile table generation time compared to exhaustive simulations. Figure 4-12 provides the training data requirement for each benchmark. We can see that some benchmarks require a lot of training data for accurate predictions (e.g., Barnes, Lu, dijkstra etc.), while some benchmarks need much less (e.g., bitcnts, sha etc.). This observation forms the basis of using error threshold instead of fixing training data percentage. Error threshold based approach allows more simulation time for benchmarks that require more training data to be accurate.

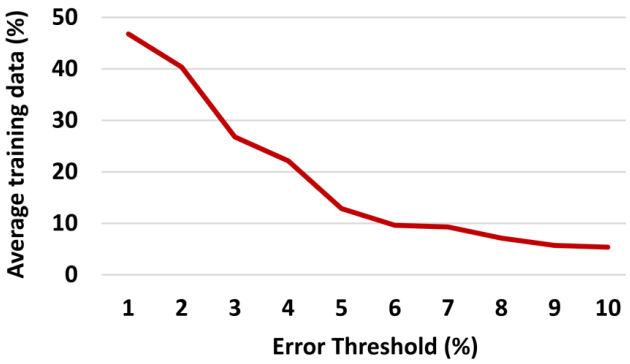


Figure 4-11. Average training data required with varying error threshold for all benchmarks.

To evaluate our approach, we compare results of the predicted profile tables with profile tables obtained from exhaustive and heuristic-based approaches. Figure 4-13 compares the accuracy of the following 3 strategies that can be used to build the profile table by showing the optimal energy consumption;

- **Exhaustive:** All cache configurations explored exhaustively. This acts as the reference which gives the global optimal solution.

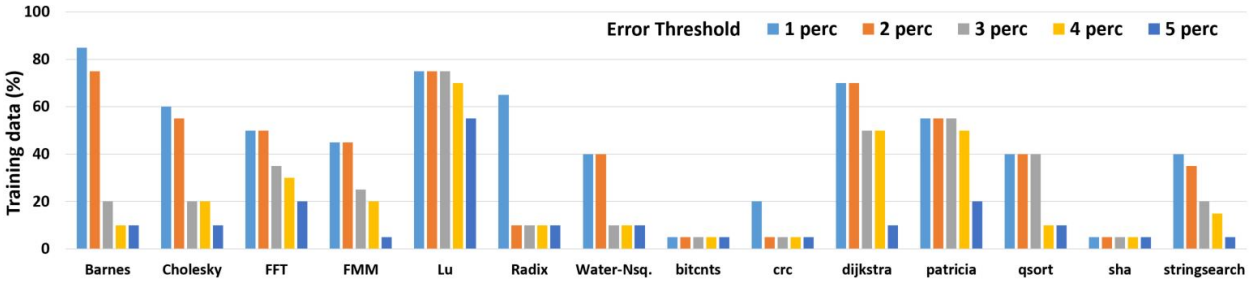


Figure 4-12. Required training data for different error thresholds.

- **Our approach (ML):** Configurations selected keeping the error threshold at 5% as training data and Algorithm 1 used for predicting.
- **Heuristic [78]:** Cache configurations selected using ICT heuristic (configurations where DL1 and IL1 are the same) proposed by Wang et al..

Algorithm 2 uses these profile tables to get the optimum cache configurations while maintaining QoS standards. Those values are then normalized to the energy consumption of the base cache configuration. As discussed in Section 4.4.1, our reconfigurable L1 cache has a base size of 32kB and a shared 512kB L2 cache. We observe in Figure 4-13 that compared to the base configuration, the profile table built with the exhaustive approach can achieve 18.49% energy savings on average across all application sets. The energy savings provided by our approach (ML) is very accurate (within 0.95% on average). It achieves the highest performance in set 1 with an error of only 0.2%, whereas heuristic gives an error of 6.9%. Heuristic-based approach gives relatively worse results as it populates only a portion of the profile table based on ICT. Therefore, it is likely that the optimum configuration found by exhaustive exploration is not in the profile table at all. In contrast, our approach uses the training data to predict the whole table and therefore, depending on the accuracy of the prediction, converges closer to the optimal.

The runtime of these approaches are shown in Figure 4-14. It takes 36.73 days to complete the exhaustive exploration using the modified gem5 simulator on an unparallelized setup. Heuristic selects 16.67% of the total exploration space and hence, shows a speedup of six times. In contrast, our approach selects only 12.86% of the total exploration space

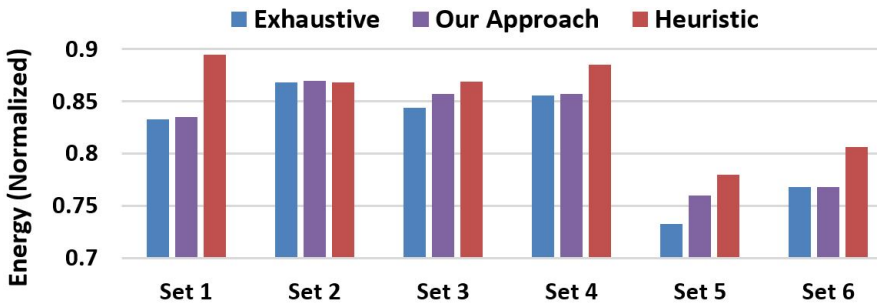


Figure 4-13. Energy consumption observed compared to the Base cache configuration across all profile table generation techniques.

as training data and takes 5 minutes to tune the neural network model which results in an effective speedup of 7.76 times. The time taken to train the ML algorithm is negligible compared to the profiling time. Therefore, we obtain higher accuracy while consuming less time compared to the heuristic approach.

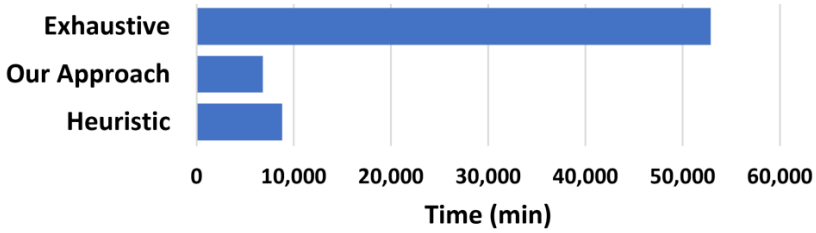


Figure 4-14. Times taken for different static profiling approaches.

Our machine learning approach populates the whole profile table unlike the heuristic method which populates less than 10% of it. This allows us to estimate energy consumption for all possible cache configurations, and as a result, leads to better energy savings. Specifically, our approach provides up to 7% (3% on average) improvement in energy efficiency while being 1.29 times faster compared to heuristic method. Most importantly, our approach is 7.76 times faster compared to the exhaustive method and still provides close to optimal energy savings (on average deviation of 0.05%).

4.5 Summary

This chapter explored cache configuration optimization in an NoC based many-core architecture and addressed the issue of large exploration space in DCR. The proposed approach

significantly reduces the exhaustive exploration time by employing a machine learning based approach. This approach selects only a few configurations and trains a neural network model which then predicts the rest of the profile table. The proposed DCR algorithm then finds the optimum L1 configuration for each core and optimum partition factor for the shared L2 cache. The results showed an average energy savings of 18.49% compared to the base configuration across all application sets. The ML predictions gave accurate (less than 1% error) energy savings compared to the exhaustive method with a 7.76 times speedup which proves to be better than previously proposed heuristic-based methods both in terms of accuracy and speedup. Overall, my approach provides an order-of-magnitude reduction in exploration effort with negligible impact on accuracy.

CHAPTER 5
INCREMENTAL CRYPTOGRAPHY FOR NOC COMMUNICATION

Protecting communications between IPs, which involve asset propagation, is a major challenge and requires additional hardware implementing security such as on-chip encryption and authentication units. However, implementation of security features introduce area, power and performance overhead. Security engineers have to take into account these non-functional and real-time constraints while designing secure architectures to address various threats [95]. Therefore, it is crucial to develop a lightweight security architecture that can provide the desired security with tolerable impact on area, power and performance.

In this chapter, I propose a lightweight encryption scheme based on "incremental encryption" that can provide confidentiality to NoC packets. My solution takes advantage of the unique characteristics of NoC traffic, and as a result, it has the ability to construct a "lighter-weight" encryption scheme without compromising the security. Incremental cryptography has been explored in areas such as software virus protection [158] and code obfuscation [159]. To the best of my knowledge, my approach is the first attempt to utilize incremental encryption to implement a lightweight and secure NoC architecture. The goal of using incremental encryption is to design cryptographic algorithms that can reduce the effort of encryption/decryption by reusing the previously encrypted/decrypted memory fetch requests/responses rather than re-computing them from the scratch. In my framework, data is encrypted at the NI of each secure IP core. The NI is chosen to accommodate the encryption framework so that each packet can be secured before injecting into the NoC. Prior research on NoC security have proposed similar architectures where the security framework was implemented at the NI [90, 101]. Our major contributions of this chapter are as follows:

- I show that consecutive NoC packets that contain memory fetch requests/responses differ only by a few bits while communicating between IP cores and memory controllers in an SoC.
- I propose a lightweight encryption scheme based on incremental cryptography that exploits the unique NoC traffic characteristics observed above.

- I show that my solution is resilient against existing NoC attacks, and it significantly improves the performance compared to state-of-the-art NoC encryption methods.

The rest of the chapter is organized as follows. Section 5.1 introduces some of the key concepts I have in this chapter. Section 5.2 motivates the need for my work. Section 5.3 describes my approach for lightweight encryption. Section 5.4 presents the experimental results. Finally, Section 5.5 summarizes the chapter.

5.1 Background

In this section, I first provide a brief overview of concepts used in this chapter.

5.1.1 Symmetric Encryption Schemes

A symmetric encryption scheme $S = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ consists of three algorithms defined as follows:

- The key generation algorithm is written as $K \leftarrow \mathcal{K}$. This denotes the execution of the randomized key generation algorithm $\mathcal{K}$ and storing the return string as K where β is the length of the key.
- The encryption algorithm $\mathcal{E}$ produces the ciphertext $C \in \{0,1\}^l$ by taking the key K and a plaintext $M \in \{0,1\}^l$ as inputs, where l is the length of the plaintext. This is denoted by $C \leftarrow \mathcal{E}_K(M)$.
- Similarly, the decryption algorithm $\mathcal{D}$ denoted by $M \leftarrow \mathcal{D}_K(C)$, takes a key K and a ciphertext $C \in \{0,1\}^l$ and returns the corresponding $M \in \{0,1\}^l$.

5.1.2 Block Ciphers

A block cipher typically acts as the fundamental building block of the encryption algorithm ($\mathcal{E}$). Formally, it is a function (E) that takes a β-bit key (K) and an n-bit plaintext (m) and outputs an n-bit long ciphertext (c). The values of β and n depend on the design and are fixed for a given block cipher. For every $c \in \{0,1\}^n$, there is exactly one $m \in \{0,1\}^n$ such that $E_K(m) = c$. Accordingly, E_K has an inverse block cipher denoted by E_K^{-1} such that $E_K^{-1}(E_K(m)) = m$ and $E_K(E_K^{-1}(c)) = c$ for all $m, c \in \{0,1\}^n$.

When using block ciphers to encrypt long messages, the plaintext (M) of a given length l is divided into b substrings (m_q) where each substring is $n(= \frac{l}{b})$ bits long and n is called the block size. Block ciphers are used in operation modes where one or more block ciphers

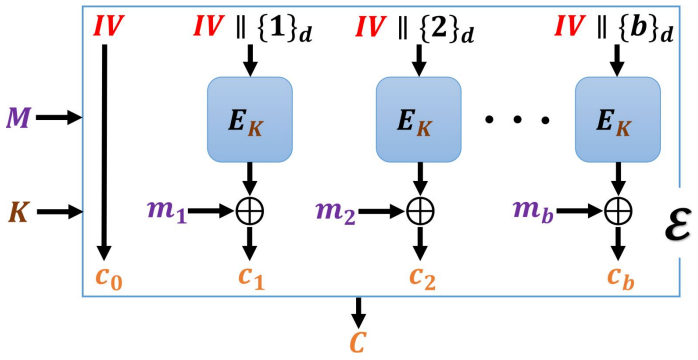


Figure 5-1. A block cipher-based encryption scheme using counter mode.

work together to encrypt n-bit blocks and concatenate the outputs at the end to create the ciphertext of l bits. Figure 5-1 shows the "counter mode" (CM) which is a popular operation mode. CM also uses an initialization vector (IV) which is concatenated with a d-bit value counter (e.g., if $d = 4$, $\{1\}_d = 0001$) before inputting to the block cipher. This is done to create domain separation by giving per message and per block variability. The decryption process is shown in Algorithm 3. In fact, the decryption process is the inverse of the encryption scheme shown in Figure 5-1.

Algorithm 3 Decryption process of Counter Mode

Input: ciphertext to decrypt C
Output: plaintext corresponding to the ciphertext M
procedure $\mathcal{D}_K$
    for all $q = 1, ..., b$ **do**
        $r_q \leftarrow E_K(IV||\{q\}_d)$
        $m_q \leftarrow r_q \oplus c_q$
    end for
    $M \leftarrow m_1 \,\|\, m_2 \,\|\, ... \,\|\, m_b$
    return M
end procedure

5.1.3 Incremental Cryptography Overview

Consider a scenario that involves encrypting sensitive files/documents. Once a file is encrypted initially, there may be minor changes in the original file. In such a scenario, if typical encryption is used, the previous encrypted file will be discarded and a new encryption will be performed on the modified file. However, since these changes are very small in comparison to

the size of the file, encrypting the entire file again is clearly inefficient. Incremental encryption can give significant advantages in such a setup [160]. Updating an obfuscated code to accommodate patches and video transmission of images when there are minor changes between frames, are two similar scenarios [159]. Incremental encryption allows to find the cryptographic transformation of a modified input not from scratch, but as a function of the encrypted version of the input from which the modified input was derived. When the changes are small, the incremental method gives considerable improvements in efficiency.

5.2 Motivation

The IPs use the capabilities given by the NoC to communicate with each other and to request/store data from/in memory. The packets injected into the network can be classified into two main categories - (1) control packets and (2) data packets. For example, a cache miss at an IP will cause a control packet to be injected into the network requesting for that data from the memory. The memory controller, upon receiving the request will reply back with a data packet containing the cache block corresponding to the requested address. The formats of these packets are shown in Figure 5-2. The NI divides the packet into flits ("fliticization") before injecting into the network. Flits are the basic building blocks of information transfer between routers. Sensitive data of each flit is encrypted by the NI and injected into the network through the local router. Encryption process of a packet consumes time as each block has to be encrypted and concatenated to create the encrypted packet. Depending on the parameters used for the block cipher (block size, key size, number of encryption rounds, etc.), the time complexity of the process differs. If each packet is encrypted independently, it takes $z \times T$ time to encrypt all of them, where z is the number of packets and T is the average time needed to encrypt one packet.

As discussed in Section 5.1, the idea of incremental encryption is to develop a scheme where the time taken to encrypt an incoming packet should not be dependent on the packet size, but rather on the amount of modifications done compared to the previous packet. To explore how to use this idea in the context of NoC, we profiled the number of bit changes

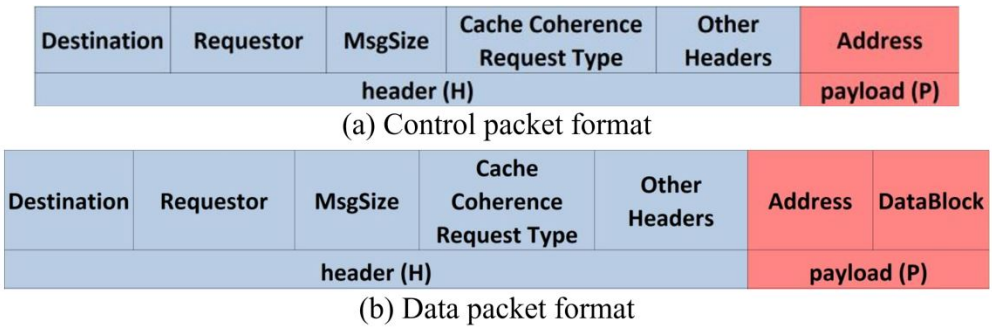


Figure 5-2. Packet formats for control and data packets. Blue shows header (H) which is sent as plaintext. Red shows the payload (P) with sensitive data encrypted.

between consecutive packets generated by a particular IP. Figure 5-3 shows the number of bit differences as a percentage of memory fetch requests (control packets) when running five benchmarks (FFT, FMM, LU, RADIX, OCEAN) from the SPLASH-2 benchmark suite on the gem5 full-system simulator [152]. More details about the experimental setup is given in Section 5.4.1. Out of the 64 bits of data to be encrypted, according to the default gem5 packet size, the maximum number of bit difference between consecutive packets was 13 bits in all benchmarks. On average, 30% of the packets differed by only one bit. This is expected since an application running on a core most likely accesses memory locations within the same memory page which differs by only a few bits.

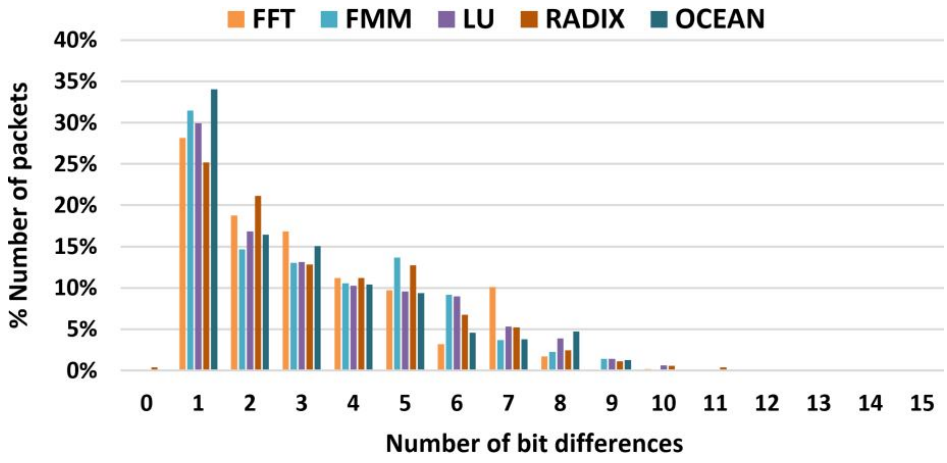


Figure 5-3. Number of bit differences between consecutive memory fetch requests in SPLASH-2 benchmarks.

Since encryption is done in blocks, we profiled this data assuming a block size of 16 bits [161]. In this case, up to 16 consecutive bit differences can be considered for each block, and the maximum number of blocks for 64 bits of secure data is 4. The results showed that on average, 80% of the packets differ by only one block and the other 20% differ by two blocks for the benchmarks we used. Similar to memory fetch requests, we profiled the response memory data packets as well. Since the response contains a whole cache block consisting of data modified by calculations, we don't observe the same optimization opportunity shown by memory fetch requests. However, it still shows that 15% of consecutive packets are identical. These observations show that the encryption process can be significantly optimized using incremental encryption.

5.3 Incremental Encryption

This section describes our incremental encryption scheme in detail. First, we give an illustrative example to demonstrate the merit of exploiting unique traffic characteristics using incremental encryption. Then we elaborate the major components in our framework.

Illustrative example: Figure 5-4 shows an example on how incremental encryption can improve the performance of an NoC. It shows the encryption process of three consecutive NoC packets (each with 16 bits) using two methods (i) traditional encryption, (ii) incremental encryption. In traditional encryption, both packets are encrypted sequentially using the two 8-bit block ciphers. In incremental encryption, each packet is compared with the previous packet and only the different blocks are encrypted. Identical blocks are filled with zeros and header bits are added to indicate the changed blocks. The decryption process uses previously received packets and header information to reconstruct the new packets. Only the first packet has to be fully encrypted since there is no prior packet for comparison. This example shows a speedup of 1.43 times. However, when many packets are encrypted, the time spent to encrypt the first packet becomes negligible and as a result, we observe a significant performance improvement as shown in Section 5.4.2. A detailed description of the methodology is given in the next three subsections.

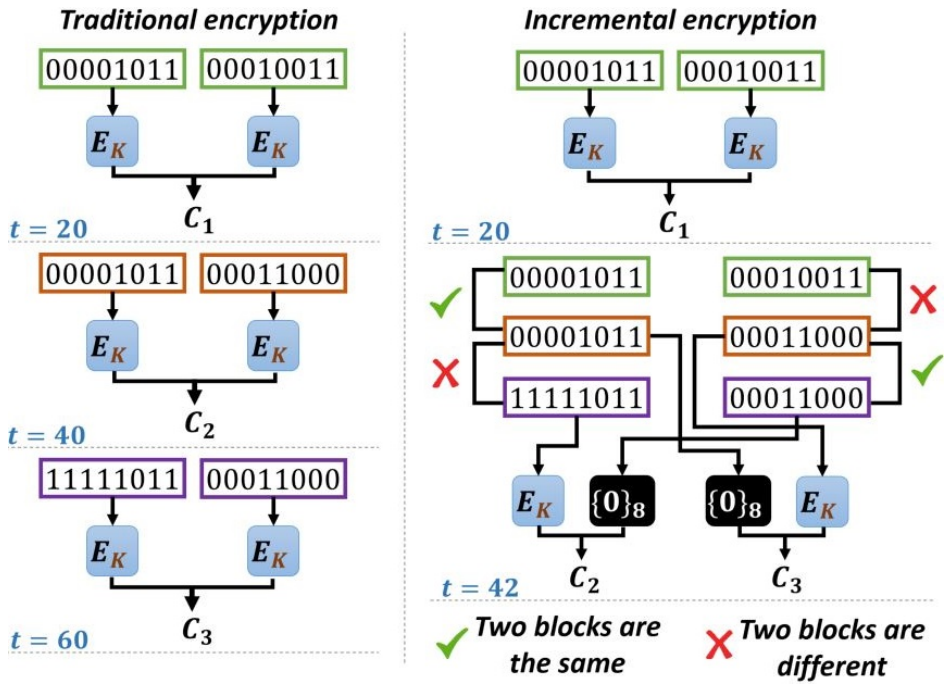


Figure 5-4. Illustrative example of using incremental encryption.

5.3.1 Overview

Figure 5-5 shows an overview of our proposed NoC security framework. It consists of two main components: (i) incremental crypto engine, and (ii) encryption scheme which includes the block ciphers. Each packet sent from an IP core has two main parts: (i) packet header (H) which is sent as plaintext across the network, and (ii) payload (P) which should be encrypted before sending to the network. Both header and payload are sent to the incremental crypto engine to start the incremental encryption process. We consider the payload to be divided into b blocks. For example, the 64-bit payload of a control packet will contain four 16-bit blocks ($b = 4$) numbered 1 through 4 starting from the least significant byte. Our encryption scheme uses block ciphers arranged in counter mode [162]. A detailed explanation of parameters used in our experiments is given in Section 5.4.1.

Algorithm 4 describes our incremental encryption process. When a packet is sent from the IP core, the incremental crypto engine first identifies which blocks are different compared to the previous packet (line 6). This is done by comparing with the previous packet

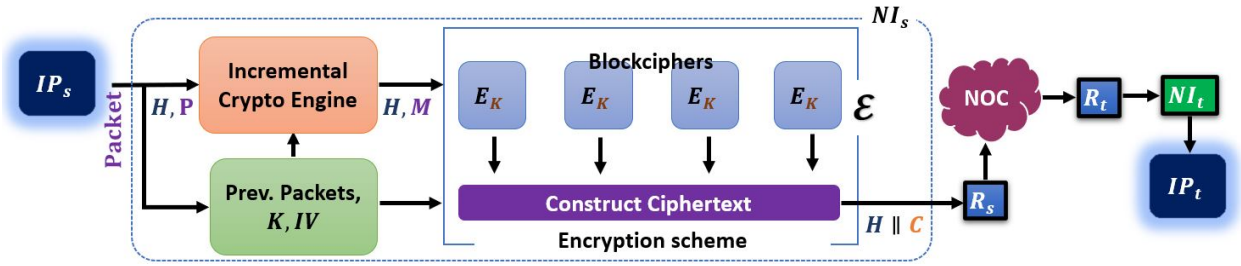


Figure 5-5. Overview of the proposed security framework.

payload (P_{i-1}) which is stored in a register inside the NI. In our model, only two packets are required to be stored for the two different packet types (control and data) at the sender's end. Similarly, the receiver's side also stores the most recent packet for each packet type. In addition to that, the key (K) and initialization vector (IV) for the encryption scheme are also stored by both sender and receiver IPs. Once block differences are computed, it is then sent to the encryption scheme which encrypts only the different blocks (line 7). The final ciphertext is derived from the encrypted blocks and block comparison results (line 8). Additional header bits are also computed in this step to be used by the decryption process. Finally, the header and encrypted payload are concatenated to create the final packet and injected into the network (line 9). At the destination node, the inverse process takes place. It also stores the previous packet for each packet type, and therefore, can construct the next packet using the stored packet and the incoming packet data. Since we store the previous packets in special registers, we don't have to encrypt/decrypt the full packet. We send only the changed blocks and the receiver replaces the changed blocks with its modifications to construct the new packet.

The remainder of this section elaborates the major components of our NoC security framework. Section 5.3.2 explains the $compareBlocks$ function which is implemented in the incremental crypto engine. Section 5.3.3 presents our encryption scheme $\mathcal{E}$ and $constructCipherText$ function in Algorithm 6 and Algorithm 7, respectively.

5.3.2 Incremental Crypto Engine

The operation of the incremental crypto engine is outlined in Algorithm 5. The payload (P_i) sent from the IP core is compared with the previous payload of that type (P_{i-1}) to

Algorithm 4 Encryption process

1: **Inputs**: current packet $packet_i$, previous payload P_{i-1}, key K, initialization vector IV
2: **Output**: encrypted packet consisting of header H_i and encrypted payload C_i
3: **procedure** $encryptPackets$
4: $P_i \leftarrow packet_i.payload$
5: $H_i \leftarrow packet_i.header$
6: $M_i, \delta_i \leftarrow compareBlocks(P_i, P_{i-1})$
7: $C' \leftarrow \mathcal{E}(IV, K, M_i)$
8: $C_i \leftarrow constructCipherText(C', \delta_i)$
9: **return** $H_i \parallel C_i$
10: **end procedure**

identify the blocks that are different (M_i). This can be implemented with a simple XOR operation in hardware (line 4). Once the bitwise differences are obtained, we split the payload into blocks (line 5) to see which blocks are different (lines 6-9). Only different blocks are sent for encryption. The incremental crypto engine also sends the different block numbers (δ_i) to build the complete ciphertext as well as to set the header bits indicating the different blocks to be used by the decryption algorithm.

Algorithm 5 Finding block-wise packet differences

1: **Inputs**: current payload P_i, previous payload P_{i-1}
2: **Output**: different blocks M_i, different block indices δ_i
3: **procedure** $compareBlocks$
4: $bitDiff \leftarrow P_i \oplus P_{i-1}$
5: $B[1], ..., B[k] \leftarrow split(bitDiff, blockSize)$
6: **for all** $x = 1, ..., size(B)$ **do**
7: **if** $B[x] > 0$ **then**
8: $M_i.append(B[x])$
9: $\delta_i[x] = 1$
10: **end if**
11: **end for**
12: **return** M_i, δ_i
13: **end procedure**

5.3.3 Encryption Scheme

We use the counter mode for encryption which uses an initialization vector (IV), a key and the message to be encrypted as inputs and produces the ciphertext. The $IV \parallel \{q\}_d$ string, which is the standard format of the input nonce to counter mode, is used to give per message

and per block variability. In our framework, it is calculated using the sequence number of the packet (let seq_j be the sequence number of packet P_j), a counter, and the IV as $IV \parallel seq_j \parallel q$ to identify different blocks. The block cipher ID ($q \in \{1, 2, 3, 4\}$) changes with each block cipher and the sequence number seq_j varies from packet to packet. As discussed before, the performance improvement is gained by encrypting multiple blocks in parallel. For example, if two consecutive control packets have differences in two blocks each, we can achieve twice the speedup by encrypting both at the same time compared to the traditional (non-incremental) approach where all four block ciphers will be used to encrypt each packet. Algorithm 6 shows the major steps of the encryption scheme.

Algorithm 6 Encrypt selected blocks

1: **Inputs**: initialization vector IV, key K, different blocks M_i
2: **Output**: encrypted blocks C'
3: **procedure** $\mathcal{E}$
4: **for all** $q = 1, ..., 4$ **do**
5: $seq_j \leftarrow getSequenceNumber(P_j)$
6: $r_q \leftarrow E_K(IV \parallel seq_j \parallel q)$
7: $C'.append(r_q \oplus M_i[q])$
8: **end for**
9: **return** C'
10: **end procedure**

C' is stored in a buffer. The final ciphertext is constructed using δ_i and C' as shown in Algorithm 7. Algorithm 7 takes the encrypted value from the buffer for the changed blocks (lines 5-6) and appends n (block size) zeros to identical blocks compared to the previous packet (lines 7-8). It ensures the construction of the same packet size, and as a result, every other functionality from fliticization to NoC traversal remains the same.

To ensure the secure implementation of our approach, the generation and management of keys and nonces needs to be addressed. Many previous studies have addressed this problem in several ways [163, 164].

Algorithm 7 Construct the encrypted payload

Inputs: encrypted blocks C', different block indices δ_i
Output: Encrypted payload C_i
procedure $constructCipherText$
    for all $x = 1, ..., size(\delta)$ **do**
        if $\delta_i[x] > 0$ **then**
            $C_i.append(C'[x])$
        else
            $C_i.append(\{0\}_n)$
        end if
    end for
    return C_i
end procedure

5.4 Experiments

In this section, we first describe the experimental setup used to evaluate our approach. Then, results are presented to show the performance gain achieved through incremental encryption by comparing it with traditional encryption. Next, we discuss the security of the proposed framework and associated overhead.

5.4.1 Experimental Setup

We validated our framework using five benchmarks chosen from the SPLASH-2 benchmark suite. Traffic traces were generated by the cycle-accurate full-system simulator - gem5 [152]. The 4×4 Mesh NoC was built on top of "GARNET2.0" model that is integrated with gem5 [147]. We modified the network interface (NI) to simulate the proposed security framework. We selected the following options to simulate architectural choices in a resource-constrained NoC.

Packet format: For control and data packet formats, we used the default GARNET2.0 implementations which allocates 128 bits for a flit. This value results in control messages fitting in 1 flit, and data packets, in 5 flits. Out of the 128 bits, 64 bits are allocated for the payload (address) in a control packet and data packets have a payload of 576 bits (64-bit address and 512-bit data). This motivated the use of 16-bit blocks to evaluate the performance of our proposed incremental encryption scheme.

Block cipher: We use an ultra-lightweight block cipher - "Hummingbird-2" as the block cipher of our encryption scheme [161]. Hummingbird-2 was chosen in our experiments mainly because it is lightweight and also, with the block size being 16, other encryption schemes can be broken using brute-force attacks in such small block sizes. However, it has been shown in [161] that Hummingbird-2 is resilient against attacks that try to recover the plaintext from ciphertext. It uses a 128-bit key and a 128-bit internal state which provides adequate security for on-chip communication. Considering the payload and block sizes, we used four block ciphers in counter mode for our encryption scheme. Each block cipher is assumed to take 20 cycles to encrypt a 16-bit block and each comparison of two-bit strings incurs a 1-cycle delay [161]. Our framework is flexible to accommodate different packet formats, packet sizes and block ciphers depending on the design requirements. For example, if a certain architecture requires 128-bit blocks, AES can be used while keeping our incremental encryption approach intact.

5.4.2 Performance Evaluation

We present the performance improvement achieved by our approach in two steps: (i) time taken for encryption (Figure 5-6) and (ii) execution time (Figure 5-7). We measured the cycles spent for encryption alone (encryption time) and total cycles executed to run the benchmark (execution time) including encryption time, using our approach as well as traditional encryption. Figure 5-6 shows the encryption time comparison. Our approach improves the performance of encryption by 57% (30% on average) compared to the traditional encryption schemes. The locality in data and the differences in operand values affect the number of changed blocks between consecutive packets. This is reflected in the encryption time. For example, if an application is doing an image processing operation on an image stored in memory, accessing pixel data stored in consecutive memory locations provides an opportunity for performance gain using our approach.

We also compare the total execution time using traditional encryption as well as incremental encryption. Figure 5-7 presents these results. When the overall system including

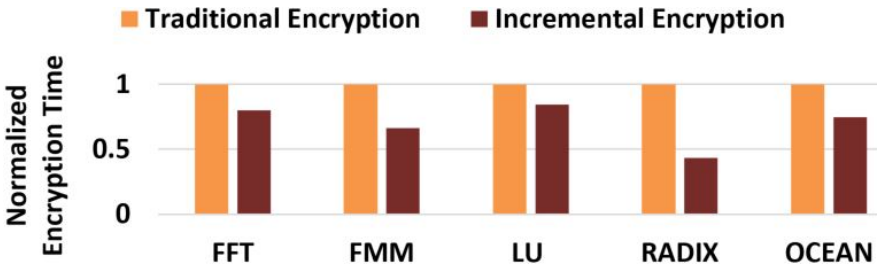


Figure 5-6. Encryption time comparison using traditional encryption and incremental encryption.

CPU cycles, memory load/store delays and delays traversing the NoC is considered, the total execution time improves upto 10% (5% on average). Benchmarks that have significant NoC traversals such as RADIX and OCEAN show higher performance improvement (10%).

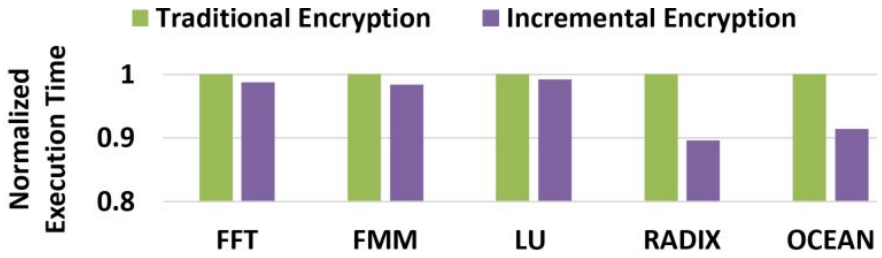


Figure 5-7. Execution time comparison using traditional encryption and incremental encryption.

5.4.3 Security Analysis

When discussing the security of our approach, three main components have to be considered: (i) incremental encryption, (ii) encryption scheme that uses counter mode, and (iii) block cipher.

Incremental encryption: Due to the inherent characteristics of incremental encryption, our approach reveals the amount of differences between consecutive packets. Studies on incremental encryption have shown that even though hiding the amount of differences is not possible, it is possible to hide "everything else" by using secure block ciphers and secure operation modes [158]. Attacks on incremental encryption using this vulnerability relies on the adversary having many capabilities in addition to the ones defined in the threat model. When using incremental encryption to encrypt documents undergoing frequent, small modifications as explained in Section 5.1, it is reasonable to assume that the adversary not

only has availability to the previously encrypted versions of documents but is also able to modify documents and obtain encrypted versions of the modified ones. This attack model allows the adversary to launch chosen plaintext attacks [158]. Discussing security of our approach for known plaintext, chosen plaintext and chosen ciphertext attacks are irrelevant in our design since the adversary does not have access to an oracle that implements the design, nor access to known plaintext/ciphertext pairs. In other words, as long as the block cipher and operation mode is secure, incremental encryption doesn't allow recovering of plaintext from the ciphertext. The same argument has been proven to hold true in previous work on incremental encryption [158, 165].

Counter mode encryption: Using our approach, each block is treated independently while encrypting, and blocks belonging to multiple packets can be encrypted in parallel. In such a setup, using the same $IV \parallel \{q\}_d$ string with the same key K can cause the "two time pad" situation. This is solved by setting the string to $IV \parallel seq_j \parallel q$ as shown in Algorithm 6. It gives per message and per block variability and ensures that the value is a nonce. Our proposed usage of counter mode adheres to the security recommendations outlined in [162].

Block cipher: As discussed above, the security of the proposed framework depends on the security of the block cipher. The security of the block cipher used in our framework, Hummingbird-2, has been discussed extensively in [161]. The first version of the Hummingbird scheme was shown to be insecure [166] and Hummingbird-2 was developed to address the security flaws. After thousands of hours of cryptanalysis, no significant flaws or sub-exhaustive attacks against Hummingbird-2 have been found [161]. Hummingbird-2 approach has been shown to be resilient against birthday attacks on the initialization, differential cryptanalisys, linear cryptanalisys and algebraic attacks. Zhang et al. presented a related-key chosen-IV attack against Hummingbird-2 that recovered the 128-bit secret key [167]. However, the attack requires 2^{28} pairs of plaintext to recover the first 4 bits of the key adding up to a data complexity of $\mathcal{O}(2^{32.6})$ [167]. As discussed before, launching such chosen plaintext attacks is not possible in the NoC setting. A brute force key recovery takes 2^{128} attempts which is not

computationally feasible according to modern computing standards as well as for computing power in the foreseeable future.

Our proposed approach allows easy plug-and-play of security primitives. Any block size/key size/block cipher can be combined with our proposed incremental encryption approach. Note that stronger security comes at the expense of performance. Therefore, security parameters can be decided depending on the desired security and performance requirements.

5.4.4 Overhead Analysis

We implemented our proposed incremental encryption approach using Verilog to show the area overhead in comparison with the original Hummingbird-2 implementation. Our implementation is capable of assigning blocks to idle block ciphers and encrypting up to four payloads in parallel. Merger and scheduler units were implemented to ensure the correctness of final encrypted/decrypted payloads. We conducted our experiments using the Synopsys Design Compiler with 90nm Synopsis library (saed90nm). Based on our results, our proposed approach introduces less than 2% overall area overhead with respect to the entire NoC. When only the encryption unit is considered, the overhead is 15%. This overhead is caused due to components responsible for buffering and scheduling of modified blocks to idle block cipher units as well as computations related to the construction of the final result. Therefore, our proposed encryption approach has a negligible area overhead and it can be efficiently implemented as a lightweight security mechanism for NoCs. While there is a minor increase in power overhead due to the additional components, there is no penalty on overall energy consumption due to the reduction in execution time.

5.5 Summary

In this chapter, I proposed a lightweight security mechanism that improves the performance of traditional encryption schemes used in NoC while incurring negligible area and power overhead. The security framework consists of an encryption/decryption scheme that provides secure communication on the NoC. I used incremental encryption to achieve performance

improvement by utilizing the unique traffic characteristics of packets observed in an NoC. I validated the framework in terms of security to prove that the performance gain is not achieved at the expense of security. Experimental results showed a performance improvement of up to 57% (30% on average) in encryption and authentication time and up to 10% (5% on average) in total execution time compared to traditional encryption while introducing less than 2% overall area overhead.

CHAPTER 6
LIGHTWEIGHT ENCRYPTION AND ANONYMOUS ROUTING

Attacks on NoC communication has become more and more complex over the years. Previous efforts have developed countermeasures against stealing information [90], snooping attacks [168], and even causing performance degradation by launching denial-of-service (DoS) attacks [85, 105]. In this chapter, I present a countermeasure for MIPs operating under the following architecture and threat models.

Threat Model: Figure 6-1 shows an SoC with heterogeneous IPs integrated on a Mesh NoC. The two nodes marked as S (source) and D (destination) are trusted IPs communicating with each other. MIPs integrated on the SoC (nodes shown in red) have the following three capabilities when packets pass through their routers: (1) They can steal information if data is sent as plaintext. (2) If data is encrypted and header information is kept as plaintext, they can gather packets generated from the same source and intended to the same destination and launch complex attacks such as linear/differential cryptanalysis. (3) When multiple MIPs are present on the same NoC, they can share information and trace messages. Security of interconnected components was explored under a similar threat model in [90, 169].

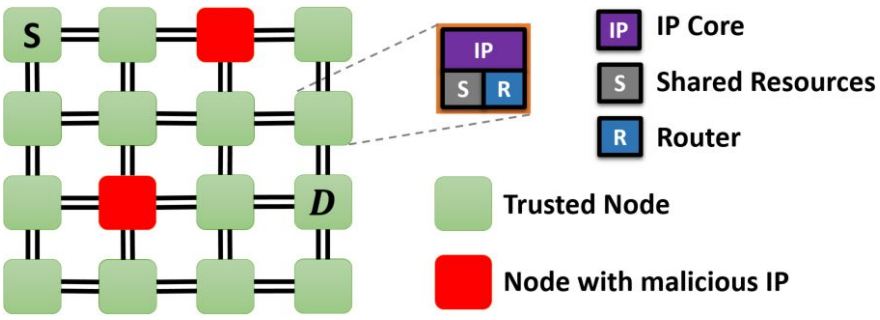


Figure 6-1. Overview of a typical SoC architecture with IPs integrated in a Mesh NoC.

It is not feasible to utilize traditional security methods (encryption, authentication, etc.) in resource-constrained embedded devices. Previous work on lightweight encryption proposed smaller block and key sizes, less rounds of encryption and other hardware optimizations [97]. Irrespective of the optimizations, these methods still have complex computations that take several cycles. In this chapter, I propose a "Lightweight Encryption and Anonymous Routing

protocol for NoCs" (LEARN) that requires only few addition and multiplication operations for encryption. I am able to eliminate the traditional encryption method consisting of ciphers and keys entirely by using the secret sharing approach proposed by Shamir [170] without compromising the security guarantees. Furthermore, my framework supports anonymous routing such that an intermediate node can neither detect the origin nor the destination of a packet.

The remainder of this chapter is organized as follows. Section 6.1 introduces some concepts used in this chapter. Section 6.2 motivates the need for my work. Section 6.3 describes the lightweight encryption and anonymous routing protocol. Section 6.4 presents the experimental results. Section 6.5 discusses possible further enhancements to my approach. Finally, Section 6.6 summarizes the chapter.

6.1 Background

This section introduces some of the key concepts used in our proposed framework.

6.1.1 Secret Sharing with Polynomial Interpolation

Shamir's secret sharing [170] is based on a property of "Lagrange polynomials" known as the (k, n) threshold. It specifies that a certain secret M can be broken into n parts and M can only be recovered if at least k $(k \leq n)$ parts are retrieved. The knowledge of less than k parts leave M completely unknown. Lagrange polynomials meet this property with $k = n$. A Lagrange polynomial is comprised of some k points (x_0, y_0), ..., (x_{k-1}, y_{k-1}) where $x_i \neq x_j$ $(0 \leq i, j \leq k-1)$. A unique polynomial of degree $k-1$ can be calculated from these points:

$$L(x) = \sum_{j=0}^{k-1} l_j(x) \cdot y_j, \tag{6-1}$$

where

$$l_j(x) = \prod_{i=0, i \neq j}^{k-1} \frac{x - x_i}{x_j - x_i} \tag{6-2}$$

Any attempt to reconstruct the polynomial with less than k or incorrect points will give the incorrect polynomial with the wrong coefficients and/or wrong degree.

$L(x)$ forms the interpolated Lagrange polynomial, and $l_j(x)$ is the Lagrange basis polynomial. In order to a share a secret using this method, a random polynomial of degree $k-1$ is chosen. It takes the form of $L(x) = a_0 + a_1x + a_2x^2 + ... + a_{k-1}x^{k-1}$. The shared secret M should be set as $a_0 = M$, and all the other coefficients are chosen randomly. Then a simple calculation at $x = 0$ would yield the secret ($M = L(0)$). In this case, k points on the curve are chosen at random and distributed together with their respective $l_j(0)$ values - the Lagrangian coefficients. To retrieve M, all the parties should share their portions of the secrets. Once all of the k points and $l_j(0)$ coefficients are combined, then the secret can be computed as:

$$M = \sum_{j=0}^{k-1} l_j(0) \cdot y_j, \tag{6-3}$$

This method makes it easier to compute M without having to recalculate each $l_j(x)$.

6.1.2 Anonymous Communication using Onion Routing

Onion routing is widely used in the domain of computer networks when routing has to be done while keeping the sender anonymous. Each message is encrypted several times (layers of encryption) analogous to layers of an onion. Each intermediate router from source to destination (called onion routers) "peels" a single layer of encryption revealing the next hop. The final layer is decrypted and message is read at the destination. The identity of the sender is preserved since each intermediate router only knows the preceding and the following routers. The overhead of onion routing comes from the fact that the sender has to do several rounds of encryption before sending the packet to the network and each intermediate router has to do a decryption before forwarding it to the next hop. While this can be done in computer networks, adopting this in resource-constrained NoCs leads to unacceptable performance overhead as illustrated in Section 6.2.

6.2 Motivation

Security and performance is always a trade-off in resource-constrained systems. While computer networks with potentially unlimited resources can accommodate very strong security techniques such as AES encryption and onion routing, utilizing them in resource-constrained

NoCs can lead to unacceptable overhead. To evaluate this impact, we ran FFT, RADIX (RDX), FMM and LU benchmarks from the SPLASH-2 benchmark suite [142] on an 8×8 Mesh NoC-based SoC with 64 IPs using the gem5 simulator [18] considering three scenarios:

- **No-Security:** NoC does not implement encryption or anonymous routing.
- **Enc-only:** NoC secures data by encrypting before sending into the network. However, it does not support anonymous routing.
- **Enc-and-AR:** Data encryption as well as anonymous routing achieved by onion routing.

We assumed a 12-cycle delay for encryption/decryption when simulating Enc-only and Enc-and-AR according to the evaluations in [98]. More details about the experimental setup is given in Section 6.4.1. Results are shown in Figure 6-2. The values are normalized to the scenario that consumes the most time. Enc-only shows 42% (40% on average) increase in NoC delay (total NoC traversal delay for all packets) and 9% (7% on average) increase in execution time compared to the No-Security implementation. Enc-and-AR gives worse results with 83% (81% on average) increase in NoC delay leading to a 41% (33% on average) increase in execution time when compared with No-Security. In other words, Enc-and-AR leads to approximately 1.5X performance degradation. When security is considered, No-Security leaves the data totally vulnerable to attackers, Enc-only secures the data by encryption and Enc-and-AR provides an additional layer of security with anonymous routing. The overhead of Enc-only is caused by the complex mathematical operations, and the number of cycles required to encrypt each packet. Onion routing used in Enc-and-AR aggravates this by requiring several rounds of encryption before injecting the packet into the network as well as decryption at each hop (router). Added security has less impact on execution time compared to NoC delay since execution time also includes the time for instruction execution and memory operations in addition to NoC delay. In many embedded systems, it would be unacceptable to have security at the cost of 1.5X performance degradation. It would be ideal if the security provided by Enc-and-AR can be achieved while maintaining performance comparable to No-security. Our

approach tries to achieve this goal by introducing a lightweight encryption and anonymous routing protocol as described in the next section.

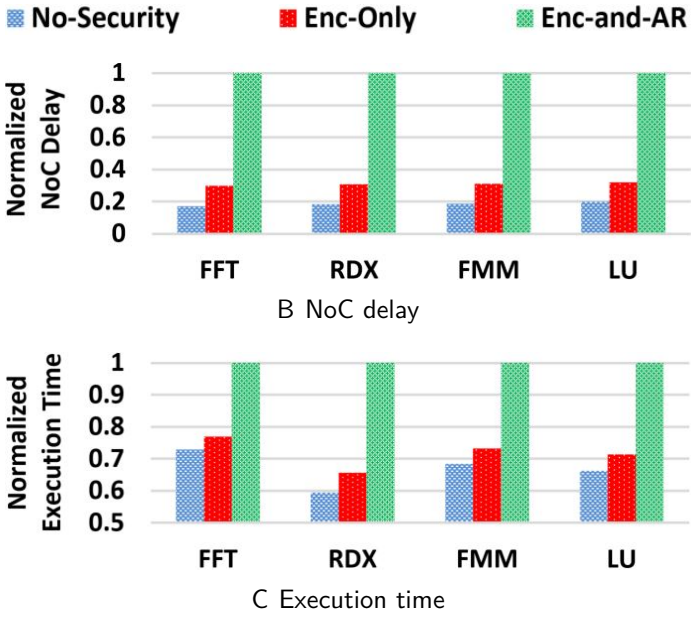


Figure 6-2. NoC delay and execution time comparison across different levels of security.

6.3 Lightweight Encryption and Anonymous Routing Protocol

This section describes our proposed approach - Lightweight Encryption and Anonymous Routing protocol for NoCs (LEARN). By utilizing secret sharing based on polynomial interpolation [170], LEARN negates the need for complex cryptographic operations to encrypt messages. A forwarding node would only have to compute the low overhead addition and multiplication operations to hide the contents of the message. As the message passes through the forwarding path, its appearance is changed at each node, which makes the message's content and route safe from eavesdropping attackers as well as internal ones. The following sections describe our approach in detail. First, we provide an overview of our framework in Section 6.3.1. Next, Section 6.3.2 and Section 6.3.3 describe the two major components of our proposed routing protocol (route discovery and data transfer). Finally, Section 6.3.4 outlines how to efficiently manage relevant parameters during anonymous routing.

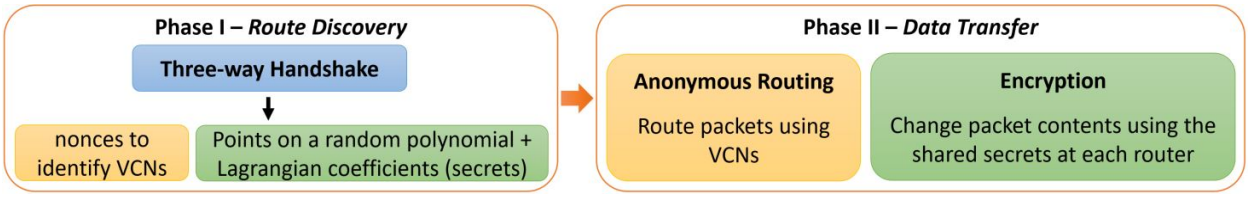


Figure 6-3. Overview of our proposed framework (LEARN)

6.3.1 Overview

LEARN has two main phases as shown in Figure 6-3. When an IP wants to communicate with another IP, it first completes the "Route Discovery" phase. The route discovery phase sends a packet and discovers the route, distributes the parameters among participants. Then the "Data Transfer" phase transfers the message securely and anonymously. The route discovery phase includes a three-way handshake between the sender and the destination nodes. The handshake uses 3 out of the 4 main types of packets sent over the network with the fourth type being used in the second phase. The four main packet types are:

1. RI (Route Initiate) - flooded packet from sender S to destination D to initialize the conversation.
2. RA (Route Accept) - packet sent from D to accept new connection with S.
3. RC (Route Confirmation) - sent from S to distribute configuration parameters with intermediate nodes.
4. DT (Data) - the data packet from S to D that is routed anonymously through the NoC.

Algorithm 1 outlines the major steps of LEARN. During the three-way handshake, a route between S and D is discovered. Each router along the routing path is assigned with few parameters that are used when transferring data - (i) random nonces to represent preceding and following routers (line 2), and (ii) a point in a random polynomial together with its Lagrangian coefficient (line 3). This marks the end of the first phase which enables the second phase - "Data Transfer". The second phase uses the parameters assigned to each router to forward the original message through the route anonymously while hiding its contents. Anonymous routing is achieved by using the random nonces which act as virtual

circuit numbers (VCN). When transferring data packets, the intermediate routers will only see the VCNs corresponding to the preceding router and the following router which reveals no information about the source or the destination (line 7). Encryption is achieved using the points in the random polynomial and their corresponding Lagrangian coefficients. Each router along the path changes the contents of the message in such a way that only the final destination will be able to retrieve the entire message (line 6).

Algorithm 8 Major steps of LEARN

```
   Phase I - Route Discovery
1: for all r ∈ routers do
2:     r ← v_i, v_j                          ▷ nonces to identify VCNs
3:     r ← (x_k, y_k, b_k)                   ▷ a point in a random polynomial
4: end for
   Phase II - Data Transfer
5: while r ≠ destination do
6:     m ← F(m, (x_k, y_k, b_k))             ▷ modify message
7:     r ← getNextHop(v_i, v_j)              ▷ get next hop
8: end while
```

LEARN improves performance by replacing complex cryptographic operations with addition/multiplication operations that consume significantly less time during the data transfer phase. The overhead occurs during the first phase (route discovery) that requires cryptographic operations. However, this is performed only a constant number of times (once per communication session). Since the route discovery phase happens only once in the beginning of a communication session, the cost for route discovery gets amortized over time. This leads to significant performance improvement.

Note that the route discovered at the route discovery stage will remain the same for the lifetime of the task. In case of context switching and/or task migration, the first phase will be repeated before transferring data. Each IP in the SoC that uses the NoC to communicate with other IPs follows the same procedure. The next two sections describe these two phases in detail. A list of notations used to illustrate the idea is listed in Table 6-1. The superscript "i" is used to indicate that the parameter is changed for each packet of a given packet type.

Table 6-1. Notations used to illustrate LEARN

Notation	Description
$OPK_S^{(i)}$	one-time public key (OPK) used by the source to uniquely identify an RA packet
$OSK_S^{(i)}$	private key corresponding to $OPK_S^{(i)}$
ρ	random number generated by the source
PK_D	the global public key of the destination
SK_D	the private key corresponding to PK_D
$TPK_A^{(i)}$	temporary public key of node A
$TSK_A^{(i)}$	the private key corresponding to $TPK_A^{(i)}$
K_{S-A}	symmetric key shared between S and A
υ_A	randomly generated nonce by node A
b_i	Lagrangian coefficient of a given point (x_i, y_i)
$E_K(M)$	a message M encrypted using the key K

6.3.2 Route Discovery

The route discovery phase performs a three-way handshake between the sender S and destination D. This includes broadcasting the first packet - RI from S with the destination D, getting a response (RA) from D acknowledging the reception of RI, and finally, sending RC with the parameters required to implement polynomial interpolation based secret sharing. Figure 6-4 shows an illustrative example of parameters (using only four nodes) shared and stored during the handshake.

The initial route initiate packet (RI) takes the form:

$$\{RI \parallel OPK_S^{(i)} \parallel E_{PK_D}(OPK_S^{(i)} \parallel \rho) \parallel TPK_S^{(i)}\}$$

The first part of the message indicates the type of packet being sent, RI in this case. $OPK_S^{(i)}$ refers to the one-time public key associated with the sender node. This public key together with its corresponding private key $OSK_S^{(i)}$ change with each new conversation or RI. This change allows for a particular conversation to be uniquely identified by these keys, which are saved in its route request table. ρ is a randomly generated number by the sender that is concatenated with the $OPK_S^{(i)}$ and then encrypted with the destination node's public key PK_D as a global trapdoor [171]. Since PK_D is used to encrypt, only the destination is able

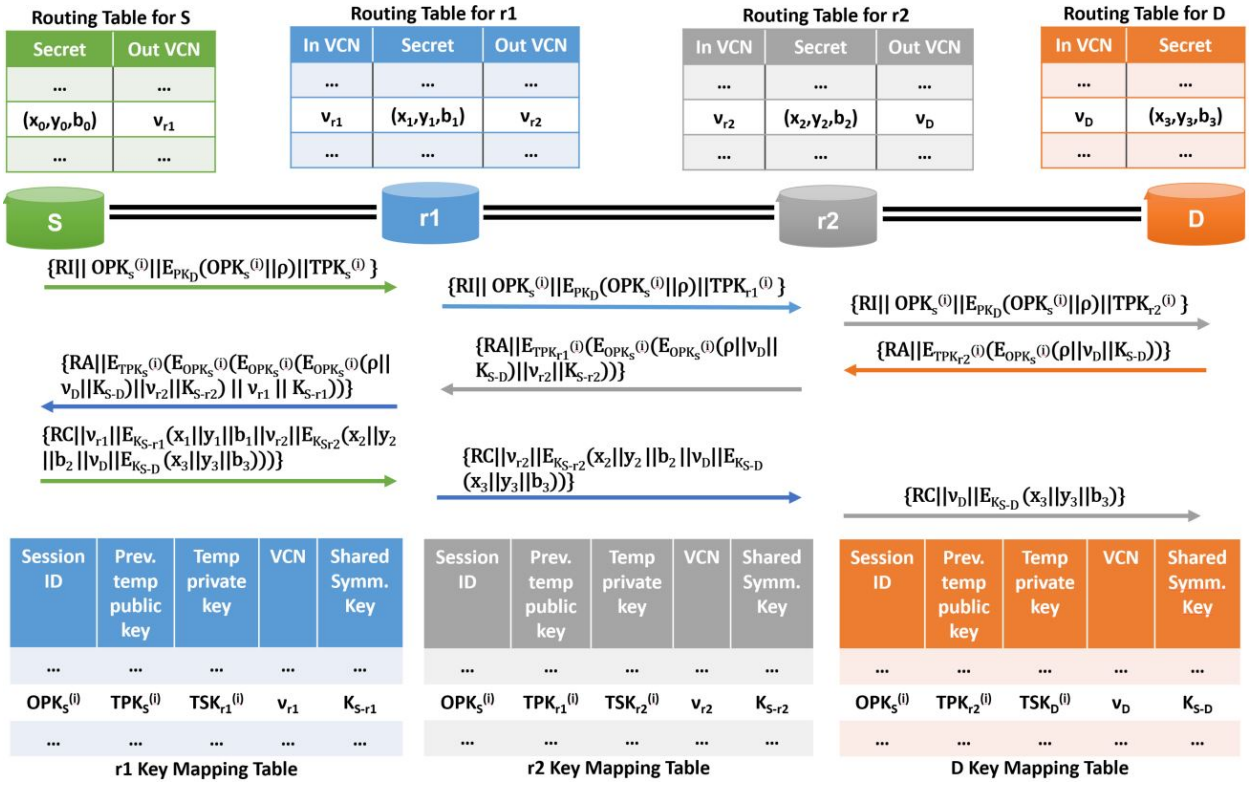


Figure 6-4. Steps of the three-way handshake and the status of parameters at the end of the process.

to open the trapdoor using SK_D. Then the $TPK_S^{(i)}$ is attached to show the temporary key of the forwarding node, which is initially the sender. The temporary keys are also implemented as one-time trapdoors to ensure security.

The next node, $r1$, to receive the RI messages goes through a few basic steps. Firstly, it checks for the $OPK_S^{(i)}$ in its key mapping table, which would indicate a duplicated message. Any duplicates are discarded at this step. Next, $r1$ will attempt to decrypt the message and retrieve ρ. Success would indicate that $r1$ was the intended recipient D. If not, $r1$ replaces $TPK_S^{(i)}$ with its own temporary public key $TPK_{r1}^{(i)}$ and broadcasts:

$$\{RI \parallel OPK_S^{(i)} \parallel E_{PK_D}(OPK_S^{(i)} \parallel \rho) \parallel TPK_{r1}^{(i)}\}$$

$r1$ also logs $OPK_S^{(i)}$ and $TPK_S^{(i)}$ from the received message and $TSK_{r1}^{(i)}$ corresponding to $TPK_{r1}^{(i)}$ in its key mapping table. This information is used later when an RA message is received from D.

D will eventually receive the RI message and will decrypt using SK_D. This will allow D to retrieve $OPK_S^{(i)}$ and ρ from $E_{PK_D}(OPK_S^{(i)} \parallel \rho)$. Then to verify that the RI has not been tampered with, D will compare the plaintext $OPK_S^{(i)}$ and the now decrypted $OPK_S^{(i)}$. If they are different, the RI is simply discarded. Otherwise, D sends a RA (route accept) message:

$$\{RA \parallel E_{TPK_{r2}^{(i)}}(E_{OPK_S^{(i)}}(\rho \parallel \upsilon_D \parallel K_{S-D}))\} \tag{6-4}$$

RA, like RI in the previous message, is there to indicate message type. D generates a random nonce, υ_D, to serve as a VCN and a randomly selected key K_{S-D} to act as a symmetric key between S and D. D stores υ_D and K_{S-D} in its key mapping table. It also makes an entry in its routing table indexed by υ_D, the VCN. The concatenation of ρ, υ_D, and K_{S-D} is then encrypted with the $OPK_S^{(i)}$, so that only S can access that information. Then the message is encrypted again by $TPK_{r2}^{(i)}$, $r2$'s temporary public key, with $r2$ being the node that delivered RI to D.

Once $r2$ receives the RA, it decrypts it using its temporary private key, $TSK_{r2}^{(i)}$, and follows the same steps as D. It generates its own nonce, υ_{r2}, and shared symmetric key, K_{S-r2}, to be shared with S. Both the nonce and symmetric key are then concatenated to the RA message and encrypted by S's public key, $OPK_S^{(i)}$, so that only S can retrieve that data. This adds another layer of encrypted content to the message for S to decrypt using $OSK_S^{(i)}$. Similar to D, $r2$ also stores υ_{r2} and K_{S-r2} in its key mapping table and routing table. It then finds the temporary public key for the previous node in the path from its key mapping table - $TPK_{r1}^{(i)}$ and encrypts the message. The message sent out by $r2$ looks like:

$$\{RA \parallel E_{TPK_{r1}^{(i)}}(E_{OPK_S^{(i)}}(E_{OPK_S^{(i)}}(\rho \parallel \upsilon_D \parallel K_{S-D}) \parallel \upsilon_{r2} \parallel K_{S-r2}))\} \tag{6-5}$$

This process is repeated at each node along the path until the RA packet makes it way back to S. The entire message at that point is encrypted with $TPK_S^{(i)}$, which is stripped away using $TSK_S^{(i)}$. Then S can "peel" each layer of the encrypted message by $OSK_S^{(i)}$ to retrieved all the VCNs, shared symmetric keys, and also, ρ. ρ is used to authenticate that the entire message came from the correct destination and was not changed during the journey.

Once S completes authentication of the received RA packet, it randomly generates $k+1$ points $(x_0, y_0), (x_1, y_1), ..., (x_k, y_k)$ on a k degree polynomial $L(x)$ as shown in Figure 6-5. $k+1$ is the number of nodes in the path from S to D. S then uses these points to calculate the Lagrangian coefficients, $b_0, b_1, ..., b_k$, using:

$$b_j = \prod_{i=0, i \neq j}^{k} \frac{x_i}{x_i - x_j} \tag{6-6}$$

Using the generated data, S constructs a route confirmation (RC) packet:

$$\{RC \parallel \upsilon_{r1} \parallel E_{K_{S-r1}}(x_1 \parallel y_1 \parallel b_1 \parallel \upsilon_{r2} \parallel E_{K_{S-r2}}(x_2 \parallel y_2 \parallel b_2 \parallel \upsilon_D \parallel E_{K_{S-D}}(x_3 \parallel y_3 \parallel b_3)))\} \tag{6-7}$$

Similar to the case in RA and RI, RC in the packet refers to the packet type. The rest of the message is layered much like the previous RA packet. Each layer contains the υ_* for each node concatenated with secret information that is encrypted with the shared key K_{S-*}, where * corresponds to $r1, r2$ or D in our example (Figure 6-4). The (υ_*, K_{S-*}) pair was generated by each node during the RA packet transfer phase and the values were stored in the key mapping tables as well as entries indexed by the VCNs created in the routing table. Therefore, each node can decrypt one layer, store incoming and outgoing VCNs together with the secret, and pass it on to the next node to do the same. For example, $r1$ receiving the packet can observe that the incoming VCN is υ_{r1}. It then decrypts the first layer using the symmetric key K_{S-r1}, that is already stored in the key mapping table, and recovers the secret (x_1, y_1, b_1) as well as the outgoing VCN υ_{r2}. It then updates the entry indexed by υ_{r1} in its routing table with the

secret tuple and the outgoing VCN. Similarly, each router from S to D can build its routing table.

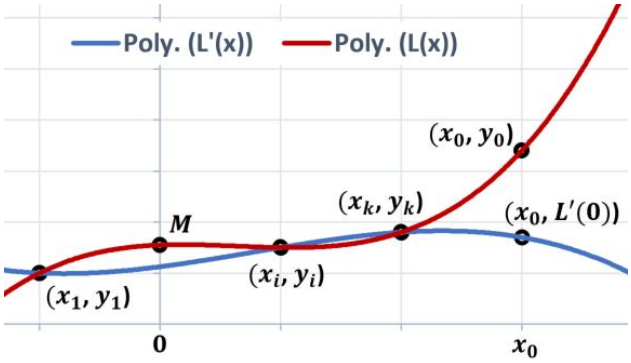


Figure 6-5. Lagrangian polynomials $L(x)$ and $L'(x)$ together with the selected points.

6.3.3 Data Transfer

The path set up can now be used to transfer messages from S to D anonymously. For each conversation, $k+1$ points were generated on a random curve $L(x)$ chosen by S. During the last step of the route discovery phase (RC packet), S kept (x_0, y_0, b_0) for itself and distributed each node on the discovered path a different point, (x_i, y_i) (where $1 \leq i \leq k$), with the corresponding Lagrangian coefficient b_i. If S wants to send the message M to D, S has to generate a new k degree polynomial $L'(x)$ which is defined by the k points distributed to nodes except for (x_0, y_0), i.e., points (x_i, y_i) where $(1 \leq i \leq k)$ and a new point $(0, M)$. This makes $L'(0) = M$ with M as the secret message, according to the explanation in Section 6.1.1. S then changes its own point (x_0, y_0) to (x_0, y_0') where $y_0' = L'(x_0)$, making sure the point retained by S is also on the curve L'(x) as shown in Figure 6-5. It is important to note that every coefficient b_i, and every point distributed to nodes along the route remain unchanged. For this scenario, considering Equation 6-3, we can derive:

$$M = y_0' b_0 + \sum_{i=1}^{k} b_i \cdot y_i \tag{6-8}$$

To transfer a secret message, M, from S to D anonymously, S constructs data transfer (DT) packet with the form:

$$\{DT \parallel \upsilon_{r1} \parallel y_0' b_0\} \tag{6-9}$$

DT, like every other packet, has an indicator of packet type at the front of the packet - DT. υ_{r1} is the VCN of the next node. $y_0' b_0$ is the portion of the message M that is constructed by S. Once $r1$ receives the DT packet, it adds its own portion of the message, $y_1 b_1$, to $y_0' b_0$. It also uses its routing table to find the VCN of the next node and replaces the incoming VCN by the outgoing VCN in the DT packet. Therefore, the message received by $r2$ has the form:

$$\{DT \parallel \upsilon_{r2} \parallel y_0' b_0 + y_1 b_1\} \tag{6-10}$$

Next, $r2$ repeats the same process and forwards the packet:

$$\{DT \parallel \upsilon_{D} \parallel y_0' b_0 + y_1 b_1 + y_2 b_2\} \tag{6-11}$$

to D. Eventually, D will be able to retrieve the secret message, $M = y_0' b_0 + y_1 b_1 + y_2 b_2 + y_3 b_3$ by adding the last portion $y_3 b_3$ constructed using the part of the secret D shared. Using this method, neither an intermediate node nor an eavesdropper in the middle will be able to see the full message since the message M is incomplete at every intermediate node and is fully constructed only at the destination D.

6.3.4 Parameter Management

To ensure the efficient implementation of LEARN, an important aspect needs to be addressed - the generation and management of keys and nonces. Many previous studies have addressed this problem in several ways. One such example is the work done by Lebiednik et al. [163]. In their work, a separate IP called the key distribution center (KDC) handles the distribution of keys. Each node in the network negotiates a new key with the KDC using a pre-shared portion of memory that is known by only the KDC and the corresponding node. The node then communicates with the KDC using this unique key whenever it wants to obtain a new key. The KDC can then allocate keys depending on whether it is symmetric/asymmetric

encryption, and inform other nodes as required. The key request can delay the communication. But once keys are established, it can be used for many times depending on the length of the encrypted packet before refreshing to prevent linear distinguishing attacks. In our approach, the keys are only used during the route discovery phase, and the discovered route will remain the same for the lifetime of the task unless context switching or task migration happens. Therefore, key refreshing will rarely happen and the cost for the initial key agreement as well as the route discovery phase will be amortized.

6.4 Experiments

This section presents results to evaluate the efficiency of our approach (LEARN). We first describe the experimental setup. Next, we compare the performance of LEARN with traditional encryption and anonymous routing protocols introduced in Section 6.2. Finally, we discuss the area overhead and security aspects of LEARN.

6.4.1 Experimental Setup

Extending the results presented in Figure 6-2, LEARN was tested on an 8×8 Mesh NoC-based SoC with 64 IPs using the gem5 cycle-accurate full-system simulator [18]. The NoC was built using the "GARNET2.0" model that is integrated with gem5 [139, 153]. The route discovery phase of our approach relies on the RI, RA, and RC packets traversing along the same path to distribute the keys and nonces. Therefore, the topology requires bidirectional links connecting the routers. While we experimented on a Mesh NoC, there are many other NoC topologies that can adopt LEARN where all links are bidirectional as evidenced by academic research [139] as well as commercial SoCs [4].

Each encryption/decryption is modelled with a 12-cycle delay [98]. Computations related to generating the random polynomial and deciding the k points is assumed to consume 200 cycles. To accurately capture congestion, the NoC was modeled with 3-stage (buffer write, route compute + virtual channel allocation + switch allocation, and link traversal) pipelined routers with wormhole switching and 4 virtual channel buffers at each input port. Each link

was assumed to consume one cycle to transmit packets between neighboring routers. The delays were chosen to be consistent with the delays of components in the gem5 simulator.

We used the default gem5 and Garnet2.0 configurations for packet sizes, virtual channels and flow control. In addition to the four main types of packets described in Section 6.3.1, the DT packets can be further divided into two categories as control and data packets. For example, in case of a cache miss, a memory request packet (control packet) is injected into the NoC and the memory response packet (data packet) consists of the data block from the memory. The address portion of a control DT packet consists of 64 bits. In the data DT packet, in addition to the 64-bit address, 512 bits are reserved for the data block. A credit-based, virtual channel flow control was used in the architecture. Each data VC and control VC was allocated buffer depths of 4 and 1, respectively.

LEARN was tested using 6 real benchmarks (FFT, RADIX, FMM, LU, OCEAN, CHOLESKY) from the SPLASH-2 benchmark suite and 6 synthetic traffic patterns: uniform random (URD), tornado (TRD), bit complement (BCT), bit reverse (BRS), bit rotation (BRT), transpose (TPS). Out of the 64 cores, 16 IPs were chosen at random and each one of them instantiated an instance of the task. The packets injected into the NoC when running the real benchmarks were the memory requests/responses. We used 8 memory controllers that provide the interface to off-chip memory which were placed on the boundary of the SoC. This memory controller placement adheres to commercial SoC architectures such as Intel's Knights Landing (KNL) [4]. An example to illustrate the IP placement is shown in Figure 6-6.

When running real benchmarks, the packets get injected to the NoC when there are private cache misses and the frequency of that happening depends on the characteristics of the benchmark. When running synthetic traffic patterns, packets were injected into the NoC at the rate of 0.01 packets/node/cycle. For synthetic traffic patterns, the destinations of injected packets were selected based on the traffic pattern. For example, uniform random selected the destination from the remaining IPs with equal probability whereas bit complement, complemented the bits of the source address to get the destination address, etc.. The choices

made in the experimental setup were motivated by the architecture/threat model and the behavior of the gem5 simulator. However, LEARN can be used with any other NoC topology and task/memory controller placement.

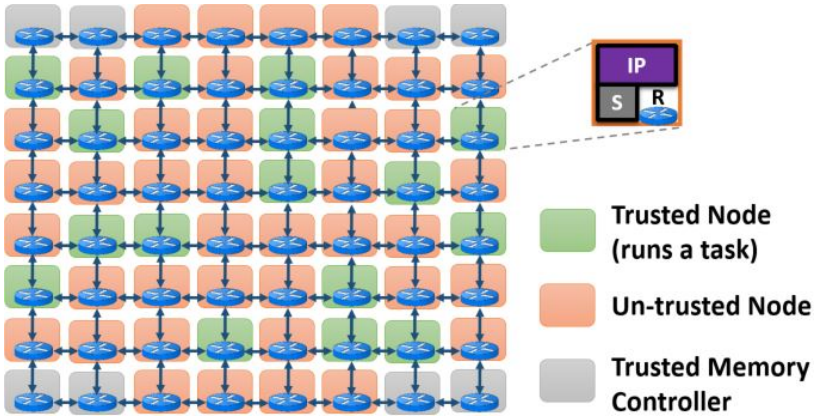


Figure 6-6. 8×8 Mesh NoC architecture used to generate results including trusted nodes running the tasks and communicating with memory controllers while untrusted nodes can potentially have malicious IPs.

6.4.2 Performance Evaluation

Figure 6-7 shows performance improvement LEARN can gain when running real benchmarks. We compare the results from LEARN against the three scenarios considered in Figure 6-2. Compared to the No-Security scenario, LEARN consumes 30% more time (28% on average) for NoC traversals (NoC delay) and that results in only 5% (4% on average) increase in total execution time. Compared to Enc-and-AR which also implements encryption and anonymous routing, LEARN improves NoC delay by 76% (74% on average) and total execution time by 37% (30% on average). We can observe from the results that the performance of LEARN is even better than Enc-Only, which provides encryption without anonymous routing. Overall, LEARN can provide encryption and anonymous routing consuming only 4% performance overhead compared to the NoC that does not implement any security features.

The same experiments were carried out using synthetic traffic traces, and results are shown in Figure 6-8. Since synthetic traffic patterns only simulate NoC traffic and do not include instruction execution and memory operations, only NoC delay is shown in the figure. Compared to Enc-and-AR, LEARN improves performance by 76% (72% on average).

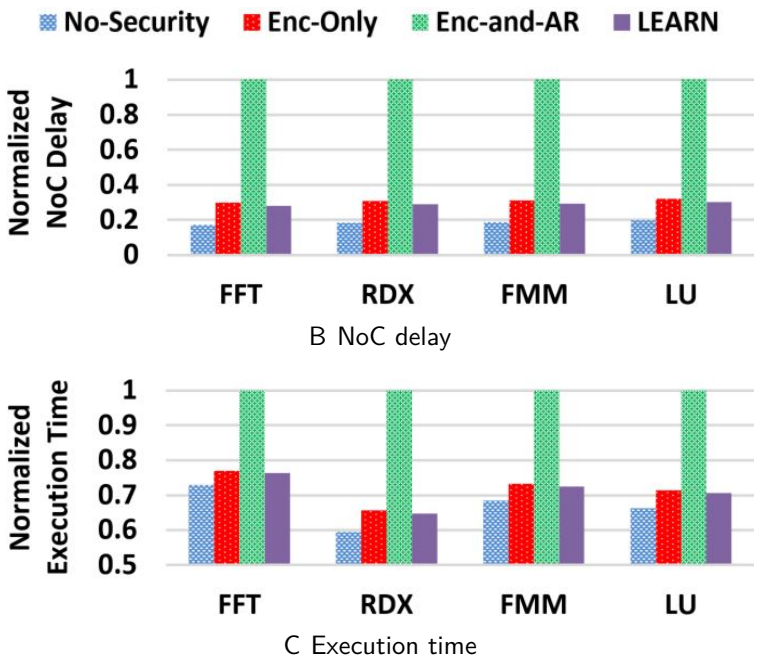


Figure 6-7. NoC delay and execution time comparison across different security levels using real benchmarks.

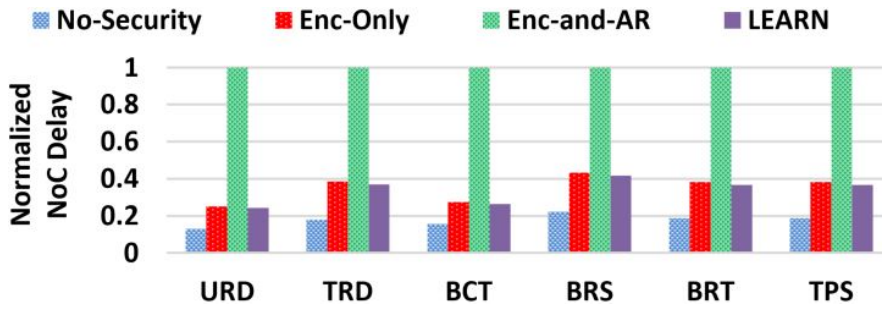


Figure 6-8. NoC delay comparison across different levels of security when running synthetic traffic patterns.

The performance improvement of LEARN comes from the fact that once the path has been set up for the communication between any two IPs, the overhead caused to securely communicate between the two IPs (data transfer phase) while preserving route anonymity is much less. The notable overhead occurs at the route discovery phase due to complex cryptographic operations. The intermediate nodes encrypt/decrypt packets to exchange parameters securely. Yet, these complex cryptographic operations are performed only a constant number of times. Majority of the work is done at the source which selects points to

be distributed among intermediate nodes after constructing a curve, calculates the Lagrangian coefficients of the selected points, and performs several rounds of encryption/decryption during the three-way handshake. Once the routing path is setup, packets can be forwarded from one router to the other by a simple table look-up. No per-hop encryption and decryption is required to preserve anonymity. The security of a message is ensured by changing the original message at each node using a few addition and multiplication operations which incur significantly fewer extra delays. Since the route discovery phase happens only once during the lifetime of a task unless context switching and/or task migration happens, and there is only a limited number of communications going on between IPs in an SoC, the cost during the route discovery phase gets amortized over time. When running real benchmarks, we observed a packet ratio of 1:1:1:6325 on average for $RI : RA : RC : DT$, respectively. For synthetic traffic patterns, the same ratio was observed to be 1:1:1:1964. This leads to a significant performance improvement compared to the traditional methods of encryption and anonymous routing.

6.4.3 Area Overhead of the Key Mapping Table

The key mapping table is an extra table compared to No-Security approach used to implement our anonymous routing protocol. The key mapping table adds a row for each session. Therefore, the size of the key mapping table is linearly proportional to the number of sessions. If at design time, it is decided to have a fixed size for the key mapping table, it is possible for the key mapping table at a router to be full after adding sessions, and in that case, new sessions cannot be added through that router. Therefore, the size has to be decided according to the communication requirements.

The maximum number of communication pairs in an 8×8 Mesh is $\binom{64}{2} \times 2 = 4032$ (assuming two-way communication between any pair out of the 64 nodes). Depending on the address mapping, only some node pairs (out of all the possible node pairs) communicate. Our simulations consisted of 256 unique node pairs. In the worst case, if we assume each communication session has one common router, the key mapping table should be $256 \times row\_size$ big. If each entry in the key mapping table is 128 bits, the total size becomes 20kB.

However, in reality, not all communication sessions overlap. It is also important to note that except for the Session ID in the key mapping table, the other entries can be overwritten once route discovery phase is complete. Therefore, it is possible to allocate a fixed size key mapping table during design time and yet keep the area overhead low.

6.4.4 Security Analysis

In this section, we discuss the security and privacy of messages transferred on the NoC using LEARN.

Security of messages: The security of messages is preserved by the (k, n) threshold property of Lagrangian polynomials discussed in Section 6.1.1. Therefore, unless an intermediate node can gather all points distributed among the routers in the routing path together with their Lagrangian coefficients, the original message cannot be recovered. Our threat model states that the source and destination are trusted IPs, and also, only some of the IPs are untrusted. Therefore, all routers along the routing path will never be compromised at the same time. The threat comes from malicious IPs sitting on the routing path and eavesdropping to extract security critical information. LEARN ensures that intermediate nodes that can be malicious, cannot recover the original message during the data transfer phase by changing the message at each hop. The complete message can only be constructed at the destination. During route discovery phase, each packet is encrypted such that only the intended recipient can decrypt it. The key and nonce exchange is also secured according to the mechanism proposed in Section 6.3.4. Therefore, LEARN ensures that no intermediate M3PIP can gather enough data to recover the plaintext from messages.

Anonymity of nodes in the network: LEARN preserves the anonymity of nodes in the network during all of its operational phases. When the source sends the initial RI packet to initiate the three-way handshake, it doesn't use the identity of the destination. Instead, the source uses the global public key of the destination (PK_D) and sends a broadcast message on the network. When the RI packet propagates through the network, each intermediate node saves a temporary public key of its predecessor. This temporary public key is then

used to encrypt data when propagating the RA packet so that unicast messages can be sent to preceding nodes without using their identities. Random nonces and symmetric keys are assigned to each node during the RA packet propagation which in turn is used by the RC packet to distribute points and Lagrangian coefficients to each node. Data transfer is done by looking up the routing table that consists of the nonces representing incoming and outgoing VCNs. Therefore, the identities of the nodes are not revealed at any point during communication.

Anonymity of routes taken by packets: In addition to preserving the anonymity of nodes, LEARN also ensures that the path taken by each packet is anonymous. Anonymity of the routing path is ensured by two main characteristics. (i) The message is changed at each hop. Therefore, even if there are two M3PIPs on the same routing path, information exchange among the two M3PIPs will not help in identifying whether the same message was passed through both of them. The same message appears as two completely different messages when passing through two different nodes. (ii) The routing table contains only the preceding and following nodes along the routing path. An M3PIP compromising a router will only reveal information about the next hop and the preceding hop. Therefore, the routing paths of all packets remain anonymous.

6.5 Discussion

In this section, we discuss possible alternatives to our design choices from both design overhead and security perspectives. Most importantly, we discuss security solutions to defend against attacks when an attacker is aware of our security mechanism.

6.5.1 Feasibility of a Separate Service NoC

Modern SoCs use multiple physical NoCs to carry different types of packets [4, 17]. The KNL architecture used in Intel Xeon-Phi processor family uses four parallel NoCs [4]. The Tilera TILE64 architecture uses five Mesh NoCs, each used to transfer packets belonging to a certain packet type such as main memory, communication with I/O devices, and user-level scalar operand and stream communication between tiles [17]. The decision to implement

separate physical NoCs is dependent on the performance versus area trade-off. If only one physical NoC is used to carry all types of packets, the packets must contain header fields such as RI, RA, RC, DT to distinguish between different types. The buffer space is shared between different packet types. The SoC performance can deteriorate significantly due to these factors coupled with the increasing number of IPs in an SoC. On the other hand, contrary to intuition, due to the advancements in chip fabrication processes, additional wiring between nodes incur minimal overhead as long as the wires stay on-chip. Furthermore, when wiring bandwidth and on-chip buffer capacity is compared, the more expensive and scarce commodity is the on-chip buffer area. If different packet types are carried on NoC using virtual channels and buffer space is shared [113], the increased buffer spaces and logic complexity to implement virtual channels becomes comparable to another physical NoC. A comprehensive analysis of having virtual channels versus several physical NoCs is given in [172].

It is possible to use two physical NoCs - one for data (DT) packet transfers and the other to carry packets related to the handshake (RI, RA, RC). However, in our setup, the potential performance improvement from a separate service NoC was not enough to justify the area and power overhead. We envision that our security mechanism to be a part of a suite of NoC security countermeasures that can address other threat models such as denial-of-service, buffer overflow, etc. The service NoC will be effective in such a scenario where more service type packets (e.g., DoS attack detection related packets [85]) are transferred through the NoC.

6.5.2 Obfuscating the Added Secret

An attacker who is aware of our security mechanism can try to infer a communication path by observing the incoming and outgoing packets at a router.

Since each intermediate node adds a constant value ($y_i b_i$) to the received DT packet, the difference between incoming and outgoing DT packets at each node will be the same for a given virtual circuit. For this attack to take place, two consecutive routers have to be infected by attackers and they have to collaborate. Alternatively, a Trojan in a router has to have the ability to observe both incoming and outgoing packets at the router. While these

are strong security assumptions, it is important to address this loophole. In this Section, we propose a countermeasure against such an attack. Even in the presence of such an attack, the secret message cannot be inferred since the complete message is only constructed at the destination and according to our threat model, we assume that the source and destination IPs are trustworthy.

This can be solved by changing the shared secret at each node for each message. However, generating and distributing secrets for each node per message can incur significant performance overhead. Therefore, we propose a solution based on each node updating its own secret. According to Equation 6-6, to derive a new Lagrangian coefficient b_i, the x coordinates should be changed. The source can easily do it for each message by changing both x_0 and y_0 when a new message needs to be sent. In other words, rather than changing the point (x_0,y_0) to (x_0,y_0'), it should be changed to (x_0',y_0'). However, the new x_0' now has to be sent to each intermediate node for them to be able to calculate the new secrets using:

$$b_j' = b_j \cdot \frac{x_0'}{x_0' - x_j} \cdot \frac{x_0 - x_j}{x_0} \tag{6-12}$$

We want to avoid such communications for performance as well as security concerns. An alternative is to use a function $\mathcal{F}(x_0, \delta)$ that can derive the next x-coordinate starting from the initial x_0.

$$x_0' = \mathcal{F}(x_0, \delta) \tag{6-13}$$

where $\mathcal{F}(x_0, \delta)$ can be a simple incremental function such as $\mathcal{F}(x_0, \delta) = x + \delta$. δ can be a constant. To increase security, δ can be picked using a psuedo-random number generator (PRNG) seeded with the same value at each iteration. Using such a method will change the shared secrets at each iteration and that will remove correlation between incoming and outgoing packets at a node.

6.5.3 Hiding the Number of Layers

Another potential vulnerability introduced by our approach is that attackers who are aware of our protocol, can infer how far they are from the source and destination based on the size of

the RA and RC packets. However, except for the corner case where the source/destination are at the edge of a certain topology, there can be more than one choice for potential source/destination candidates. In our experiments, we use the Mesh topology in which from the perspective of any node, there can be more than one node that is at distance d away. However, the attacker can reduce the set of possible source/destination candidates for a given communication stream. Therefore, depending on the security requirements, this vulnerability can be addressed using the mechanism proposed in this section.

After receiving the RI Packet, when the RA packet is initiated at D, D generates m $\langle$nonce, key$\rangle$ pairs ($\langle v_D^1, K_{S-D}^1\rangle, \langle v_D^2, K_{S-D}^2\rangle, ..., \langle v_D^m, K_{S-D}^m\rangle$) and adds m layers to the packet. As a result, the RA packet sent from D to $r2$ takes the form:

$$\{RA \parallel E_{TPK_{r2}^{(i)}}(E_{OPK_S^{(i)}}(...E_{OPK_S^{(i)}}(E_{OPK_S^{(i)}}(\rho \parallel v_D^1 \parallel K_{S-D}^1) \parallel v_D^2 \parallel K_{S-D}^2)... \parallel v_D^m \parallel K_{S-D}^m))\}$$

D stores the $\langle$nonce, key$\rangle$ pairs in its key mapping table. When S receives the RA packet, S cannot distinguish whether the m pairs were generated from multiple nodes or one node. Therefore, when the RC packet is generated at S, instead of generating $k+1$ points (corresponding to the number of nodes in the path), the number of generated points depends on the number of $\langle$nonce, key$\rangle$ pairs received. During RC packet transfer, each intermediate node along the routing path stores points (VCNs and secrets) corresponding to the nonces stored in the key mapping table. As a result, nodes can receive multiple secrets which can then be used during the data transfer phase. Depending on the required level of security, m can vary and also, each intermediate node can add multiple layers to the RA packet.

This method hides the correlation between the number of nodes and the length of the routing path, and therefore, eliminates the said vulnerability. However, this increases the performance penalty. Therefore, the number of extra layers should be decided after considering the security-performance trade-off.

6.6 Summary

Security and privacy are paramount considerations during electronic communication. Unfortunately, we cannot implement well-known security solutions from computer networks on resource constrained SoCs in embedded systems and IoT devices. Specifically, these security solutions can lead to unacceptable performance overhead. In this chapter, I proposed a lightweight encryption and anonymous routing protocol that addresses the classical trade-off between security and performance. My approach uses a secret sharing based mechanism to securely transfer data in an NoC based SoC. Packets are changed at each hop and the complete packet is constructed only at the destination. Therefore, an eavesdropper along the routing path is unable to recover the plaintext of the intended message. Data is secured using only a few addition and multiplication operations which allows us to eliminate complex cryptographic operations that cause significant performance overhead. My anonymous routing protocol achieves superior performance compared to traditional anonymous routing methods such as onion routing by eliminating the need for per-hop decryption. Experimental results demonstrated that implementation of existing security solutions on NoC can introduce significant (1.5X) performance degradation, whereas my approach can provide the desired security requirements with minor (4%) impact on performance.

CHAPTER 7
RUNTIME DETECTION AND LOCALIZATION OF DOS ATTACKS

With the increased popularity of IoT and embedded devices, SoCs are used in well-defined and time-critical systems. These systems can be one of the main targets of Denial-of-Service (DoS) attacks due to their real-time requirements with task deadlines. Early detection of DoS attacks in such systems is crucial as increased latency in packet transmission can lead to real-time violations and other consequences. Importance of NoC security has led to many prior efforts to mitigate DoS attacks in an NoC such as traffic monitoring [101, 106] and formal verification-based methods [104]. Other real-time traffic monitoring mechanisms have also been discussed in non-NoC domains [173]. However, none of the existing techniques explored a lightweight and real-time mechanism to detect potential DoS attacks as well as localize the malicious source(s) in an NoC setup.

As outlined in Section 7.1, it is a major challenge to detect and localize a malicious IP in real-time. The problem is more challenging in the presence of multiple malicious IPs, and it gets further aggravated when multiple attackers help each other to mount the Distributed DoS (DDoS) attack. In this chapter, I propose an efficient method that focuses on detecting changes in the communication behavior in real-time to identify DDoS attacks. It is a common practice to encrypt critical data in an NoC packet and leave only few fields as plain text [93]. This motivated my approach to monitor communication patterns without analyzing the encrypted contents of the packets. In this chapter, I propose a real-time and lightweight DDoS attack detection and localization technique for NoC-based SoCs.

Major contributions of this chapter can be summarized as follows;

1. I propose a real-time and lightweight DDoS attack detection technique for NoC-based SoCs. The routers store statically profiled traffic behavior and monitor packets in the NoC to detect any violations in real-time.
2. We have developed a lightweight approach to localize the M3PIP(s) in real-time once an attack is detected.

3. We have evaluated the effectiveness of our approach against different NoC topologies using both real benchmarks and synthetic traffic patterns considering DoS attacks originating from a single malicious IP as well as from multiple malicious IPs.

4. To further evaluate the applicability of our approach, we use an architecture model similar to one of the commercially available SoCs - Intel's KNL architecture [174].

The remainder of the chapter is organized as follows. Section 7.1 discusses the threat model and communication model used in my framework. Section 7.2 describes the real-time attack detection and localization methodology. Section 7.3 presents the experimental results. Section 7.4 presents the case study using KNL. Section 7.5 discusses the applicability and limitations of the proposed approach. Finally, Section 7.6 summarizes the chapter.

7.1 System and Threat Models

7.1.1 Threat Model

Previous works have explored two main types of DDoS attacks on NoCs [175] - (i) MIPs flooding the network with useless packets frequently to waste bandwidth and cause a higher communication latency causing saturation, and (ii) draining attack which makes the system execute high-power tasks and causes fast draining of battery. An illustrative example is shown in Figure 7-1 to demonstrate the first type of DDoS attack. As a result of the injected traffic from the malicious IPs to the victim IP (this can be a critical NoC component such as a memory controller), routers in that area of the NoC get congested and responses experience severe delays.

A practical example of a draining attack was shown in [176]. A malware known as a worm spread through Bluetooth and multimedia messaging services (MMS) and infected the recipient's mobile phone. The code is crafted in such a way that it sends continuous requests to the Bluetooth module for paging and to scan for devices. Power consumption in the infected phone was increased up to 500% compared to the idle state causing significant degradation of battery lifetime. There are instances of draining attacks where even though the computation overhead increases, the communication traffic does not increase. Such attacks cannot be

detected using a security mechanism implemented at the NoC, and therefore, are beyond the scope of this chapter.

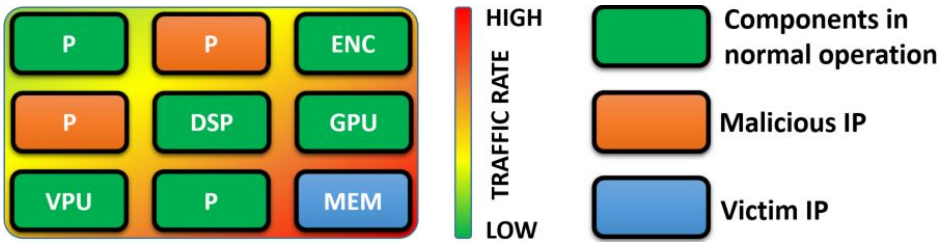


Figure 7-1. Example DDoS attack from malicious IPs to a victim IP in an NoC setup with Mesh topology.

Our threat model is generic, it does not make any assumption about the placement or the number of malicious IPs or victim IPs. Figure 7-2 shows four illustrative examples of malicious/victim IP placements that can lead to different communication patterns. Figure 7-2(a) shows a scenario involving one malicious IP and one victim IP. The other three examples represent scenarios where the packets injected from the malicious IPs to victim IPs are routed through paths that (b) partially overlap, (c) completely overlap and (d) form a loop. Our proposed approach is capable of both detecting and localizing all the malicious IPs in all these scenarios.

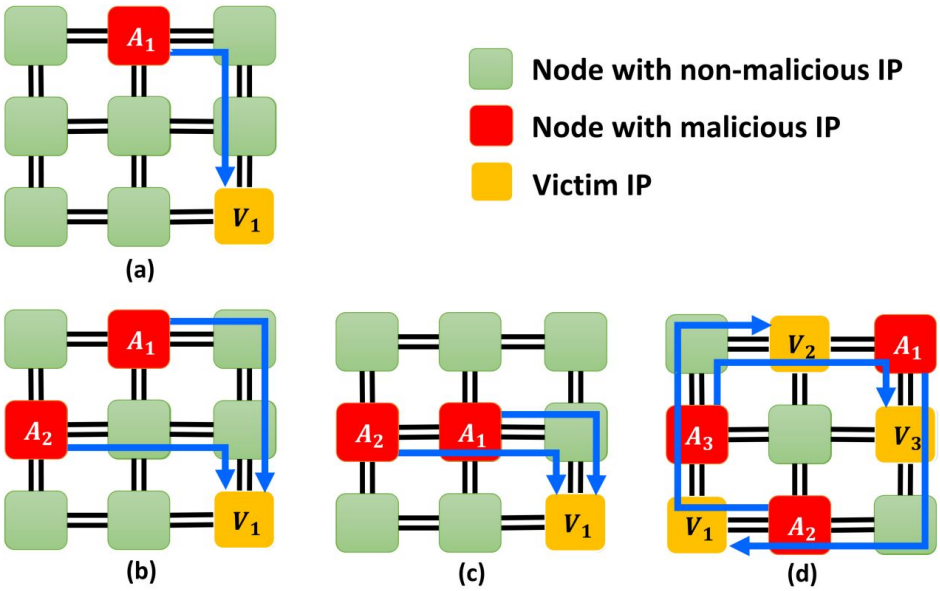


Figure 7-2. Different scenarios of malicious and victim IP placement.

7.1.2 Communication Model

Since each packet injected in the NoC goes through at least one router, we identify it to be an ideal NoC component for traffic monitoring. The router also has visibility to the packet header information related to routing. Packet arrivals at a router can be viewed as "events" and captured using arrival curves [177]. We denote the set of all packets passing through router r during a program execution as a "packet stream" P_r. Figure 7-3 shows two packet streams within a specific time interval $[1, 17]$. The stream P_r (blue) shows packet arrivals in normal operation and $\widetilde{P_r}$ (red) depicts a compromised stream with more arrivals within the same time interval. The packet count $N_{p_r}[t_a, t_b)$ gives the number of packets arriving at router r within the half-closed interval $[t_a, t_b)$. Equation 7-1 formally defines this using $N_{p_r}(t_a)$ and $N_{p_r}(t_b)$ - maximum number of packet arrivals up to time t_a and t_b, respectively. $\forall t_a, t_b \in \mathbb{R}^+, t_a < t_b, n \in \mathbb{N}$:

$$N_{p_r}[t_a, t_b) = N_{p_r}(t_b) - N_{p_r}(t_a) \tag{7-1}$$

Unlike [173] that monitors message streams at ECUs in a bus-based automotive architecture, our model is designed to monitor packets at routers of NoC-based SoC architectures.

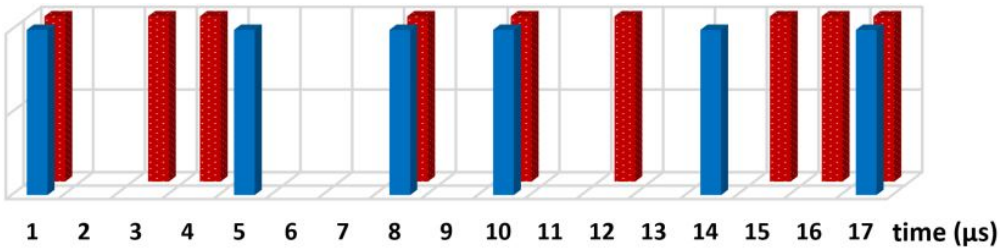


Figure 7-3. Example of two event traces. Six blue event arrivals represent an excerpt of a regular packet stream P_r and nine red event arrivals represent a compromised packet stream $\widetilde{P_r}$.

7.2 Real-Time Attack Detection and Localization

Figure 7-4 shows the overview of our proposed security framework to detect and localize DoS attacks originating from one or more MIPs. The first stage (upper part of the figure) illustrates the DDoS attack detection phase while the second stage (lower part of the figure)

represents the localization of MIPs. During the detection phase, the network traffic is statically analyzed and communication patterns are parameterized during design time to obtain the upper bound of "packet arrival curves" (PAC) at each router and "destination packet latency curves" (DLC) at each IP. The PACs are then used to detect violations of communication bounds in real-time. Once a router flags a violation, the IP attached to that router (local IP) takes responsibility of diagnosis. It looks at its corresponding DLC and identifies packets with abnormal latencies. Using the source addresses of those delayed packets, the local IP communicates with routers along that routing path to get their congestion information to localize the MIPs. The remainder of this section is organized as follows. The first two sections describe parameterization of PAC and DLC. Section 7.2.3 elaborates the real-time DDoS attack detection mechanism implemented at each router. Section 7.2.4 describes the localization of MIPs.

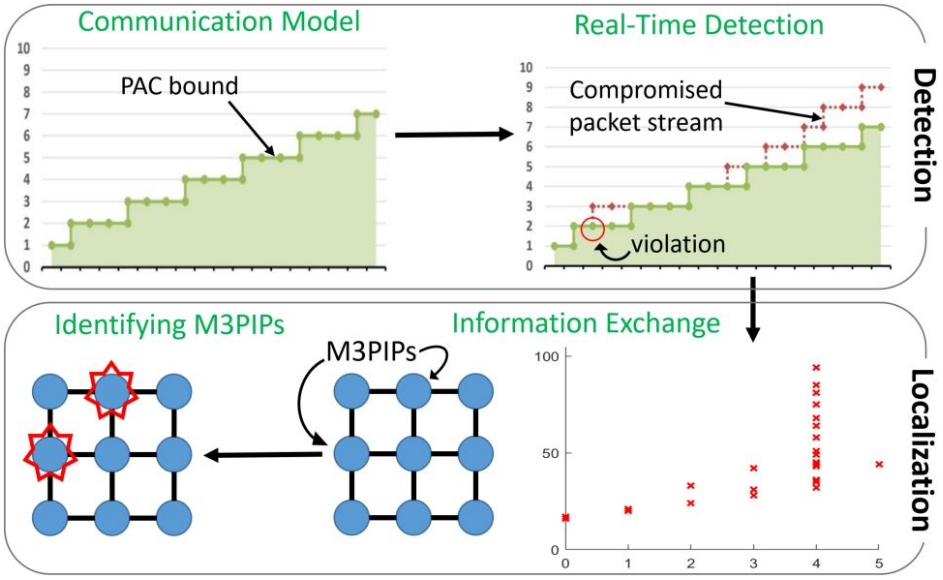


Figure 7-4. Overview of our proposed framework.

7.2.1 Determination of Arrival Curve Bounds

To determine the PAC bounds, we statically profile the packet arrivals and build the upper PAC bound ($\lambda^u_{pr}(\Delta)$) at each router. For this purpose, we need to find the maximum number of packets arriving at a router within an arbitrary time interval $\Delta(= t_b - t_a)$. This is done by

sliding a window of length Δ across the packet stream P_r and recording the maximum number of packets as formally defined in Equation 7-2.

$$\lambda^u_{p_r}(\Delta) = \max_{t \geq 0}\{N_{P_r}(t+\Delta) - N_{P_r}(t)\} \tag{7-2}$$

Repeating this for several fixed Δ, constructs the upper PAC bound. These bounds are represented as step functions. A lower PAC bound can also be constructed by recording the minimum number of packets within the sliding window. However, we exclude it from our discussion since in a DoS attack, we are only concerned about violating the upper bound. An example PAC bound and two PACs corresponding to the packet streams in Figure 7-3 are shown in Figure 7-5. During normal execution, the PACs should fall within the shaded area.

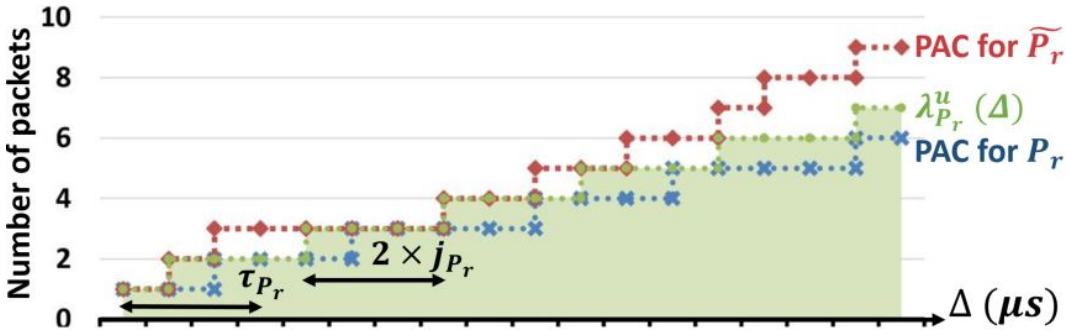


Figure 7-5. Graph showing upper ($\lambda^u_{p_r}(\Delta)$) bound of PACs (green line with green markers) and the normal operational area shaded in green.

While NoCs in general-purpose SoCs may exhibit dynamic and unpredictable packet transmissions, for vast majority of embedded and IoT systems, the variations in applications as well as usage scenarios (inputs) are either well-defined or predictable. Therefore, the network traffic is expected to follow a specific trend for a given SoC. SoCs in such systems allow the reliable construction of PAC bounds during design time. To get a more accurate model, it is necessary to consider delays that can occur due to NoC congestion, task preemption, changes of execution times and other delays. To capture this, we consider the packet streams to be periodic with jitter. The jitter corresponds to the variations of delays. Equation 7-3 represents the upper PAC bound for a packet stream P_r with maximum possible jitter j_{P_r} and period

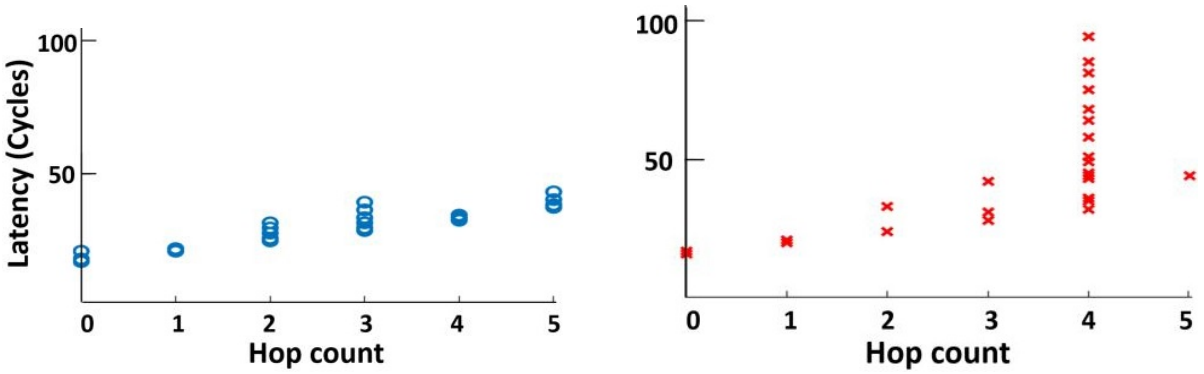
τ_{P_r} [178].

$$\forall \tau_{P_r}, j_{P_r} \in \mathbb{R}^+, \Delta > 0 : \lambda^u_{p_r}(\Delta) = \left\lceil \frac{\Delta + j_{P_r}}{\tau_{P_r}} \right\rceil \quad (7\text{-}3)$$

The equation captures the shift of the upper PAC bound because of the maximum possible jitter j_{P_r} relative to a nominal period τ_{P_r}. This method of modeling upper PAC bounds is validated by the studies in modular performance analysis (MPA) that uses real-time calculus (RTC) as the mathematical basis. MPA is widely used to analyze the best and worst case behavior of real-time systems. Capturing packet arrivals as event streams allows the packet arrivals to be abstracted from the time domain and represented in the interval domain (Figure 7-5) with almost negligible loss in accuracy [178]. The same model is used in the MATLAB RTC toolbox [179].

7.2.2 Determination of Destination Latency Curves

Similar to the PACs recorded at each router, each destination IP records a DLC. An example DLC in normal operation is shown in Figure 7-6A. The graph shows the latency against hop count for each packet arriving at a destination IP D_i. The distribution of latencies for each hop count is stored as a normal distribution, which can be represented by its mean and variance. Mean and variance of latency distribution at destination D_i for hop count k are denoted by $\mu_{i,k}$ and $\sigma_{i,k}$, respectively. In our example (Figure 7-6A), $\mu_{i,4}$ is 31 cycles and $\sigma_{i,4}$ is 2. During the static profiling stage, upon reception of a packet, the recipient IP extracts the timestamp and hop count from the packet header, and plots the travel time (from the source to the recipient IP) against the number of hops. The mean and variance are derived after all the packets have been received. The illustrative example considered one malicious IP four hops away from the victim IP launching the DoS attack. No other IP is communicating with the victim IP in a path that overlaps with the congested path. Therefore, the increased delay is observed only at hop count 4. In general, when multiple IPs send packets with destination D_i, and the paths overlap with the congested path, the increased delay will be reflected in several hop counts in the DLC. We did not show this scenario for the ease of illustration. However, such overlapping paths are considered in our experiments.



A Normal operation

B Attack scenario

Figure 7-6. Destination packet latency curves at an IP. The large variation in latency at hop count 4 in Figure 7-6(b) compared to Figure 7-6(a), contributes to identifying the malicious IP.

7.2.3 Real-time Detection of DoS Attacks

Detecting an attack in a real-time system requires monitoring of each message stream continuously in order to react to malicious activity as soon as possible. For example, each router should observe the packet arrivals and check whether the pre-defined PAC bound is violated. The attack scenario can be formalized as follows;

$$\exists t \in \mathbb{R}^{+} : \lambda^{u}_{p_r}(\Delta) < \max_{t \geq 0}\{N_{\widetilde{P_r}}(t+\Delta) - N_{\widetilde{P_r}}(t)\} \tag{7-4}$$

An obvious way to detect violations with the upper bound would be to construct the PAC and check if it violates the bound as shown in Figure 7-5. However, to construct the PAC, the entire packet stream should be observed. In other words, all packet arrivals at a router during the application execution should be recorded to construct the PAC. While it is feasible during upper PAC bound construction at design time, it doesn't lead to a real-time solution. Therefore, we need an efficient method to detect PAC bound violations during runtime.

To facilitate runtime detection of PAC bound violations, we use the "leaky bucket" algorithm, which considers packet arrivals and the history of packet streams and gives a real-time solution [180]. Once $\lambda^{u}_{p_r}(\Delta)$ is parameterized, the algorithm checks the number of packet arrivals within all time intervals for violations. Algorithm 9 outlines the leaky bucket

approach where $\theta_{r,s}$ denotes the minimum time interval between consecutive packets in a staircase function s at router r, and $\omega_{r,s}$ represents the burst capacity or maximum number of packets within interval length zero. $\lambda^u_{p_r}(\Delta)$, which is modeled as a staircase function can be represented by n tuples - ($\theta_{r,s}$, $\omega_{r,s}$), $s \in \{1, n\}$ sorted in ascending order with respect to $\omega_{r,s}$. This assumes that each PAC can be approximated by a minimum on a set of periodic staircase functions [181].

Algorithm 9 Detecting compromised packet streams

```
1: Input: (θ_{r,s},ω_{r,s}) tuples containing parameterized PAC bound at router r.
2: for s ∈ {1, n} do
3:     TIMER_{r,s} = θ_{r,s}
4:     COUNTER_{r,s} = ω_{r,s}
5: end for
6: if packetReceived = TRUE then
7:     for s ∈ {1, n} do
8:         if COUNTER_{r,s} = ω_{r,s} then
9:             TIMER_{r,s} = θ_{r,s}
10:        end if
11:        COUNTER_{r,s} = COUNTER_{r,s} − 1
12:        if COUNTER_{r,s} < 0 then
13:            attacked(r) = TRUE
14:        end if
15:    end for
16: end if
17: for s ∈ {1, n} do
18:    if timeoutOccured(TIMER_{r,s}) = TRUE then
19:        COUNTER_{r,s} = min(COUNTER_{r,s} + 1, ω_{r,s})
20:        TIMER_{r,s} = θ_{r,s}
21:    end if
22: end for
```

Lines 2-5 initializes the timers (TIMER$_{r,s}$) to $\theta_{r,s}$ and packet counters at time zero (COUNTER$_{r,s}$) to corresponding initial packet numbers $\omega_{r,s}$, for each staircase function and packet stream P_r. The DDoS attack detection process (lines 6-16) basically checks whether the initial packet limits (COUNTER$_{r,s}$) have been violated. Upon reception of a packet (line 6), the counters are decremented (line 11), and if it falls below zero, a potential attack is flagged (line 13). If the received packet is the first within that time interval (line

8), the corresponding timer is restarted (line 9). This is done to ensure that the violation of PAC upper bound can be captured and visualized by aligning the first packet arrival to the beginning of the PAC bound. When the timer expires, values are changed to match the next time interval (lines 18-21). As demonstrated in Section 7.3, the algorithm allows real-time detection of DDoS attacks under our threat model. Another important observation described in Section 7.3.4.1 drastically reduces the complexity of the algorithm allowing a lightweight implementation. The leaky bucket algorithm is originally proposed to check the runtime conformity of event arrivals in the context of network calculus. Its correctness is proven by [182].

7.2.4 Real-time Localization of Malicious IPs

Figure 7-6B shows an example DLC during an attack scenario, where all IPs are injecting packets exactly the same way as shown in Figure 7-6A except for one MIP, which injects a lot of packets to a node attached to a memory controller. Those two nodes are 4-hops apart in the Mesh topology. This makes the latency for 4-hop packets drastically higher than usual. For every hop count, we maintain the traffic distribution as a normal distribution using $\mu_{i,k}$ and $\sigma_{i,k}$. Once a potential threat is detected at a router, it sends a signal to the local IP. The local IP then looks at its DLC and checks if any of the curves have packets that took more than $\mu_{i,k} + 1.96\sigma_{i,k}$ time (95% confidence level). One simple solution is to examine source addresses of those packets and conclude that the source with most number of packets violating the threshold is the MIP. However, this simple solution may lead to many false positives. As each IP is distributed and examines the latency curve independently, the IP found using this method may or may not be a real MIP (attacker). Therefore, we call it a "candidate MIP".

To illustrate the difference between an attacker and a candidate MIP, we first examine four scenarios with only one attacker as shown in Figure 7-7. In these scenarios, the attacker A is sending heavy traffic to a victim IP V, and as a result, local IP D is experiencing large latency for packets from source S. The first three examples in Figure 7-7 show examples where candidate MIP S is not the real attacker A. Since a large anomalous latency is triggered by

the congestion in the network, the only conclusion obtained by the local IP from its DLC is that at least part of the path from candidate MIP to local IP is congested. We call the path from attacker A to victim V as the "congested path".

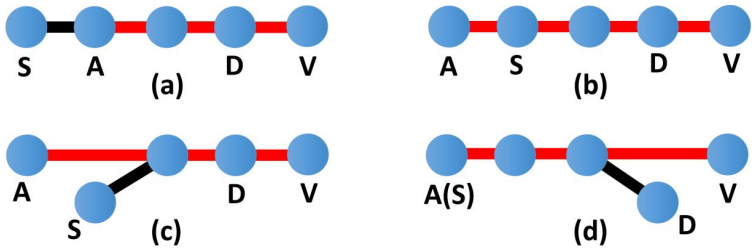


Figure 7-7. Four scenarios of the relative positions of local IP (D), attacker IP (A), victim IP (V), and the candidate MIP (S) as found by D.

In Figure 7-7(a) and Figure 7-7(c), the false positives of the candidate MIP S can be removed with global information of congested paths, by checking the congestion status of path from S to its first hop. It is certain that S is not the attacker when this path is not congested. However, we cannot tell whether S is the attacker when the path of S is congested. For example, the routers of Figures 7-7(b) and 7-7(d) are both congested, but S is not the attacker in 7-7(b).

Things get much worse when multiple attackers are present. If we look at the example in Figure 7-8, the path from candidate MIP S to local IP D is part of all paths along which three different attackers are sending packets to different victims. We define the "congested graph" as the set of all congested paths and all the routers in the paths. Since each hop connecting two routers consists of two separate uni-directional links, a congested graph is a bi-directional graph as shown in Figure 7-8. In order to detect attackers and avoid false positives, one simple solution would be building the entire congested graph by exchanging information from all the other routers and analyzing the graph to detect the actual MIPs. However, it would add a lot of burden on the already congested paths.

To overcome the bottlenecks, we propose a distributed and lightweight protocol implemented on the routers to detect the attackers. The event handler for each router for

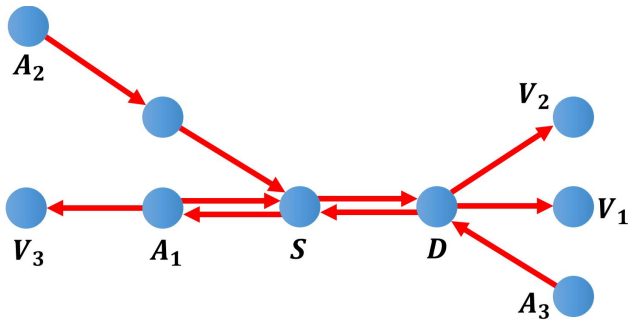


Figure 7-8. Congested graph of three attackers.

MIP localization is shown in Algorithm 10. The description of the steps of our complete protocol are shown below:

1. The router R detects an ongoing attack and sends a signal to the local IP (line 4). In Figure 7-7, both D and V will send a signal to their local IPs.

2. The local IP D looks at its DLC and responds to its router with a diagnostic message $< S, D >$ indicating the address of the candidate MIP S and destination D. The local router then forwards the packet towards S.

3. Each port in each router maintains a three-state flag to identify the attacker. The flag is 0, 1 and 2 to denote the attacker is undefined, local IP or others, respectively. When a diagnostic message $< S, D >$ comes in, R checks if the candidate MIP S is the local IP. If yes and its flag is not set yet, it will set the flag to be 1 (line 9). If S is not the local IP, it first finds out its neighbor N which sits in the path from S to R. If the one-hop path from N to R is congested, it sends the message to N (line 16) and sets the flag to 2, to indicate other IP as a potential attacker (line 17). Except for these two scenarios, the received message is a false positive and no action is taken (line 11 and 18), which will be explained in our examples. Note that the flag cannot decrease except for the reset signal which sets it to undefined (line 2). Therefore, if a diagnostic message already mentioned that other IPs may be the potential attackers, a new diagnostic message from the same port claiming that the local IP is the attacker will be ignored.

4. Each router maintains a timer. The timer starts as soon as any one of the router ports receive a diagnostic message. A pre-defined timeout period is used by each router. If the flag of any port is 1 after timeout, it broadcasts a message alerting that its local IP (line 23) is the attacker. Finally, a reset signal is triggered (line 24).

First, we will show that our approach works when a DoS attack is originating from only one MIP in the NoC. Later, we will describe how the proposed approach works in the presence of multiple MIPs mounting a DDoS attack.

7.2.4.1 DoS attack by a single MIP

We use Figure 7-7(b) to illustrate how our approach will localize the attacker when a DoS is caused by a single MIP. The router of S will receive two messages, one from the router of D saying that its local IP is a candidate MIP, and the other from the router of V saying that A is a candidate MIP, i.e., $< S, D >$ and $< A, V >$. Depending on the arrival time of these two messages, there are two scenarios. (a) $< S, D >$ comes first. It will change the flag of the corresponding port to 1 to denote that the local IP is the potential attacker. Then, S will receive $< A, V >$ through the same port. In this example, A is also the neighbor N. As the one-hop path from A to S is congested, the flag will be set to 2, denoting that the attacker is some other IP. (b) $< A, V >$ comes first. It will change the flag of the corresponding port to 2 to denote that the other IP is the potential attacker. Then, S will receive $< S, D >$ through the same port. As the flag is already set to 2, the received message is a false positive (line 11). When timeout occurs, nothing happens at the router of S. However, the router of A receives only the message from V indicating that its local IP is the potential attacker and its flag remains 1 when timeout occurs. A broadcast is sent indicating that A is the attacker.

For the case in Figure 7-7(a), A will receive a message from D indicating that S is a candidate MIP. However, when A checks the congestion status of the one-hop path from S to A, it will find out that the path is not congested. Therefore, the message is a false positive (line 18), and A will not change its flag. In other words, the flag of A will be set to 1 after receiving the message from V, and will not be changed by the message from D to S. After timeout, A will be identified as the attacker.

7.2.4.2 DDoS attack by multiple MIPs

Before giving an illustrative example of how our approach will localize attacks by multiple malicious IPs, we formally prove the correctness of our approach by proving the following theorem.

Theorem 7.1. *If the congested graph contains no loops, Algorithm 10 can localize at least one attacker.*

Algorithm 10 Event handler for router R

1: upon event **RESET:**
2: $R.flag[p_i] = 0$ for all ports p_i
3: upon event attacked == **TRUE:**
4: send a signal to local IP
5: upon receiving a diagnostic message $\langle S, D \rangle$ from port p_i:
6: start **TIMEOUT** if all $R.flag == 0$
7: **if** S is local IP **then**
8: **if** $flag[p_i] == 0$ **then**
9: $flag[p_i] = 1$ ▷ local IP is the MIP
10: **end if**
11: **if** $flag[p_i] == 2$ **then** ▷ false positive, do nothing
12: **end if**
13: **else** ▷ S is not local IP
14: Let N be the neighbor of R that sits in the path from S to R
15: **if** path from N to R is congested **then**
16: sends a diagnostic message $\langle S, D \rangle$ to N indicating that S is a candidate attacker
17: $flag[p_i] = 2$ ▷ other IP is the MIP
18: **else** ▷ false positive, do nothing
19: **end if**
20: **end if**
21: upon event **TIMEOUT:**
22: **if** any flag in $R.flag$ is 1 **then**
23: broadcasting that its local IP is the attacker
24: **RESET**
25: **end if**

Proof. We merge multiple diagnostic messages with the same destination as one message and ignore all false positive messages detected in line 11 and line 18 of Algorithm 10. We define message φ_i as a diagnostic message which points out that A_i is a candidate MIP. Consider the port of any attacker A_i that receives message φ_i. Such a port always exists in a DDoS attack scenario due to the fact that victim V_i will send a message φ_i to A_i saying that A_i is a candidate MIP. If φ_i is the only message received from this port, our algorithm can declare A_i as an attacker.

Our algorithm fails only when all routers connected to the attackers have flags set to either 0 or 2 in each of their ports as illustrated in Algorithm 10. This can only happen when each port that receives a diagnostic message, receives another diagnostic message

which causes the flag to be set to 2. Assume that a port in router of A_i receives messages $MS_i = \{\varphi_i, \varphi_j, ...\}$. It will digest the message φ_i and send out the remaining ones. We will construct a diagnostic message path in the following way. First, we add A_i to the path. Then, we select any message from MS_i other than φ_i, e.g., φ_j. Next, we follow the diagnostic message path from A_i to A_j, and add all routers to the path. By the same process, we select one message other than φ_j from MS_j, e.g., φ_k. Next, we follow the path from A_j to A_k. We can do this one by one since for every message set MS_u at attacker A_u, there is at least one message other than φ_u to select from. Therefore, the constructed diagnostic message path contains an infinite number of attackers, as shown in Figure 7-9. The infinite number of attackers implies that this path contains repeated attackers. Without loss of generality, we can assume that $A_k = A_i$. Since A_i cannot be sending out diagnostic messages MS_i through the same port that receives MS_i, the diagnostic path must form a loop. It is easy to see that diagnostic paths are the reverse of congested paths. As a result, there exists a loop in the congested graph, which contradicts the assumption made. Hence, Theorem 1 is proven. □

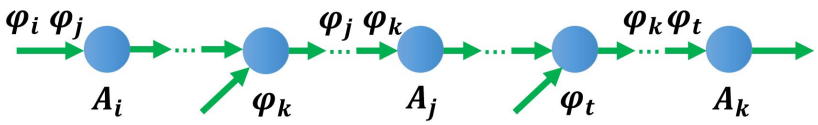


Figure 7-9. An example of a diagnostic message path constructed by following the flow of a diagnostic message in each attacker.

Thus, there always exists a port of the router connected to attacker A_i which receives only one diagnostic message φ_i given that there are no loops. This is a sufficient condition to detect A_i using Algorithm 10. Using our approach for localizing multiple malicious IPs gives rise to three cases that behave differently depending on how the MIPs are placed.

1. **Case 1:** If the congested paths do not overlap, all MIPs will be localized in one iteration using the process outlined above. This is the best case scenario for our approach and localizes MIPs in minimum time.

2. **Case 2:** If at least two paths overlap, it will need more than one iteration to localize all MIPs. To explain this scenario, an illustrative example is shown in Figure 7-10. Figure 7-10(a) shows the placement of the four MIPs (A_1, A_2, A_3, A_4) attacking the

victim IP (V). Once the attack is detected, in the first iteration, A_1, A_3 and A_4 are detected as shown in Figure 7-10(b). Due to the nature of our approach, A_2 is not marked as an attacker. This is caused by two diagnostic messages going in the paths $V \rightarrow A_2$ and $V \rightarrow A_3$. The router of A_2 will receive a message from the router of V saying that its local IP is a candidate MIP. It will change the flag of the corresponding port to 1 to denote that A_2 is the potential attacker. A_2 will receive another message from the router of V through the same port saying that A_3 is a candidate MIP. In this example, A_3 is also the neighbor of A_2. As the one-hop path from A_3 to A_2 is congested, the flag will be set to 2, denoting that the attacker is some other IP. When timeout occurs, nothing happens at the router of A_2. However, the router of A_3 receives only the message from V indicating that its local IP is the potential attacker and its flag remains 1 when timeout occurs. Therefore, A_3 is detected as an attacker whereas A_2 is not. In the case of A_1 and A_4, there is no overlap of congested paths and the two attackers are detected without any false negatives. Once the system resumes with only A_2 being malicious, the attacker will be detected and localized in the second iteration (Figure 7-10(c)). This case consumes more time since an additional detection phase is required to localize all MIPs. The number of iterations will depend on how many overlapped paths can be resolved at each iteration. In the worst case (where all congested paths can overlap and each iteration will resolve one path), the number of iterations will equal to the number of MIPs. However, our approach is guaranteed to localize all MIPs.

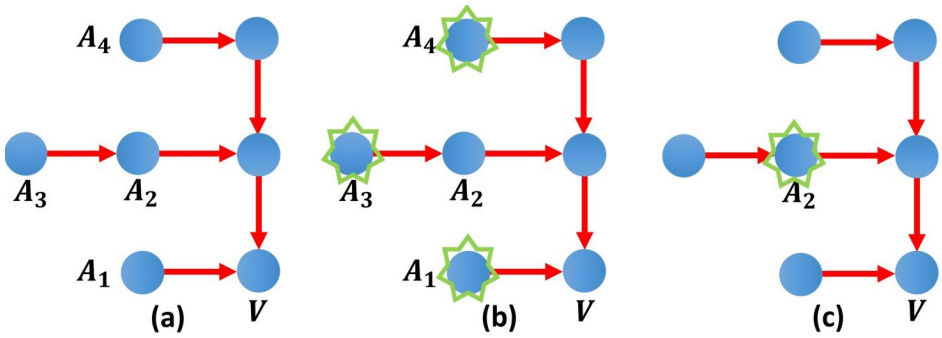


Figure 7-10. Illustrative example to show how our detection and localization framework works.

3. **Case 3:** The proof of Theorem 1 had the assumption that the congested graph contains no loops. Therefore, using our approach as it is, will not lead to localizing all MIPs if the congested graph forms a loop as shown in Figure 7-11. One solution is that any router in the congested loop can randomly "stop working" and resume after a short while. By breaking the loop, our approach will detect attackers with the new congested graph. The router "stopping work" can be triggered by the system observing that a DDoS attack is going on (during the detection phase), but no MIPs being localized.

In summary, our approach will detect one or more MIPs at each iteration depending on whether congested paths overlap. After detecting attackers(s) in the congested graph, their

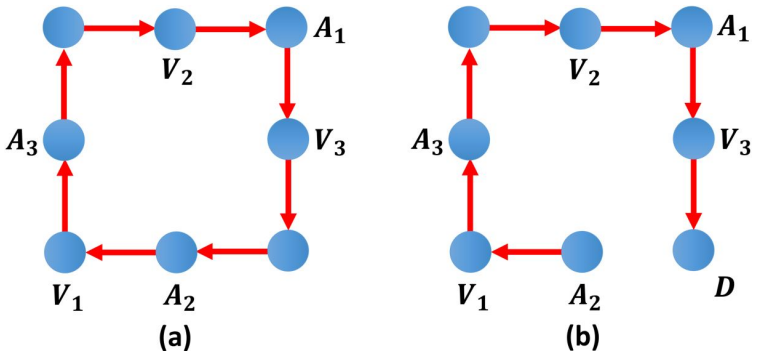


Figure 7-11. How three attackers can cooperate and construct a loop in the congested graph and how to localize attackers in such a scenario.

local router(s) can remove the attacker by dropping all its packets. Then, the process will be repeated with a new congested graph if more attackers exist. Our approach continues to find more attackers until either all attackers have been found, or the congested graph forms a loop, which can be handled using the method outlined above (Case 3).

It is easy to see that the extra work for the router is minimal in our protocol because all computations are localized. It only needs to check the congestion status of connected paths (one hop away), and compute the flag which has two bits for each port. Our protocol relies on the victim to pinpoint the correct attackers and the other routers to remove false positives. The timeout should be large enough for the victim to send messages to all the routers in the path of the attack. In practice, it can be the maximum communication latency between any two routers. The total time from detection to localization is the latency for packet traversal from the victim to attackers plus the timeout. Therefore, the time complexity for localization is linear in the worst case with respect to the number of IPs. It is important to note that most of the time, the diagnostic message path is the reverse of the congested path, and therefore, it is not congested.

7.3 Experiments

We have explored DoS attacks caused by a single MIP as well as multiple MIPs using the architecture shown in Figure 7-12. In Section 7.4, we evaluate the efficiency of our approach in an architecture model similar to one of the commercially available SoCs [4].

7.3.1 Experimental Setup

Our approach was evaluated by modeling an NoC-based SoC using the cycle-accurate full-system simulator - gem5 [18]. The interconnection network (NoC) was built on top of the "GARNET2.0" model that is integrated with gem5 [139]. The default gem5 source was modified to include the detection and localization algorithms. We experimented using several synthetic traffic patterns (uniform_random, tornado, bit_complement, bit_reverse, bit_rotation, neighbor, shuffle, transpose), topologies (Point2Point (16 IPs), Ring (8 IPs), Mesh4×4, Mesh8×8) and XY routing protocol to illustrate the efficiency of our approach across different NoC parameters. A total of 40 traffic traces were collected using the simulator by varying the traffic pattern and topology. Synthetic traffic patterns were only tested using one MIP in the SoC launching the DoS attack and an application instance running in 50% of the available IPs. These traffic traces act as test cases for our algorithms. The placement of the MIP, victim IP and IP(s) running the traffic pattern were chosen at random for the 40 test cases.

Our approach was also evaluated using real traffic patterns based on 5 benchmarks (FFT, RADIX, OCEAN, LU, FMM) from the SPLASH-2 benchmark suite [142] in Mesh 4×4 topology. Traffic traces from real traffic patterns were used to test both single-source DoS attacks as well as multiple-source DDoS attacks. The attack was launched at a node connected to a memory controller. Relative placements of the MIP and victim IP used to test the single-source DoS attack were the same as for the synthetic traces running on Mesh 4×4 topology (test case IDs 1 through 5 in Figure 7-14). For the DDoS attack involving multiple MIPs, we ran tests using the same set of benchmarks and topology with the victim and MIP placements as shown in Figure 7-12. The placement captures both Case 1 and Case 2 discussed in Section 7.2.4.2. Each node with a non-malicious IP ran an instance of the benchmark while the four nodes in the four corners were connected to memory controllers. The jitter for all applications was calculated using the method proposed in [183].

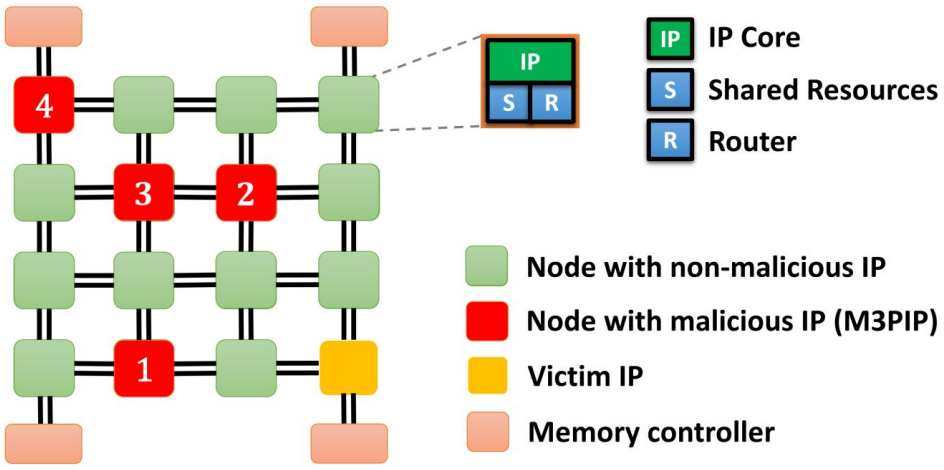


Figure 7-12. MIP and victim IP placement when running tests with real benchmarks on a 4x4 Mesh NoC.

7.3.2 Efficiency of Real-time DoS Attack Detection

Before showing results of our experimental evaluation, we will first give an illustrative example to show how the parameters associated with the leaky bucket algorithm (Algorithm 9) is calculated and used in attack detection.

An important observation allows us to reduce the number of parameters required to model the PACs, and as a result, implement a lightweight scheme with much less overhead. The model in Equation 7-3 is derived using the fact that the packet streams are periodic with jitter. As proposed in [173] and [184], for message streams with such arrival characteristics, the PACs can be parameterized by using only worst case jitter j_{P_r}, period τ_{P_r} and an additional parameter ϵ_r which denotes the packet counter decrement amount. The relationship between these parameters are derived in [181] as shown in Equation 7-5.

$$\theta_r = greatest\_common\_divisor(\tau_{P_r}, \tau_{P_r} - j_{P_r}) \tag{7-5a}$$

$$\omega_r = 2 \times \epsilon_r - \frac{\tau_{P_r} - j_{P_r}}{\theta_r} \tag{7-5b}$$

$$\epsilon_r = \frac{\tau_{P_r}}{\theta_r} \tag{7-5c}$$

To use these parameters, the only changes to Algorithm 9 are at line 11 ($\text{COUNTER}_{r,s} = \text{COUNTER}_{r,s} - \epsilon_r$) and one tuple per packet stream instead of n tuples ($s \in \{1\}$). The illustrative example is based on this observation.

Illustrative Example: Consider the example packet streams shown in Figure 7-3. Assume that the packet steam P_r has a period $\tau_{P_r} = 3\mu s$ and jitter $j_{P_r} = 1.5\mu s$. During an attack scenario, this stream is changed to stream $\widetilde{P_r}$ with $\tau_{\widetilde{P_r}} = 2\mu s$ and no jitter. Using these values in Equation 7-5 will give $\theta_r = 1.5\mu s, \omega_r = 3$ and $\epsilon_r = 2$, which are the parameters used in the leaky bucket algorithm. Therefore, $COUNTER_{r,s}$ is initialized with 3 (line 4, line 19) and decremented by 2 at each message arrival (line 11). $TIMER_{r,s}$ is initialized to $1.5\mu s$ (line 3, line 20). Using these values and running the detection algorithm during the attack scenario will lead to a detection time of $4\mu s$. Figure 7-13 shows the values of the parameters changing with each packet arrival and timeout leading to the detection of the attack at $t = 4\mu s$.

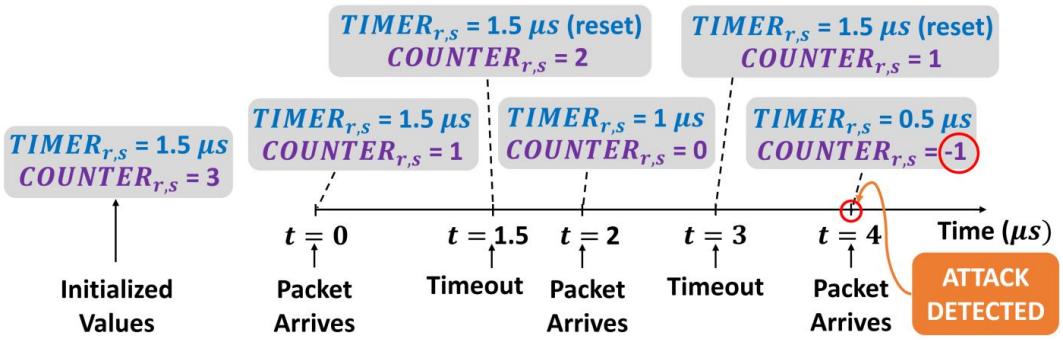


Figure 7-13. Illustrative example of parameter changes in the leaky bucket algorithm with packet arrivals and timeouts.

The experimental evaluation follows the same process as the illustrative example using the experimental setup described in Section 7.3.1. Figure 7-14 shows the detection time across different topologies for synthetic traffic traces in the presence of one MIP. The 40 test cases are divided into different topologies, 10 each. The packet stream periods are selected at random to be between 2 and 6 microseconds. Attack periods are set to a random value between 10% and 80% of the packet stream period. The detection time is approximately twice

the attack period in all topologies. This is expected according to Algorithm 9 and consistent with the observations in [173].

In addition to the time taken by the leaky bucket approach, the detection time also depends on the topology. For example, attack detection in Point2Point topology (Figure 7-14(a)), where every node is one hop away, requires less time to detect compared to Mesh8×8 (Figure 7-14(d)) where some nodes can be multiple hops away. The topology mainly affects attack localization time due to the number of hops from detector to attacker. But for detection, topology plays a relatively minor role since the routers are connected to each IP and detection mechanism neither takes into account the source nor the destination of packets. The routers only look at how many packets arrived in a given time interval. It is also important to note that any router in the congested path can detect the attack, not only the router connected to the victim IP. A combination of these reasons have led to the topology playing a relatively minor role in attack detection time. These results confirm that the proposed approach can detect DoS attacks in real-time.

Results for DDoS attack detection in the presence of multiple attacking MIPs are shown in Figure 7-15 and Figure 7-16. For all of these experiments, packet stream period is fixed at $2.5\mu s$ and attack period is set to $1.5\mu s$. Figure 7-15 shows detection time variation in the presence of different number of IPs across benchmarks. The time to detect an ongoing attack in the multiple MIP scenario is typically less than the single MIP scenario. When more IPs are malicious, the detection time shows a decreasing trend. This is expected since multiple attackers flood the NoC faster and cause PAC bound violations quicker. To compare detection time with packet stream period and attack period, we have shown the detection time variation in the presence of four MIPs across benchmarks in Figure 7-16.

7.3.3 Efficiency of Real-time DoS Attack Localization

We measured the efficiency of attack localization by measuring the time it takes from detecting the attack to localizing the malicious IPs. According to our protocol, this is mainly dominated by the latency for packet traversal from victim to attacker (V2AL) as well as the

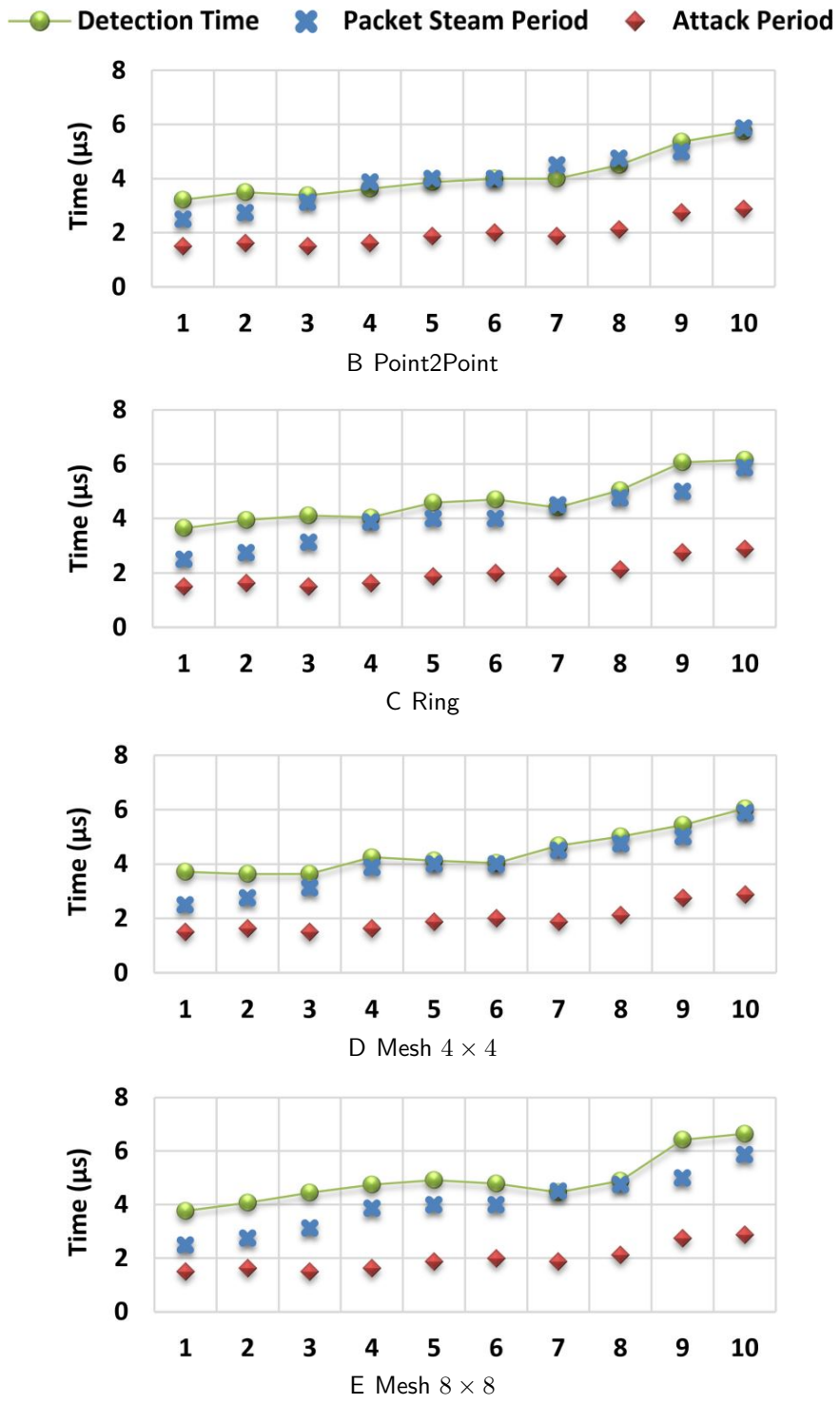


Figure 7-14. Attack detection time for different topologies when running synthetic traffic patterns with the presence of one MIP.

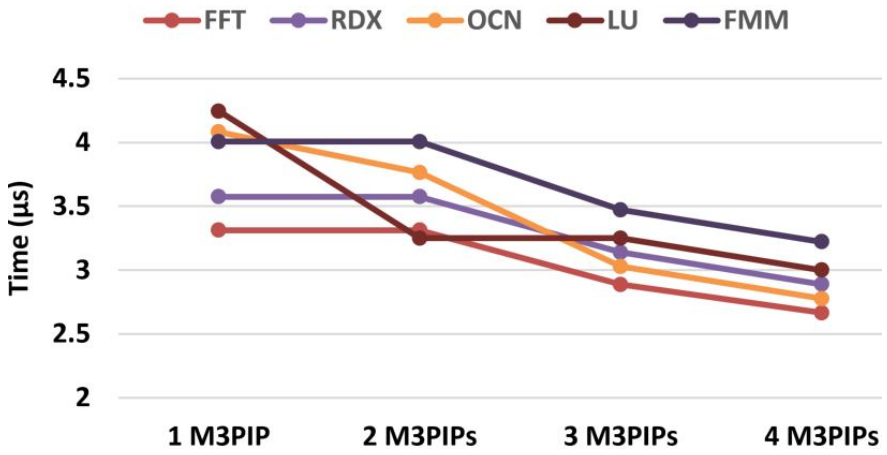


Figure 7-15. Attack detection time when running real benchmarks with the presence of different number of MIPs.

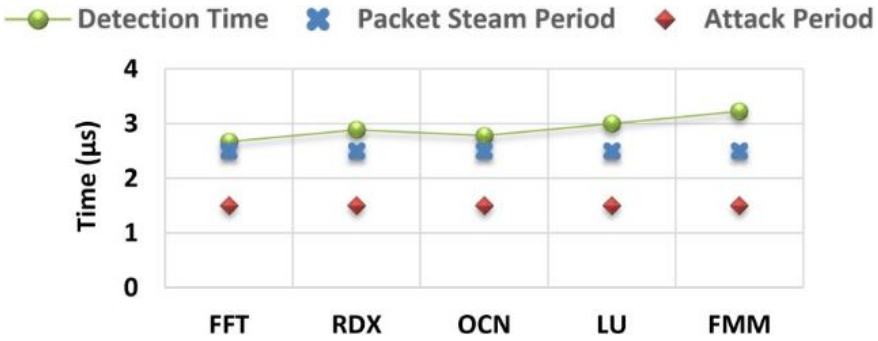


Figure 7-16. Attack detection time when running real benchmarks with the presence of four MIPs.

timeout (TOUT) described in Section 7.2.4. Figure 7-17 shows these statistics using the same set of synthetic traffic patterns for the single MIP scenario. The experimental setup for the localization results corresponds to the experimental results for the detection results in Figure 7-14. Unlike the detection phase, since the localization time depends heavily on the time it takes for the diagnostic packets to traverse from the IPs connected to the routers that flagged the attack to the potentially malicious IPs, the localization time varies for each topology. For example, in a Point2Point topology, localization needs diagnostic message to travel only one hop, whereas a Mesh8x8 topology may require multiple hops. Therefore, localization is faster in Point2Point compared to Mesh8x8 as shown in Figure 7-17. The localization time is less compared to detection time because the localization process completes

once the small number of diagnostic packets reach all the potentially malicious IPs, whereas detection requires many packets before violating a PAC bound during runtime.

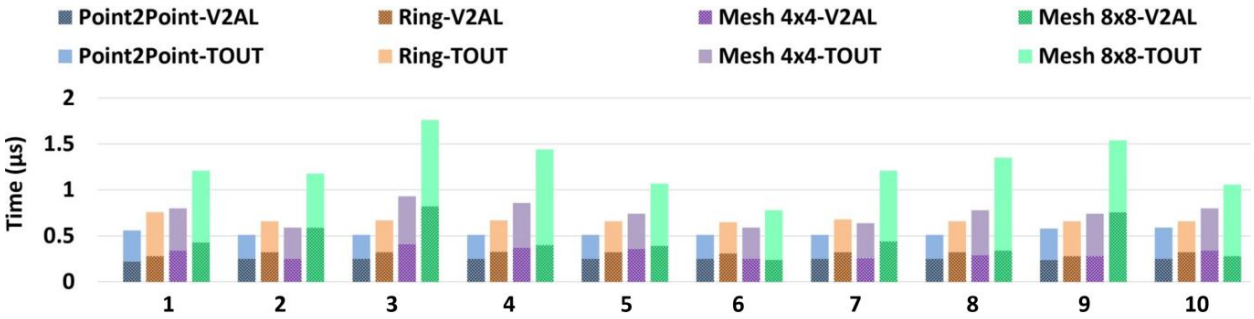


Figure 7-17. Attack localization time for synthetic traffic patterns in the presence of one MIP.

Results for DDoS attack localization in the presence of multiple MIPs when running real benchmarks is shown in Figure 7-18. Similar to the experiments done for DDoS attack detection efficiency, localization results are shown for one, two, three and four MIPs attacking the victim IP at the same time. The time is measured as the time it takes since launching the attack, until the localization of all MIPs. Once the first iteration of localization and detection is complete, the attack has to be detected again before starting the localization procedure. Therefore, the y-axis shows detection as well as localization time. For clarity of the graph, unlike in Figure 7-17, we have shown total localization time for each iteration rather than dividing the localization time as V2AL and TOUT. For both one and two MIP scenarios, only one iteration of detection and localization is required. When the third MIP is added, the two congested paths from victim to second MIP and from victim third MIP overlap. Therefore, only the first and third MIPs are localized during the first iteration leaving the second MIP to be detected during the second iteration. Similarly, in the four MIP scenario, first, third and fourth MIPs are localized during the first iteration and the second MIP, during the second iteration. This is consistent with our discussion presented in Section 7.2.4.2. The results show that both detection and localization can be achieved in real-time. If a system requires only detection, the architecture of our framework allows easy decoupling of the two steps.

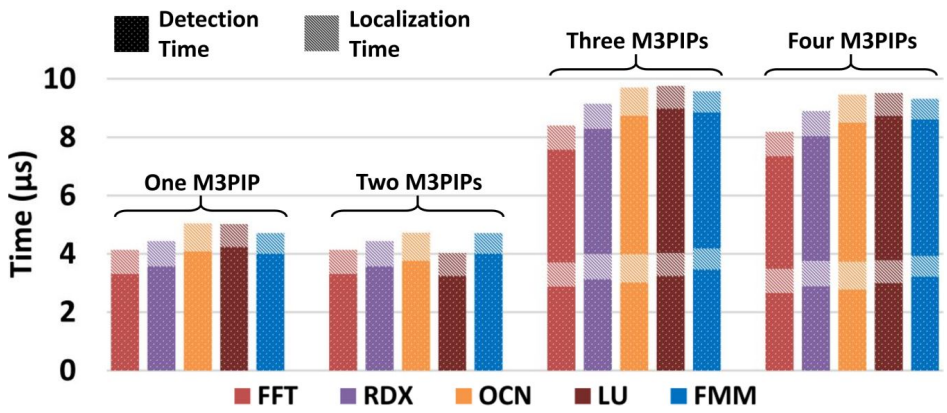


Figure 7-18. Attack localization time when running real benchmarks with the presence of different number of MIPs.

7.3.4 Overhead Analysis

The overhead is caused by the additional hardware that is required to implement the DoS attack detection and localization processes. The detection process requires additional hardware components and memory implemented at each router to monitor packet arrivals as well as store the parameterized curves. The localization process uses DLCs stored at IPs and the communication protocol implemented at the routers. Figure 7-19 shows an overview of how our security components are integrated into the NoC components. The observation made in Section 7.3.1 allows us to reduce the number of parameters required to model the PACs, and as a result, reduces the additional memory requirement and improves performance. The following sections evaluate the power, performance and area overhead of the optimized algorithms.

7.3.4.1 Performance overhead

In our work, we used the 5-stage router pipeline (buffer write, virtual channel allocation, switch allocation, switch traversal and link traversal) implemented in gem5. The computations related to the leaky bucket algorithm can be carried out in parallel to these pipeline stages once separate hardware is implemented. Therefore, no additional performance penalty for DDoS attack detection.

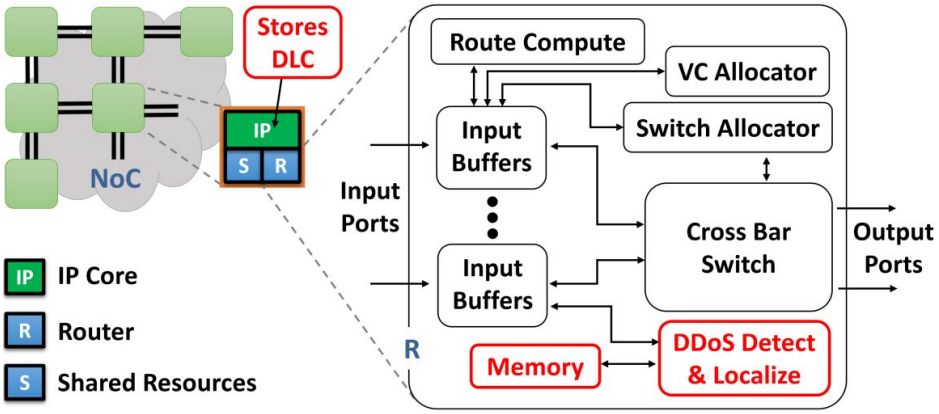


Figure 7-19. Block diagram of NoC architecture showing additional hardware required to implement our security protocol in red.

During the localization phase, the diagnostic messages do not lead to additional congestion for two reasons. (1) As shown in Algorithm 10, the diagnostic message is transmitted along the reverse direction of the congested path. Since routers utilize two separate uni-directional links, the diagnostic messages are not sent along the congested path. (2) While it is unlikely, it is possible for multiple MIPs to carefully select multiple victims to construct a congested path in both directions. Even in this scenario, the number of diagnostic messages is negligible. This is because when an attack is flagged by the detection mechanism, diagnostic messages are sent to the source IPs which have violated the DLC threshold. Since the number of such source IPs can be at most the number of IPs communicating with the node that detected the attack, the performance impact by diagnostic messages is negligible.

7.3.4.2 Hardware overhead

We consider overhead due to modifications in the router, packet header as well as local IPs, as outlined below.

Router: The proposed leaky bucket algorithm is lightweight and can be efficiently implemented with just three parameters per PAC bound as discussed above. The localization protocol requires two-bit flags at each port resulting in 10 bits of memory per router in Mesh topology. To evaluate the area and power overhead of adding the distributed DoS attack detection and localization mechanism at each router, we modified the RTL of an open-source

NoC Router [185]. The design is synthesized with the 180nm GSCLib Library from Cadence using the Synopsys Design Compiler. It gave us area and power overhead of 6% and 4%, respectively, compared to the default router.

Packet Header: In a typical packet header, the header flit contains basic fields such as source, destination addresses and the physical address of the (memory) request. Some cache coherence protocols include special fields such as flags and timestamps in the header. If the header carries only the basic fields, the space required by these fields are much less compared to the wide bit widths of a typical NoC link. Therefore, most of the available flit header space goes unused [186]. We used some of these bits to carry the timestamp to calculate latency. This eliminates the overhead of additional flits, making better utilization of bits that were being wasted. If the available header bit space is not sufficient, adding an extra "monitor tail flit" is an easily implementable alternative [186]. In most NoC protocols, the packet header has a hop count or time-to-live field. Otherwise, it can be derived from the source, destination addresses and routing protocol details.

Local IP: The DLPs are stored and processed by IPs connected to each node of an NoC. Since the IPs have much more resources than any other NoC component, the proposed lightweight approach has negligible power and performance overhead. We store $\mu_{i,k} + 1.96\sigma_{i,k}$ as a 4-byte integer for each hop count. Therefore, the entire DLP at each IP can be stored using $1 \times m$ parameters where m is the maximum number of hops between any two IPs in the NoC. It gives a total memory space of just $1 \times m \times 4$ bytes.

Our evaluations demonstrate that the area, power and performance overhead introduced by our approach is negligible.

7.4 Case Study with Intel KNL Architecture

In the previous section, we have applied our approach using a regular 4x4 Mesh architecture (Figure 7-12). In order to demonstrate the applicability of our approach across NoC architectures, in this section, we evaluate the efficiency of our approach in an architecture model similar to one of the commercially available SoCs - Intel's KNL architecture [4] that

was introduced in Section 3.2. We model the architecture on gem5 according to the validated simulator model discussed in Chapter 3 and show results for both DDoS attack detection and localization. Our goal is to simulate the NoC traffic behavior in a realistic architecture and evaluate how our security framework performs in it.

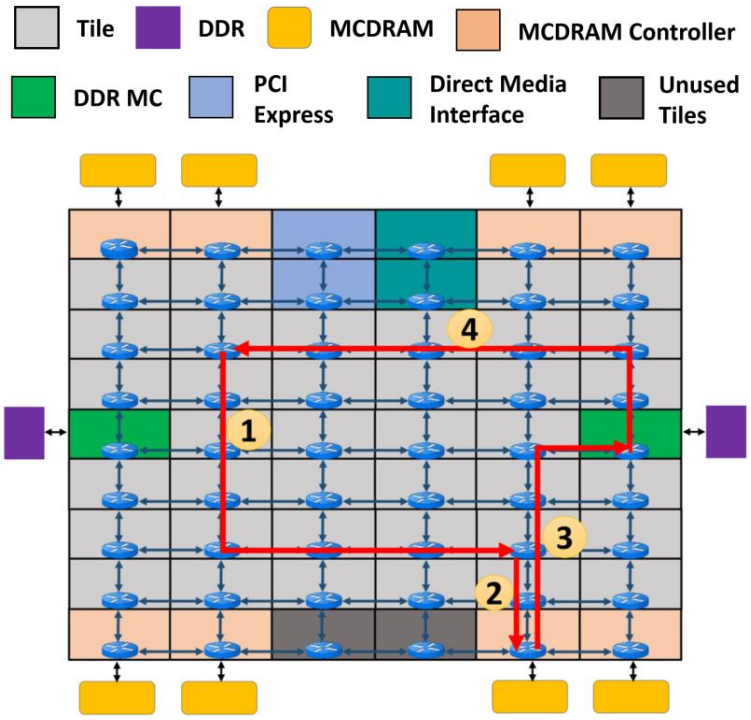


Figure 7-20. Overview of the KNL architecture together with an example of MCDRAM miss in Cache memory mode and All-to-all cluster mode.

In this model, 32 tiles connect on a Mesh NoC. Each tile is composed of a core that runs at 1.4 GHz, private L1 cache, tag directory and a router. Each cache is split into data and instruction caches with 16kB capacity each. The complete set of simulation parameters are summarized in Table 7-1. The memory controllers are placed to match the architecture shown in Figure 7-20. We made few modeling choices that deviates from the actual KNL hardware due to the following reasons;

- 32 tiles are used instead of the 36 in KNL since the number of cores in gem5 must be a power of 2. This can be considered as a use-case where the KNL hardware has switched off cores in four of its tiles.

Table 7-1. System configuration parameters used when modelling KNL on gem5 simulator.

Parameter Class	Parameter	Value
Processor Configuration	Number of cores	32
	Core frequency	1.4 GHz
	Instruction set architecture	x86
Memory System Configuration	L1 cache	private, separate instruction and data cache. Each 16kB in size.
	Cache coherence	distributed directory-based protocol
	Memory size	4GB DDR
	MCDRAM	shared, direct mapped cache
	Access latency	300 cycles
Interconnection Network Configuration	Topology	8x4 Mesh
	Routing scheme	X-Y deterministic
	Router	4 port, 4 input buffer router with 5 cycle pipeline delay
	Link latency	1 cycle

- The cache sizes we used are less compared to the actual KNL hardware numbers. This was done to get 95% hit rate in L1 cache, which is usually the hit rate when running embedded applications for the benchmarks we used. If we used a larger cache size, the L1 hit rate would be 100%, and NoC optimization will not affect cache performance.
- KNL runs AVX512 instructions whereas the gem5 model runs X86. gem5 is yet to support AVX512 instructions.
- Each tile in KNL consists of two cores. Our detection mechanism is capable of detecting DDoS attacks irrespective of whether one or both cores in a tile are active. However, the localization method can only pinpoint which tile is malicious. Since detection as well as localization happens at the router level, it is not possible to pinpoint the malicious core in a tile if both cores are active. Therefore, in our experimental setup, we assumed that one core per tile is active simulating 50% utilization.

Therefore, our gem5 model is a simplified version of the real KNL hardware. However, our previous work has validated the model and related performance and energy results to show that it accurately captures relative advantages/disadvantages of using different memory and cluster modes [153]. To evaluate our security framework, out of the memory and cluster modes, we model the cache memory mode and all-to-all cluster mode. The traffic flow when applications are running is defined by these modes. Figure 7-20 shows an example traffic flow.

We ran the same real traffic patterns (benchmarks) we used in Section 7.3.1. To mimic the highly parallel workloads executable by the KNL architecture, we utilized 50% of the total available cores when running each application by running an instance of the benchmarks in each active core. The DDR address space was used uniformly for each benchmark. Attackers were modeled and placed randomly in 25% of the tiles that doesn't have an application instance. The DDoS attack was launched at the memory controller that experienced highest traffic during normal operation. Given that our model has 32 cores, 16 of them ran instances of the benchmark and 4 of the non-active cores injected packets directed at the memory controller to simulate the behavior of malicious IPs launching a DDoS attack. The packet stream period and attack period were selected as explained in Section 7.3.2. Figure 7-21 shows the placement of the four MIPs, cores running the benchmarks (active cores) and the victim IP when running the RADIX benchmark. The victim IP depends on the benchmark since it is the IP connected to the memory controller experiencing highest traffic during normal operation.

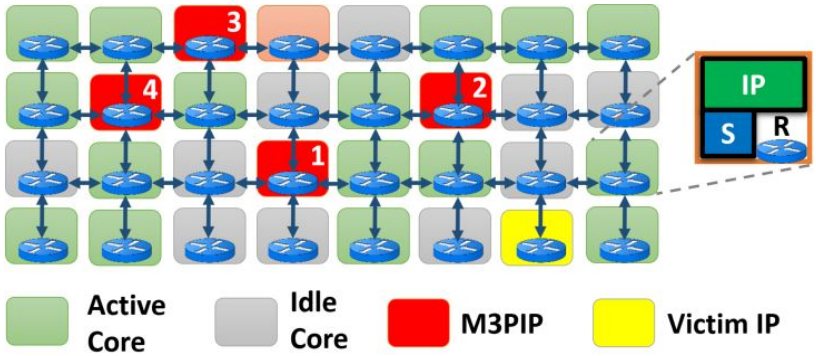


Figure 7-21. 4×8 Mesh NoC architecture used to simulate DoS attacks in an architecture similar to KNL.

Similar to the experimental results presented in Section 7.3.1, the DDoS attack detection results are shown in Figure 7-22 and Figure 7-23. Figure 7-22 shows detection time variation across benchmarks and number of MIPs. A zoomed-in version of the four MIP scenario is shown in Figure 7-23. Attack localization results are shown in Figure 7-24. Until the fourth MIP is added, there are no overlapping congested paths. Therefore, the MIPs are localized using only one iteration. Once the fourth MIP is added, the first, third and fourth MIPs are

localized during the first iteration and a second iteration is required to localize the second MIP. This is reflected in localization time in Figure 7-24. From these as well as the previous results we notice that our detection and localization framework gives real-time results across different topologies and architectures.

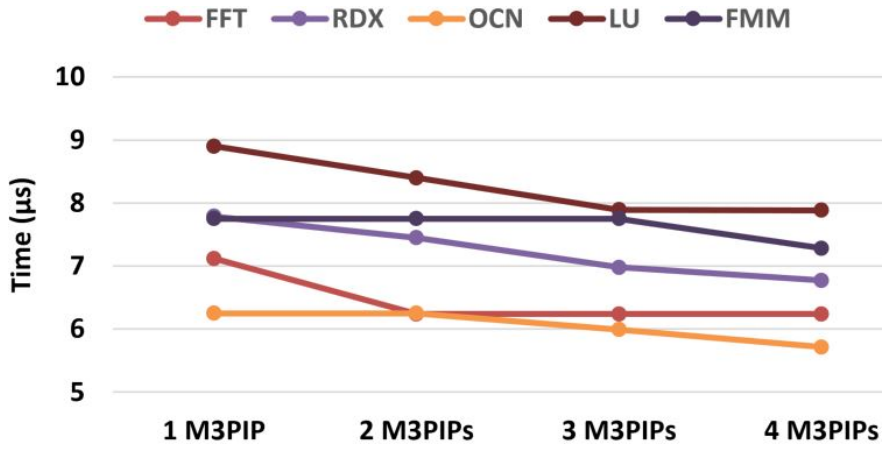


Figure 7-22. Attack detection time when running real benchmarks on an architecture similar to KNL with the presence of different number of MIPs.

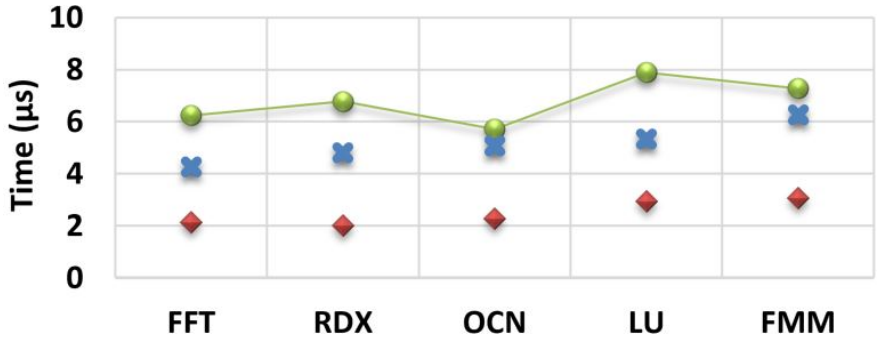


Figure 7-23. Attack detection time when running real benchmarks on an architecture similar to KNL with the presence of four MIPs.

7.5 Discussion

Our proposed approach is designed for DDoS attack detection and localization, and therefore, it is not suitable to capture other forms of security violations such as eavesdropping, snooping and buffer overflow. Specific security attacks would require other security countermeasures which are not covered in this chapter. Due to the low implementation cost, our approach can be easily coupled with other security countermeasures.

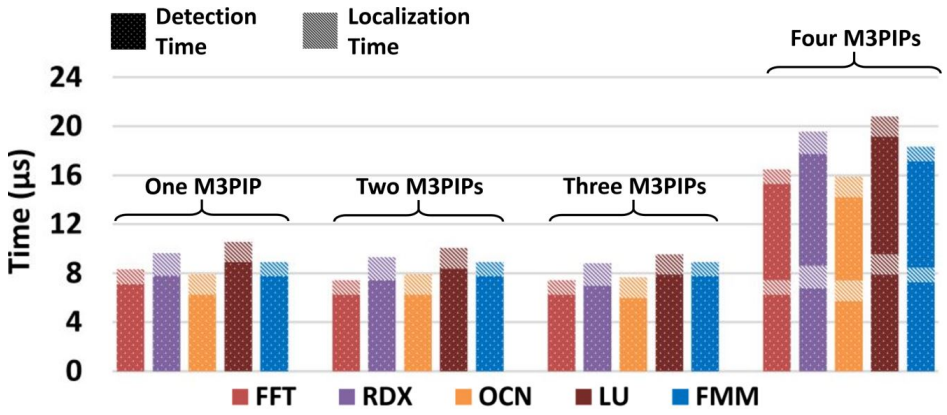


Figure 7-24. Attack localization time when running real benchmarks on an architecture similar to KNL with the presence of different number of MIPs.

For example, [49] discussed a snooping attack in which the header of the packet is modified before injecting into the NoC. This will alter the source address of the packet. While our detection mechanism does not depend on any of the header information of the packet, since our localization method uses the source address to localize the M3PIPs, an address validation mechanism needs to be implemented at each router to accommodate header modification. The address validation can be implemented as follows. Before a router injects each packet that comes from the local IP into the NoC, the router can check the source address and if it not the address of the local IP attached to that router, the router can drop it without injecting in to the NoC.

Our proposed work is targeted for embedded systems with real-time constraints. Such systems allow only a specific set of scenarios in order to provide real-time guarantees. Features commonly observed in general purpose computing such as task mapping, runtime task-migration, adaptive routing and introduction of new applications during runtime are beyond the scope of this work. In order to apply our proposed approach in general purpose systems, we need to store PACs and DLCs corresponding to each scenario and select the respective curves during runtime. As discussed in Section 7.3.4, the hardware overhead to store the parameterized curves for each scenario is minimal, which consists of two major parts (i) overhead for storing the curves ($1 \times m \times 4$ bytes), and (ii) overhead for runtime monitoring

(6% of NoC area). For example, if we consider an 8x8 Mesh, the memory overhead to store the curves would be 56 bytes ($m = 14$). If N scenarios are considered, the overhead would be 6% + $N \times 56$. Therefore, it may be feasible to consider a small number of scenarios (e.g., $N < 10$) without violating area overhead constraints.

7.6 Summary

This chapter presented a real-time and lightweight DDoS attack detection and localization mechanism for IoT and embedded systems. It relies on real-time network traffic monitoring to detect unusual traffic behavior. This chapter made two major contributions. It proposed a real-time and efficient technique for detection of DDoS attacks originating from multiple malicious IPs in NoC-based SoCs. Once an attack is detected, my approach is also capable of real-time localization of the malicious IPs using the latency data in the NoC routers. I demonstrated the effectiveness of my approach using several NoC topologies and traffic patterns. In my experiments, all the attack scenarios were detected and localized in a timely manner. Overhead calculations have revealed that the area overhead is less than 6% to implement the proposed framework on a realistic NoC model. This framework can be easily integrated with existing security mechanisms that address other types of attacks such as buffer overflow and information leakage.

CHAPTER 8
TRUST-AWARE ROUTING IN THE PRESENCE OF MALICIOUS IPS

Integrity of packets traversing through a network is a well-studied problem. Consider a scenario where the integrity of exchanged data is ensured using a message authentication code (MAC). The sender IP sends a packet together with an authentication tag, and the receiver re-computes the tag to check for data integrity. If it doesn't match, the packet has been tampered during communication, and a re-transmission is required. This method of error correction is widely employed in NoC-based SoCs [98, 187]. However, re-transmissions due to corrupt packets can lead to several problems:

- Increased latency because of re-transmission as well as additional stall cycles introduced by the IP cores while waiting for the requested data.
- This can increase the number of packets traversing the network, and as a result, increased energy consumption and performance penalty.
- In MAC-then-encrypt protocols [188][1] , authentication tag is computed on the plaintext, appended to the data, and then tag and plaintext are encrypted together. When MAC is computed in this way, the receiver IP has no way of knowing whether the message was indeed authentic or tampered until the message is decrypted. Therefore, the resources spent to decrypt a tampered packet is wasted.

Systematic exploitation of error correction protocols, such as the one explained above, can lead to Denial-of-Service (DoS) attacks. For example, a malicious IP can corrupt data on purpose and cause continuous re-transmissions leading to a DoS attack [104]. Specifically, the threat model explored in this chapter is as follows.

Threat Model: Figure 8-1 shows a standard NoC-based many-core architecture with IPs connected in a Mesh topology. Each IP connects to a router via a network interface. The network interface accommodates the authentication scheme which implements MAC-based authentication [97]. A packet originating from a source IP (src) in a secure zone has to traverse through the non-secure zone in order to reach the destination IP (dest) in another

[1] MAC-then-encrypt is the standard method used in TLS [188].

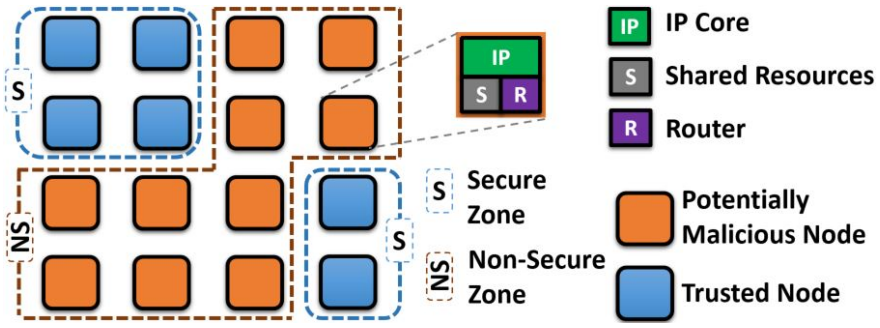


Figure 8-1. Overview of a typical SoC architecture with secure and non-secure zones.

secure zone. The IPs in the non-secure zone are potentially malicious. In reality, out of all the potentially malicious IPs, only a small fraction is actually malicious. We call them malicious IPs (MIP) in this chapter. If the packet traverses through such an MIP, it can tamper with the packet and therefore, at dest, the authentication tag computation will not match and the packet will be dropped. The src will re-transmit the packet since a response is not received from the dest within the time-out period. The problem of minimizing this impact gets aggravated due to two challenges. (1) The MIP will not always behave maliciously. In other words, it will tamper packets only in sporadic intervals. (2). Since the src depends on the response from the dest to know whether the packet was received or not, the MIP can tamper the packet between src and dest or tamper the response packet between dest and src, and both of these scenarios lead to the same outcome from the src's point of view. I consider both of these challenges when proposing the solution. A similar threat model was used in a previous study that proposed countermeasures for DoS attacks [104].

In this chapter, I propose a trust-aware routing protocol that avoids MIPs when two secure IPs are communicating with each other. The proposed approach leads to less re-transmissions, and as a result, improved performance and energy efficiency. Trust-aware routing can complement existing NoC attack detection and mitigation techniques by allowing on-chip communication even in the presence of an adversary while minimizing the energy and performance overhead.

The major contributions can be summarized as follows;

1. I propose a "trust model" that effectively calculates trust between neighboring routers and propagates trust values through the NoC.

2. I have developed a routing protocol that uses the trust values between routers to make routing decisions such that the MIPs are avoided by packets when routing from source to destination.

3. I have evaluated the effectiveness of the approach using both real benchmarks and synthetic traffic patterns to demonstrate that it leads to significant improvement in both performance and energy efficiency.

The remainder of the chapter is organized as follows. Section 8.1 discusses the motivation behind my work. Section 8.2 presents my proposed NoC trust model. Section 8.3 describes the trust-aware routing protocol that utilizes the NoC trust model. Section 8.4 presents the experimental results. Finally, Section 8.5 summarizes the chapter.

8.1 Motivation

Lightweight authentication schemes implemented on NoC-based SoCs, try to provide desired security while consuming minimum number of cycles. However, if the MAC fails to match at the receiver's end, the src has to re-transmit again, leading to wasted effort in repeated NoC traversal and MAC calculation [187]. The challenge is aggravated in MAC-then-encrypt protocols because MAC can only be calculated and matched after decryption is done. If the packet is tampered, time and energy spent on decryption is wasted. To analyze these overheads, we ran FFT, RADIX (RDX), FMM and LU benchmarks from the SPLASH-2 benchmark suite on an 8×8 Mesh NoC-based SoC with 64 cores which implements a MAC-then-encrypt security protocol and XY routing protocol. The behavior of an MIP was simulated by one of the IPs along the routing path dropping n consecutive packets after every p (period) packets. NoC delay (total NoC traversal delay for all packets) including encryption/decryption and MAC calculation time, execution time and number of packets injected were recorded with and without the presence of an MIP. The encryption/decryption and authentication process is assumed to take 20 cycles per transmission [189]. Results are shown in Figure 8-2B, Figure 8-2C, and Figure 8-2D, respectively. We observed 67.2% increase in NoC delay and a 4.7% increase in execution time on average across all benchmarks. The

number of packets injected increased by 60.1%. The combination of execution time and number of injected packets directly affect the energy consumption since both time spent to execute the task and dynamic power are increased.

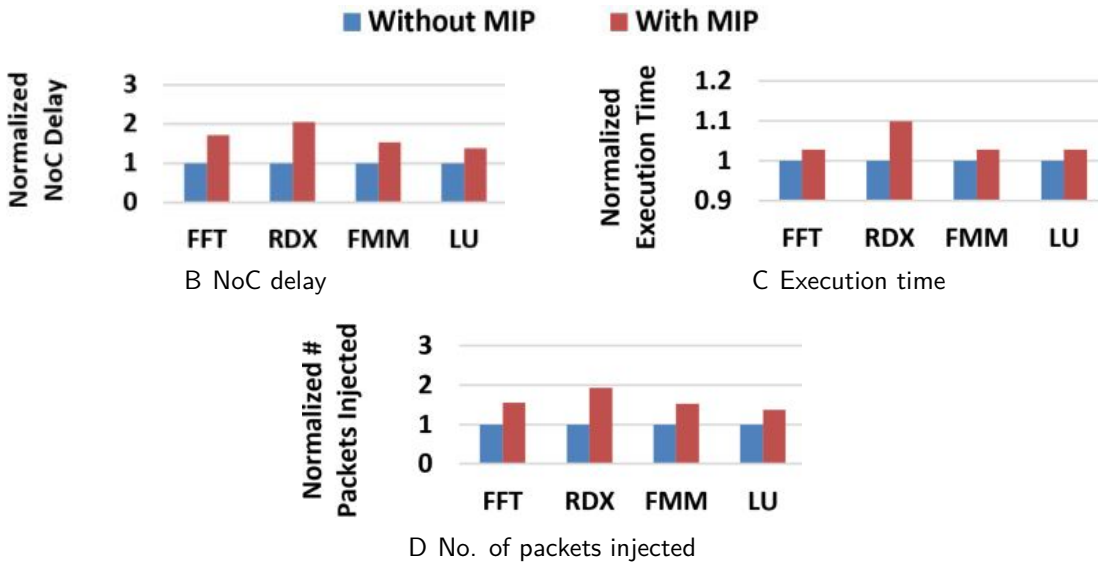


Figure 8-2. NoC delay, execution time and number of packets injected comparison with and without the presence of an MIP when $p = 20$ and $n = 14$.

It is evident that in addition to checking data integrity, a mechanism to avoid MIPs when routing through the non-secure zone can lead to less re-transmissions, and as a result, increased performance and energy efficiency.

8.2 NoC Trust Model

This section describes our proposed trust model to quantitatively measure the trust between two nodes. Trust is established between two nodes to handle packets without tampering with the data. In particular, one node trusts the other node to perform the intended action on the received packet (in the case of routing, forward the packet to the next hop). In this chapter, the first node is referred to as the "producer" (α) and the second node as the "consumer" (β). We introduce the notation $\{producer \rightarrow consumer\}$ ($\alpha \rightarrow \beta$) to denote a

trust relationship[2] . Trust can be established in two ways - (1) delegated trust, and (2) direct trust. Direct trust is established when a node calculates trust about one of its neighbors. Trust is said to be delegated when one node recommends a consumer node to another producer node that is not directly connected to the consumer. The recommending node is referred to as "recommender". Figure 8-3A shows such an example. In this three node setup, direct trust can be established between B and C, and A and B. But, trust between A and C can only be established via B's recommendation. Therefore, $A \rightarrow C$ has a delegated trust relationship.

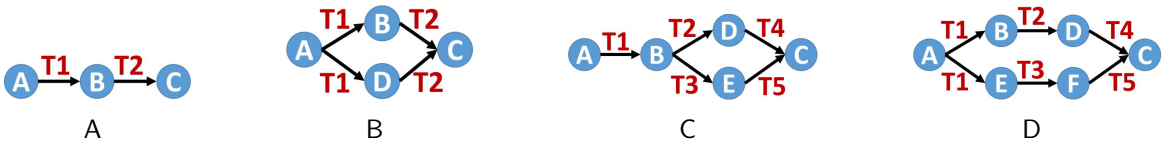


Figure 8-3. Trust delegation across NoC. The values on the arrows represent the trust. For example, $T1$ in (a) denotes $T^{(a)}_{A \rightarrow B}$ where the superscript (a) corresponds to Figure 8-3A.

To quantify trust between two entities, a measure of trust is required. Keeping a binary value per node (either trusted or not) doesn't capture the entire trust model due to several reasons: (i) Trust can be delegated (in the example in Figure 8-3A, the amount of trust A places on C depends on how much A trusts B), (ii) a malicious node might not launch an attack at first, but do so after a while or periodically. Therefore, we assign a value (denoted $T_{\alpha \rightarrow \beta}$) between -1 and 1 for each trust relationship ($-1 \leq T_{\alpha \rightarrow \beta} \leq 1$) to indicate a trust value in the "potentially malicious" spectrum. The two bounds are defined as follows:

- When the producer is confident that the consumer will always function correctly: $T_{\alpha \rightarrow \beta} = 1$.
- When the producer is confident that the consumer is definitely malicious: $T_{\alpha \rightarrow \beta} = -1$

[2] The producer and consumer notations are different from src and dest since any two routers along the routing path can be producer/consumer whereas src and dest refer to the origin of the packet and its destination, respectively.

In addition to the two bounds, $T_{\alpha\to\beta} = 0$ implies that the producer has no idea whether the consumer is malicious or not. Therefore, at the beginning of network packet transmission, all trust relationships are initialized to the value of zero. During operation, with information received from nodes, trust values are calculated. It is important to note that, when B recommends C to A (delegated trust), $T^{(a)}_{A\to C}$ can be established only if $T^{(a)}_{A\to B} \geq 0$. In other words, A should not trust its enemy to recommend someone as trustworthy. Once this condition is met, we present three axioms such that the trust delegation calculation adheres to those. The remainder of this section describes these axioms (Section 8.2.1) and elaborate how delegated trust (Section 8.2.2) and direct trust (Section 8.2.3) are calculated.

8.2.1 Axioms for Trust Delegation

Axiom 1: In delegated trust, trust value between producer and consumer should not be higher than the trust between producer and recommender as well as the trust between recommender and consumer. This can be formalized using Figure 8-3A;

$$\left|T^{(a)}_{A\to C}\right| \leq min(T^{(a)}_{A\to B}, T^{(a)}_{B\to C}) \tag{8-1}$$

Axiom 2: Producer receiving the same recommendation about the same consumer via multiple different recommenders should not reduce the trust between producer and consumer. In other words, the producer will be more certain about the consumer or at least maintain the same level of certainty if the producer obtains an extra recommendation that agrees with the producer's current opinion. For example, Figure 8-3A and Figure 8-3B show two scenarios where A in first figure establishes trust with C via only one path and in the second scenario, trust with C is established through two same-trust paths.

$$T^{(b)}_{A\to C} \geq T^{(a)}_{A\to C} \geq 0, \ for\ T1 > 0\ and\ T2 \geq 0 \tag{8-2}$$

$$T^{(b)}_{A\to C} \leq T^{(a)}_{A\to C} \leq 0, \ for\ T1 > 0\ and\ T2 < 0 \tag{8-3}$$

This holds only if the multiple paths give the same recommendations.

Axiom 3: In a setup similar to Figure 8-3C, it is possible to receive multiple recommendations from a single node (B). Compared to that, recommendations from independent nodes such as the ones shown in Figure 8-3D (B and E) should always be trusted more. In other words, recommendations from independent nodes can reduce uncertainty more effectively than the recommendations from correlated nodes. Formally;

$$T^{(d)}_{A\to C} \geq T^{(c)}_{A\to C} \geq 0,\ if\ T^{(c)}_{A\to C} \geq 0 \tag{8-4}$$

$$T^{(d)}_{A\to C} \leq T^{(c)}_{A\to C} \leq 0,\ if\ T^{(c)}_{A\to C} < 0 \tag{8-5}$$

8.2.2 Delegated Trust Calculation

The calculation of trust from the point of view of any given node should adhere to the above axioms. For the example shown in Figure 8-3A, we established that the necessary condition is to satisfy Axiom 1. To achieve this, trust can be calculated by concatenation as $T^{(a)}_{A\to C} = T^{(a)}_{A\to B} \cdot T^{(a)}_{B\to C}$. In general;

$$T_{\alpha\to\beta} = T_{\alpha\to\gamma} \cdot T_{\gamma\to\beta} \tag{8-6}$$

where γ is the recommender. As mentioned before, this can only be calculated if $T_{\alpha\to\gamma} \geq 0$. It can be noticed that if α has no idea about the trustworthiness of γ ($T_{\alpha\to\gamma} = 0$), no matter how much γ trusts β, α won't trust β ($T_{\alpha\to\beta} = 0$).

In case of multi-path trust delegation such as the example in Figure 8-3B, axioms 2 and 3 have to be satisfied in addition to Axiom 1. When α can establish trust with β via two paths, one via δ and another via ϵ ($\alpha - \delta - \beta$ and $\alpha - \epsilon - \beta$), we combine the ratios of trust concatenation.

$$T_{\alpha\to\beta} = z_1 \cdot (T_{\alpha\to\delta} \cdot T_{\delta\to\beta}) + z_2 \cdot (T_{\alpha\to\epsilon} \cdot T_{\epsilon\to\beta}) \tag{8-7}$$

where

$$z_1 = \frac{T_{\alpha\to\delta}}{T_{\alpha\to\delta} + T_{\alpha\to\epsilon}},\ and\ z_2 = \frac{T_{\alpha\to\epsilon}}{T_{\alpha\to\delta} + T_{\alpha\to\epsilon}} \tag{8-8}$$

8.2.3 Direct Trust Calculation

We calculate direct trust based on the "sigmoid function" ($\frac{1}{1+e^{-x}}$), where x keeps track of the number of successful transmissions at a given router. Since the sigmoid function ranges between 0 and 1, we scaled it to range between -1 and 1 (Figure 8-4).

$$S(x) = 2 \cdot \frac{1}{1+e^{-x}} - 1 \tag{8-9}$$

Assume that α and β are neighbors. Initially, α has no trust information about β. Therefore, $x = 0$, and as a result, $S(x) = 0$. When α learns about β's behavior, it changes the value x and re-calculates $S(x)$. For example, if α gets a positive feedback about β's trust, direct trust is calculated as $T_{\alpha\to\beta} = S(x+\delta)$ where δ is a small positive number. Since $S(x)$ is an increasing function as shown in Figure 8-4, α's trust about β is now increased. Similarly, to reduce trust, $T_{\alpha\to\beta} = S(x-\delta)$. Therefore, direct trust is calculated as;

$$x = x \pm \delta, \quad T_{\alpha\to\beta} = S(x) \tag{8-10}$$

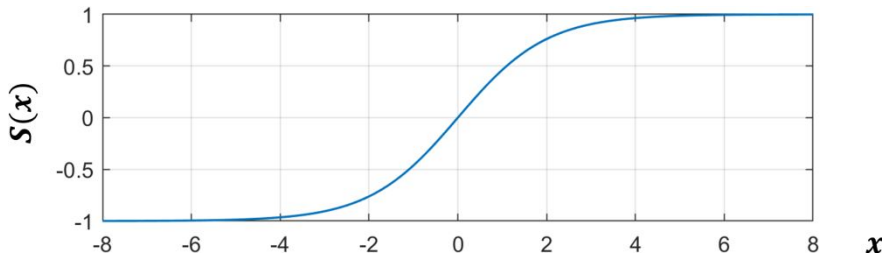


Figure 8-4. Sigmoid function $S(x)$ variation with input x.

8.3 Trust-aware Routing

Once the trust values are established, they are used by our proposed routing protocol. The basic idea is to route packets through highly trusted nodes so that MIPs are avoided. It is important to note that, trust values have to be dynamically updated during SoC execution since MIPs shift between malicious and non-malicious behavior according to our threat model. The following subsections explain in detail how direct trust and delegated trust are updated at

each router (Section 8.3.1 and Section 8.3.2, respectively) and how those trust values are used in routing (Section 8.3.3).

8.3.1 Updating Trust

According to the threat model used in this chapter, if a src IP doesn't receive a response to the packet sent, it can be because of two reasons;

- **Message was lost between src and dest**: In this case, a response is never received. The src times out after a while and re-transmits the packet. The routers along the routing path observe that this is a re-transmission and reduces the direct trust of their next-hop neighbors. Direct trust is reduced since a packet took that path before and it was tampered. Direct trust re-calculation is done every time a re-transmission is observed. Once the trust values go down compared to the other possible paths, the packet takes an alternate path avoiding the MIP according to the routing protocol and is received at the dest.
- **Response was lost between dest and src**: This means that the packet was received at dest, but the response was not received by src. Again, src sends a re-transmission which is received by dest. dest observes that this is an address that was previously served and sends the response again. Again, routers along the path observes that this is a re-transmission and reduces direct trust. This process is repeated until the response is received by src. This causes the routers between src to dest to reduce trust unnecessarily (false negative). However, we don't try to correct it because to do that, src has to keep track of all the paths the re-transmitted packets took to reset trust values. Furthermore, the routers should also maintain previous trust values. Therefore, we allow false negatives to happen. With several ongoing communications overlapped between routers, the false negatives will regain trust over time.

Considering these scenarios, we use an event-driven approach to update trust. The overview of our algorithm is shown in Algorithm 11. To keep track of the re-transmissions and to increase/decrease direct trust according to that, we implement a separate data structure at each router-"Communication Table" (ComTable). It stores each pending communication using src, dest, address of corresponding memory location (addr), timestamp to indicate when the entry was added to the table and a re-transmission flag (rtx flag). When a new packet arrives at a router, it checks to see if there is a pending communication between the same src and dest by matching src and dest fields in the packet header to entries in the ComTable (line 1). If yes, it can either be for the same address (line 3) or for a different address. If it is

for a different address, it means that the previous communication has completed successfully. If it is for the same address, then it is identified as a re-transmission. The rtx flag is set to indicate this (line 4) and direct trust with the next hop ($getNextHop$ routine elaborated in Section 8.3.3) is reduced (line 6). If it is a new communication, the rtx flag is checked to see whether the previous communication between the same src and dest has not been flagged as a re-transmission before (line 8). If it has not been flagged before, the path can be trusted. Then the direct trust with next hop router is increased (line 10) and the trust is delegated (line 11) to other neighbors as explained in Section 8.3.2. If it has already been flagged as a re-transmission, no further action is taken since it has already been penalized and as a result, direct trust has been reduced in a previous iteration (lines 4-6). In both cases, when it is a new communication, the ComTable is updated by removing the old entry and adding the new one (line 13). If it is the first communication that is passing through that router for that src and dest pair, a new entry is added in the ComTable (line 16). The ComTable also records a timestamp for each entry. The timestamp is used to stop the exponential growth of the ComTable by removing old entries after a certain time threshold.

One limitation of this model is that it assumes that an IP will only send a second request to the same destination once the first one is served. For architectures that support multiple pending requests, we can easily extend this scheme. The sender maintains a list of pending requests and adds a header bit in the next packet to indicate that this is another request with the same src, dest, but has a different address. Then, the routers check this bit before removing the previous entry and trust is increased only if this bit is not set. The rest of the methodology remains the same.

8.3.2 Delegating Trust in the NoC

Once a communication is successfully completed, trust about the next-hop ($T_{\alpha\to\beta}$) is delegated to nearby routers by each router ($delegateTrust$ routine in Algorithm 11). This is done by broadcasting a packet that contains $T_{\alpha\to\beta}$ with a pre-defined time-to-live (τ) value in the header in all directions except for the direction of the next hop router. In our experiments,

Algorithm 11 Updating direct and delegated trust

This routine is called by each router every time a packet arrives.
Input: packet
Current node is assumed to be α

```
1: entry ← checkComTable(packet)
2: if entry ≠ NULL then
3:     if entry.addr = packet.addr then
4:         entry.rtxFlag ← 1
5:         β ← getNextHop(packet)
6:         T_{α→β} ← S(x − δ)
7:     else
8:         if entry.rtxFlag ≠ 1 then
9:             β ← getNextHop(packet)
10:            T_{α→β} ← S(x + δ)
11:            delegateTrust()
12:        end if
13:        updateComTable(packet)
14:    end if
15: else
16:    updateComTable(packet)
17: end if
```

Routine:checkComTable
Input: packet

```
18: for entry ∈ comTable do
19:     if entry.src = packet.src & entry.dest = packet.dest then
20:         return entry
21:     end if
22: end for
23: return NULL
```

Routine:updateComTable
Input: packet

```
24: for entry ∈ comTable do
25:     if entry.src = packet.src & entry.dest = packet.dest then
26:         comTable.delete(entry)
27:     end if
28: end for
29: newEntry.src ← packet.src, newEntry.dest ← packet.dest
30: newEntry.addr ← packet.addr, newEntry.rtxFlag ← 0
31: newEntry.timestamp ← 0
32: comTable.add(newEntry)
```

we set $\tau = 1$. This causes the trust about the next hop router to be delegated to all other neighbouring routers. An illustrative example of this mechanism is shown in Figure 8-5. Once router α completes a communication where according to the routing protocol, the next hop router is β, it sends the direct trust value ($T_{\alpha\to\beta}$) to B, D and E. These three routers now calculate $T_{B\to\beta}$, $T_{D\to\beta}$ and $T_{E\to\beta}$, which are delegated trust values, according to the trust model in Section 8.2.2 (Equations 8-6). As a result, B, D and E learn about the trustworthiness of a router (β) two hops away from them.

It is possible that this delegated trust packet itself is tampered and in that case, delegated trust will not be updated. This has no impact since a delegated trust packet being dropped means an MIP is on that path and its trust value will be negative. Delegated trust is updated only when it comes from a trusted source with a positive trust value according to Equation 8-6.

8.3.3 Routing Protocol

The goal of the routing protocol is to avoid MIPs in the non-secure zone while routing through the most trusted routers. Each router stores the trust values of routers that are one (direct trust) and two hops away from it (delegated trust). When a router receives a packet, it first updates the trust values according to Algorithm 11. Next, the packet is forwarded to the next hop. Both forwarding and Algorithm 11 use the $getNextHop$ routine, which works as follows;

- Read the dest ID of the packet
- Compare dest and current router IDs
  - If dest is located in the same row or column as the current router, next hop is the neighbouring router along that row or column towards dest.
  - Else, check the sum of trust values of routers one and two hops towards the dest, and select the neighbor along the path which has the largest trust value as the next hop. If two paths have the same largest trust value, randomly pick one.

For example, in Figure 8-5, assume a packet arrives at router B with the destination G. Since B is not in the same row or column as G, next hop is selected based on trust values.

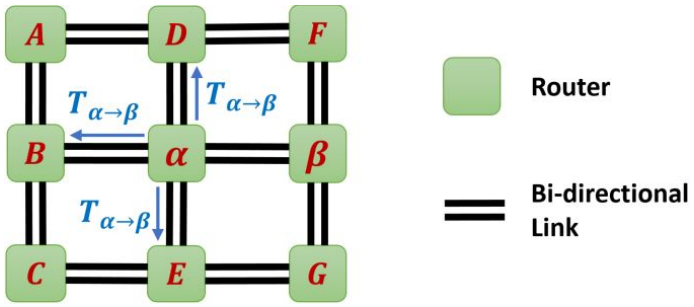


Figure 8-5. Illustrative example showing that once a communication completes, the direct trust between α and β ($T_{\alpha\to\beta}$) is delegated to nodes one hop away from α.

When considering routers that are one and two hops away from B in the direction of G, there are three possible paths: $B-\alpha-\beta$, $B-\alpha-E$, and $B-C-E$. Therefore, B calculates $max(T_{B\to\alpha}+T_{B\to\beta}, T_{B\to\alpha}+T_{B\to E}, T_{B\to C}+T_{B\to E})$ and if $T_{B\to C}+T_{B\to E}$ gives the maximum trust value, next hop is C. Considering nodes that are always towards the destination (in the example, B only considers α and C as next hops) ensures that the packet traverses the network following only one of the shortest paths. This together with the use of bi-directional links ensures the deadlock and livelock free nature of the routing algorithm. Our routing protocol is identical to the congestion-aware routing protocol presented in [190] except that we are using trust values instead of congestion values. Therefore, we can show that our routing protocol is also is also deadlock-free and livelock-free using the same arguments from [190].

It is important to note that our trust-aware routing protocol works even if all the IPs in the non-secure zone are malicious or, MIPs isolate the untrusted zone into several disconnected sub zones of secure IPs. If all the neighbors of a router has a trust value of -1 (all routers are malicious) it will still be routed through that path since -1 is the largest value. Therefore, the packet is guaranteed to reach the destination, but might be corrupted. If there is a path from source to destination that does not contain an MIP, our approach is guaranteed to find that path and deliver the packets without being corrupted.

8.4 Experiments

This section explores the feasibility and effectiveness of our approach by presenting experimental results and discussing the overheads associated with it.

8.4.1 Experimental Setup

We modeled an 8×8 Mesh NoC-based SoC with 64 cores using the gem5 cycle-accurate full-system simulator [152, 153]. The interconnection network was built on top of "GARNET2.0" model that is integrated with gem5 [147]. Each router in the Mesh topology connects to four neighbors and a local IP via bidirectional links. Each IP connects to the local router through a network interface, which implements the MAC-then-encrypt protocol. The default XY routing protocol was modified to implement our trust-aware routing protocol. In our experiments, we used $\delta = 0.5$ (Equation 8-10) when increasing/reducing direct trust. The value 0.5 was chosen experimentally such that the algorithm chooses alternative paths as quickly as possible while minimizing the impact of false negatives.

We tested the system using 4 real benchmarks (FFT, RADIX, FMM, LU) from the SPLASH-2 benchmark suite and 7 synthetic traffic patterns (uniform random (URD), tornado (TRD), bit complement (BCT), bit reverse (BRS), bit rotation (BRT), shuffle (SHF), transpose (TPS)). When running both real benchmarks and synthetic traffic patterns, each IP in the top (first) row of the Mesh NoC instantiated an instance of the task. Real benchmarks used 8 memory controllers that provide the interface to off-chip memory which were connected to the bottom eight IPs. As synthetic traffic patterns don't use memory controllers, the destination of injected packets were selected based on the traffic pattern. For example, uniform random selected the destination from the IPs at the bottom row with equal probability. Source and destination modelling was done this way to mimic the secure and non-secure zones. Four MIPs were modeled and assigned at random to IPs in the other six rows. To simulate the sporadic behavior of the MIPs as discussed in the threat model, each MIP corrupted n consecutive packets after every p (period) packets. According to our architecture model, the IPs in the top row (secure zone) communicate with the IPs in the bottom row (secure zone) through the other 6 rows (non-secure zone) of IPs out of which, 4 are malicious. Our approach will work the same for any other secure, non-secure zone selection and MIP placement. The

output of the gem5 simulation statistics was fed to the McPAT power modelling framework to obtain power consumption [151].

8.4.2 Performance Improvement

Figure 8-6 shows results related to the performance improvement when running real benchmarks. The figure compares performance results without the presence of MIPs (Without MIP), with the presence of MIPs when default XY routing is used (With MIP-Default), and when our approach is used with the presence of MIPs (With MIP-Our Approach). We can observe that our approach reduces NoC delay by 53% (43.6% on average) compared to the default XY routing protocol. Execution time and number of packets injected are reduced by 9% (4.7% on average) and 71.8% (66% on average), respectively. When the MIPs corrupt packets, re-transmissions are caused and its trust is reduced. As a result, alternative paths are chosen. The performance improvement depends on how quickly the algorithm chooses an alternative path once an attack is initiated.

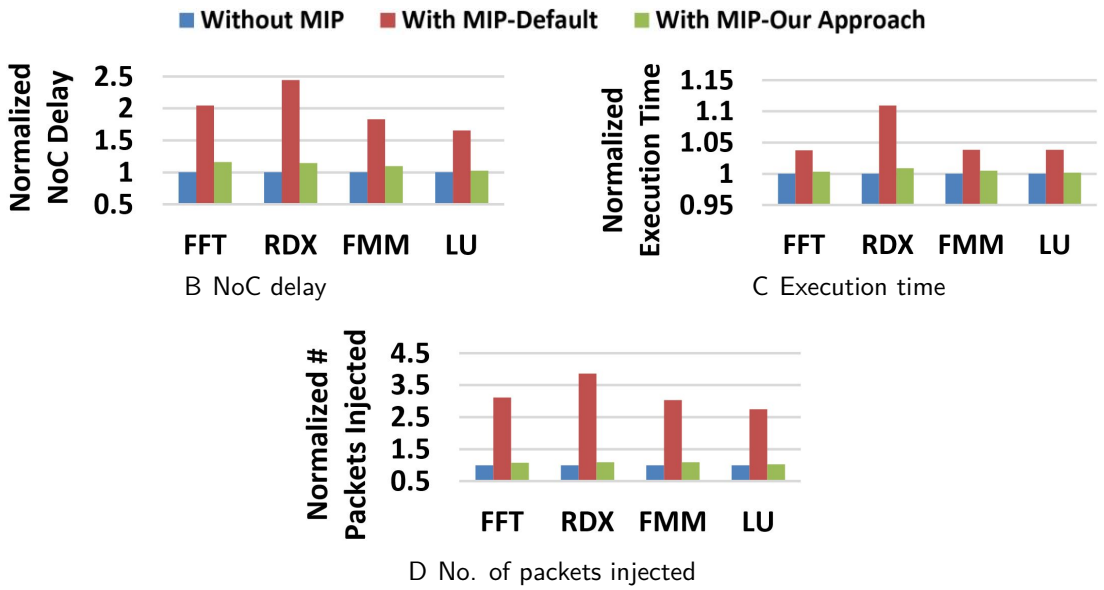


Figure 8-6. NoC delay, execution time and number of packets injected with and without our trust-aware routing model when running real benchmarks. $p = 20$ and $n = 14$.

In addition to real benchmarks, we experimented with synthetic traffic traces as well. Results related to synthetic traffic patterns are shown in Figure 8-7. The comparison is the

same as that of Figure 8-6. It shows that NoC delay and number of packets injected on the NoC are reduced by 57.1% (51.2% on average) and 56.7% (50.1% on average), respectively.

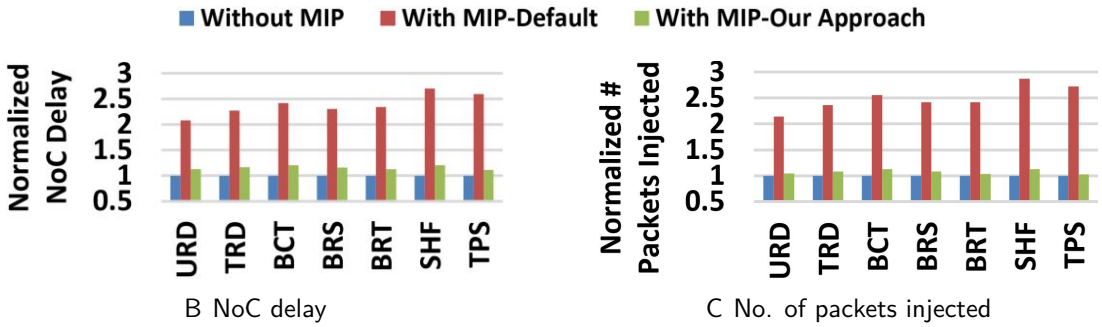


Figure 8-7. Execution time and number of packets injected with and without our trust-aware routing model when running synthetic traffic patterns. $p = 20$, $n = 14$.

8.4.3 Energy Efficiency Improvement

As a result of reduced execution time and reduced number of re-transmissions, the energy consumption of the SoC also reduces. Figure 8-8 shows the energy consumption comparison. Note that 47.4% (28.3% on average) less energy is consumed by real benchmarks when routing using our approach compared to the default XY routing in the presence of MIPs. Synthetic traffic demonstrate energy savings of up to 75.6% (67.6% on average). Compared to real benchmarks, synthetic traffic patterns show more energy reduction since synthetic traffic focuses only on network traversals unlike real benchmarks which goes through the entire processor pipeline including instruction execution, NoC traversal and memory operations.

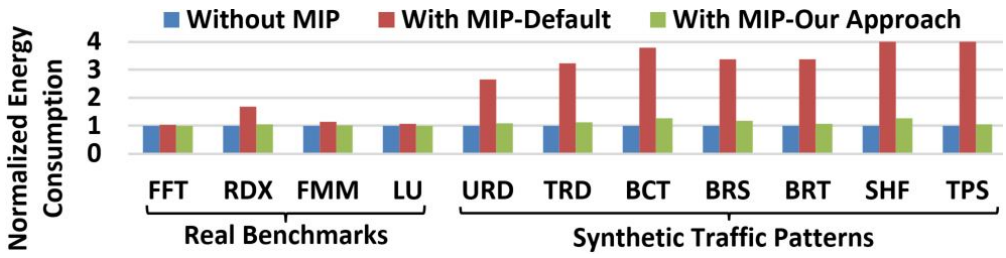


Figure 8-8. Energy consumption with and without our trust-aware routing model when running real benchmarks and synthetic traffic patterns. $p = 20$, $n = 14$.

8.4.4 Overhead Analysis

To implement our routing protocol, additional hardware is required at each router. This includes extra memory to store trust values and hardware to calculate, update and propagate trust. To accommodate a row in the ComTable, 10 bytes of memory is required (6-bit src, 6-bit dest, 32-bit addr, 1-bit rtx flag, 32-bit timestamp). The maximum size of the ComTable during our experiments was 24. This leads to 240 bytes of extra memory requirement per router.

We used the default 5-stage router pipeline (buffer write, virtual channel allocation, switch allocation, switch traversal and link traversal) implemented in gem5. Once separate hardware is implemented, computations related to trust can be carried out in a pipelined fashion in parallel to the computations in the router pipeline. To evaluate the area overhead, we modified the RTL design of an open source NoC router [185] and synthesized the design with 180nm GSCLib library from Cadence using Synopsis Design Compiler. This resulted in an area overhead of 6% compared to the default router. This shows that the proposed trust-aware routing protocol is lightweight and can be effectively implemented at routers in an NoC-based SoC.

8.5 Summary

In this chapter, I proposed a trust-aware routing protocol that is capable of routing packets by avoiding malicious IPs in NoC-based SoCs. The routing protocol is implemented based on a trust model that calculates how much a neighboring node can be trusted to route packets through that router. The experiments conducted by using both real benchmarks and synthetic traffic patterns demonstrated significant performance and energy efficiency improvements compared to traditional XY routing in the presence of a MAC-then-encrypt security protocol. Overhead analysis has revealed that the area overhead to implement the routing protocol is only 6%. This approach can be integrated with any existing authentication scheme as well as other threat mitigation techniques, to secure the SoC while minimizing the performance and energy efficiency degradation caused by a malicious IP tampering packets.

CHAPTER 9
RECONFIGURABLE NETWORK-ON-CHIP SECURITY ARCHITECTURE

In the early days of IoT and embedded devices in general, they were intended for a single or very few use cases. The requirements and working conditions were well defined and predictable. Therefore, it was easy to make design choices to fit the requirements. For example, a device for a power-thrifty application was designed to conserve power at the cost of performance, while a high-performance system exhibited a different, yet predictable trade-off. In comparison to that, the devices manufactured today are intended to serve general purpose applications that are diverse and sometimes, not yet defined. Therefore, it is not possible to statically optimize the devices to fit each use case.

Furthermore, in the pre-IoT era, before devices being integrated to our everyday lives, the devices only required to last for a few years. In case of a phone or a personal computer, new features will come into products within few years, or even few months and the previous models will become out-dated. In contrast, if you are building a smart house or a smart grid, you expect them to last well over 10 years. However, the requirements of a smart system over a long life-span of 10+ years can change drastically. For example, a car equipped with the state-of-the-art security mechanisms will be secure in the present-day, but will not be secure against future attacks. The system is secure until the zero-day vulnerability is exposed. Clearly, IoT devices must be adaptable on-field to changing application requirements.

The remainder of the chapter is organized as follows. Section 9.1 motivates the need for reconfigurable security architecture. Section 9.2 outlines the threat model. Section 9.3 introduces some relevant background information. Section 9.4 explains our reconfigurable security architecture. Section 9.5 presents the experimental results. Finally, Section 9.6 summarizes the chapter.

9.1 Motivation

It is evident that securing IoT devices based on complex SoCs throughout the device lifetime among changing requirements and use-case scenarios should be considered during

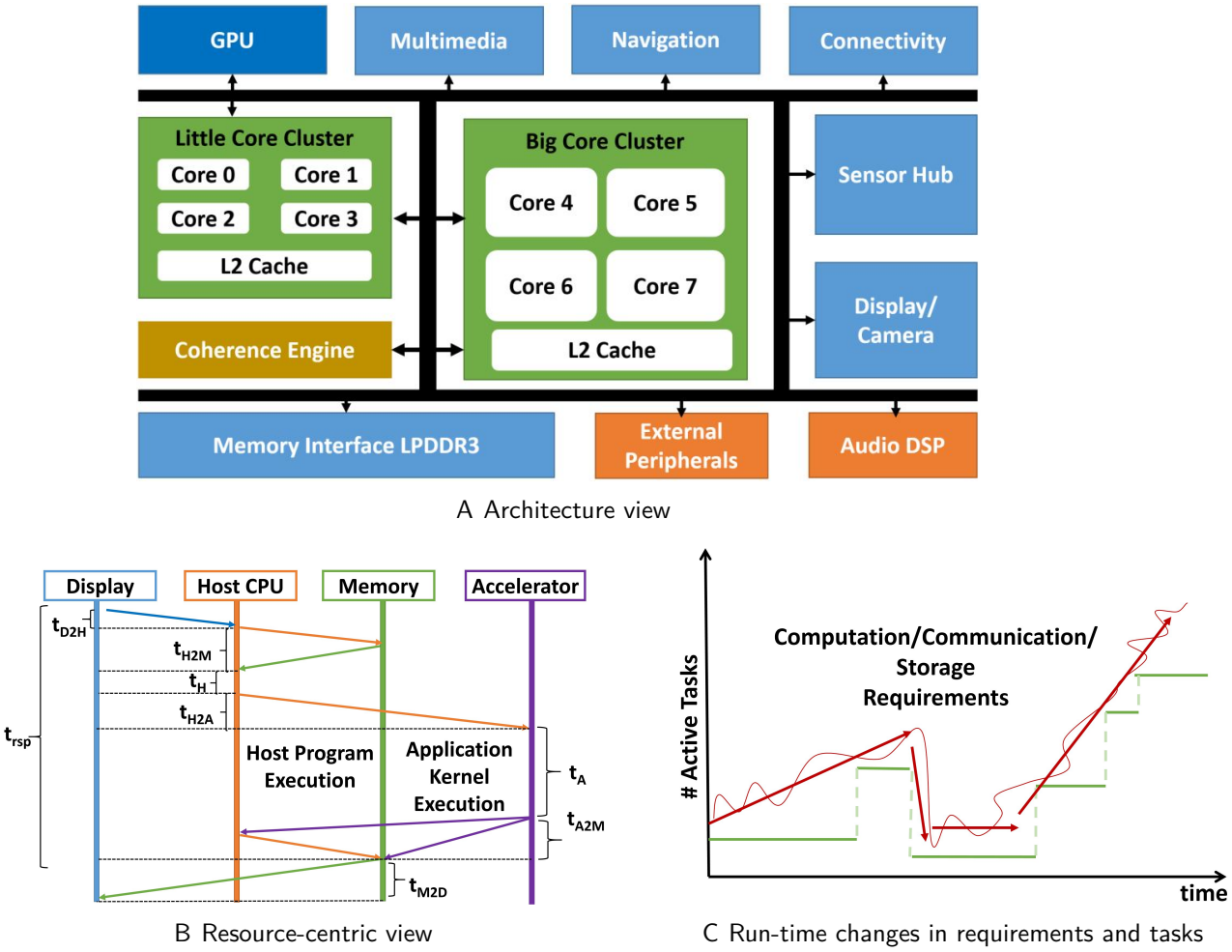


Figure 9-1. Dynamic changes in IoT application characteristics: (a) an example IoT SoC, (b) application flow, and (c) run-time change in requirements.

design time. Due to the resource-constrained nature of IoT SoCs, it is not always feasible to enforce the strongest security mechanisms. Security has to be considered among other interoperability constraints such as performance, power and area overheads. Besides, employing the full security arsenal may not be required depending on the application characteristics and use-case scenarios. For example, consider a smart watch that is used for browsing the Internet at home as well as in a public coffee shop. It may be okay to trust the wireless network at home and impose a light-weight security requirement in favor of a lower energy profile. However, a stronger security mechanism is necessary when communicating with the untrusted network in a coffee shop at the cost of power and performance. On the other

hand, if the current state of the device battery is low, it might be desirable to compromise on security and save more power to ensure application execution. The trade-off between performance and energy is also integrated in modern-day smart phones by the introduction of "power-saver" modes. Similarly, the discussion on security among other interoperability constraints is required.

According to our threat model, the security threat comes from the malicious IPs (MIP) integrated on the SoC. Due to mass production and tight time-to-market deadlines, most SoC manufacturers outsource IP cores to third party vendors. These third party manufacturers are not always trustworthy. Their IPs might contain hardware Trojans and other malicious implants that can launch active as well as passive attacks on other legitimate components on the SoC once they are activated. We call these third party IPs "potentially malicious IPs". Due to the distributed nature of the NoC, MIPs use resources offered by the NoC to launch attacks [113]. To capture these scenarios, we use an architecture and threat model as described below.

9.2 Architecture and Threat Models:

In our work, we use an architecture model similar to the one shown in Figure 9-2. It shows an NoC-based SoC divided into secure and non-secure zones similar to the architecture proposed in the ARM TrustZone architecture [191]. The secure zone comprises of IPs we can trust to not contain malicious implants (secure IPs) and the non-secure zone contains IPs obtained by third party vendors (potentially malicious IPs), which cannot be trusted. An IP in one secure zone (top left) communicates secure information with a secure IP in the other zone (bottom right). Since the packets traverse through the non-secure zone, the presence of a MIP can pose a security threat.

Depending on increasing capabilities of MIPs, we divide the threats into tiers. Each tier is assumed to include the capabilities of the previous tier. For example, a MIP classified in tier 3 has capabilities of tier 1 and 2 as well.

Tier 1:

- MIPs can eavesdrop on the packets traversing through the network

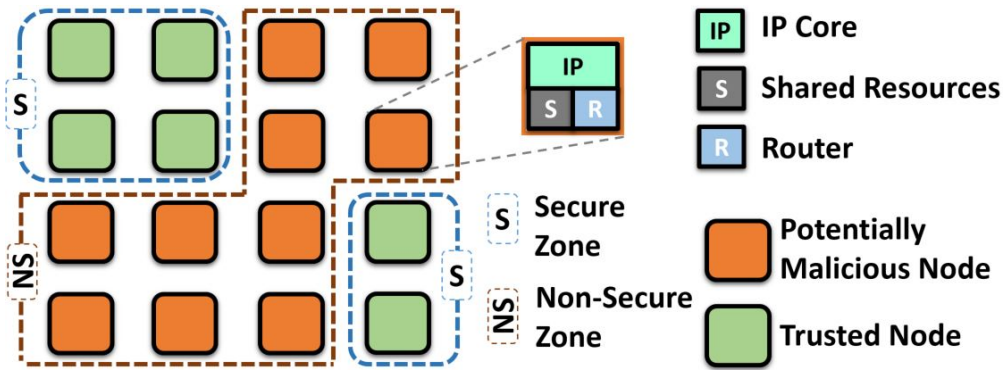


Figure 9-2. Overview of a typical SoC architecture with secure and non-secure zones.

- Copied packets can cause information leakage.

Tier 2:

- MIPs can corrupt/spoof packets. Corrupted packets can lead to the erroneous execution of programs as well as system failures.
- Spoofed packets inject new packets to the network causing system to malfunction.
- Packets can be re-routed to MIPs to leak information.

Tier 3:

- MIPs can launch denial-of-service (DoS) attacks on a critical component of the SoC causing significant performance degradation.

We propose our reconfigurable security architecture to secure the SoC against these different capabilities of MIPs depending on the usage scenario. The goal is to ensure secure communication between secure IPs and to prevent any attacks. Major contributions of this chapter are as follows:

- I propose a reconfigurable fabric that would enable utilization of security primitives in a plug-and-play manner based on application requirements.
- I implement a tier-based security architecture that allows reconfigurable security. Solutions proposed for each tier are countermeasures for the capabilities of MIPs at each tier. An overview of possible attacks and corresponding countermeasures is shown in Figure 9-3.
- I show that the security architecture can be dynamically reconfigured based on changing application requirements.

- I implement these mechanisms on an NoC-based SoC, and evaluate the efficiency of different security tiers in terms of performance, energy and area. Then, I discuss how different levels of security can be used depending on the use-case scenario.

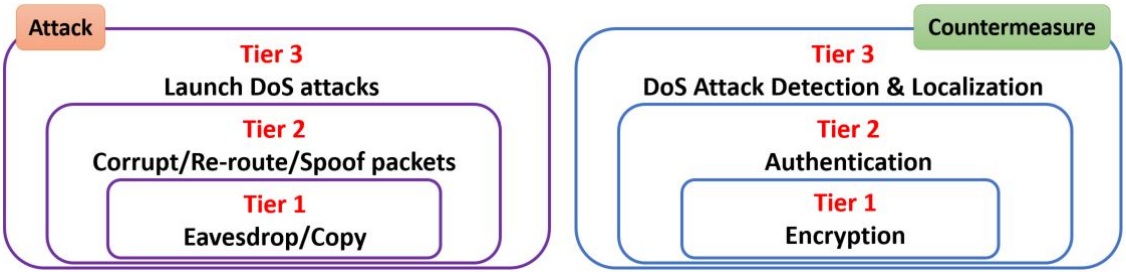


Figure 9-3. Potential attacks and corresponding countermeasures in the tier-based security architecture.

The primary objective of the chapter is not to improve any of the security tiers when taken separately. Instead, the goal is to introduce a framework to integrate them together and discuss pros and cons of activating each one against the other.

9.3 Background

This section introduces some key concepts used to implement our reconfigurable security architecture.

9.3.1 Block Cipher Based Symmetric Encryption

In symmetric encryption, both encryption and decryption are done using the same key ($\mathcal{K}$). Let $\mathcal{E}$ denote the encryption algorithm. If the message to be encrypted is M, the ciphertext C is produced by taking the key K and a plaintext M as inputs. This is denoted by $C \leftarrow \mathcal{E}_{\mathcal{K}}(M)$. Decryption algorithm $\mathcal{D}$ performs the inverse operation to recover the plaintext denoted by $M \leftarrow \mathcal{D}_{\mathcal{K}}(C)$. Based on input type, encryption algorithms are divided into two categories, Block Ciphers and Stream Ciphers. In block cipher based encryption schemes, the encryption algorithm comprises of one or more block ciphers. Formally, a block cipher is a function E that takes a β-bit key K and an n-bit plaintext m and outputs an n-bit long ciphertext c. The values of β and n depends on the design and are fixed for a given block cipher. To encrypt M using block ciphers, M of a given length is divided into n-bit substrings where n is called the block size ($n = |m|$). Each block cipher encrypts an n-bit plaintext

m and concatenates the outputs at the end to create the ciphertext C corresponding to M. The arrangement of block ciphers is defined by the mode of operation used in the encryption scheme. Electronic Code Book (ECB) [192], Cipher Block Chaining (CBC) [193] and Counter Mode (CM) [162] are three common block cipher modes of operation.

9.3.2 Hashing

Unlike encryption that relies on the ability to "reverse" (decrypt) the encrypted data to produce the plaintext, hashing data makes it extremely difficult to reverse. In fact, the security of a hash function relies on the output being computationally hard to reverse (known as pre-image resistance) and the hash function being collision resistant [194]. A hash function is a mathematical function that takes a key H and data to be hashed α as inputs and produces a hash digest Δ as the output denoted by $\Delta \leftarrow \mathcal{H}(H, \alpha)$. A typical hash function produces a fixed-length digest irrespective of the size of the input data.

9.4 Reconfiguration of NoC Security Primitives

This section presents our proposed reconfigurable security architecture. It consists of a reconfigurable security engine (RSE) which is a dedicated IP on the SoC and security mechanisms implemented at routers and network interfaces (NI) as outlined in Section 9.4.1. While there are many security primitives, we consider three commonly utilized security primitives in NoCs (encryption, authentication and DoS attack prevention). The security tiers are selected together with relevant parameters. We define a set of parameters that have been proposed in existing literature as well as parameters that became reconfigurable due to our architecture. Each security tier is associated with the reconfigurable parameters as shown in Table 9-1.

Table 9-1. Security primitives and corresponding reconfigurable parameters.

Security Primitive	Reconfigurable Parameters
Encryption (Tier 1)	Blockcipher, Key size, Block size, IV length
Authentication (Tier 2)	Hash function, Key size, Input size
DoS attack detection and localization (Tier 3)	Detection only/Detection and localization, Detection interval

The reconfigurable parameters in encryption and authentication are self-explanatory and have been discussed in [195–197]. Tier 3 allows decoupling of DoS attack detection and localization. If detection only is selected, the SoC will detect an ongoing DoS attack, but not localize the malicious IP, whereas the other option enables both detection as well as localization. The detection interval defines the duration which the detection mechanism is active. It can be always active leading to quick detection of DoS attacks, or can be periodically active to save power. The following sections describe each of these components in detail. Section 9.4.1 describes our reconfigurable security architecture. The next three sections present reconfigurable encryption, authentication and DoS attack detection & localization mechanisms used in our architecture, respectively.

9.4.1 Reconfigurable Security Architecture

Our reconfigurable security architecture has two main parts. (i) Tier-based security countermeasures and (ii) a reconfiguration mechanism. Figure 9-4 shows how the security countermeasures are integrated in the NoC. The encryption and authentication tiers are integrated in the NI whereas dedicated hardware for DoS attack detection and localization is implemented in each router and IP. Different tiers of security and their capabilities are outlined in Figure 9-3. The reconfiguration mechanism decides which security tier to activate and which parameters to pass to the selected tier based on the system characteristics as well as security requirements. Security tiers and parameters are selected using the reconfiguration registers (RRG) integrated in each NI which are modified by the reconfiguration mechanism.

The reconfiguration mechanism has two types of components integrated in the SoC:

1. **Security Agent (SAG)**: a security agent is integrated in each NI. SAGs monitor the network for potential security attacks and also check the system characteristics such as NoC congestion and battery life through sensors.

2. **Reconfigurable Security Engine (RSE)**: a dedicated IP integrated on the SoC that contains security policies and takes decisions on when to reconfigure security based on the data given by SAGs.

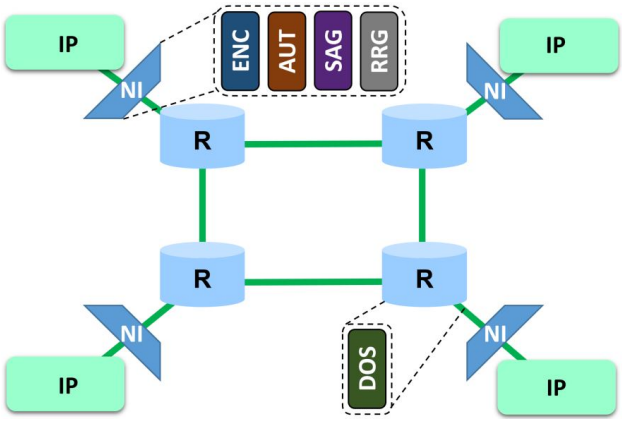


Figure 9-4. Additional hardware implemented at NIs and routers to facilitate our reconfigurable security architecture.

Figure 9-5 shows an overview of how the RSE is connected on the SoC and Figure 9-4 shows how the SAGs are integrated into each NI. The SAGs offer three different services:

1. Gather data about system characteristics such as battery level and NoC congestion.
2. Pass messages received by security tiers to RSE. For example, if an ongoing DoS attack is detected, SAGs send that information to RSE which can take the decision on activating the localization component of security tier 3 to localize the attack.
3. Set the RRG in each NI to indicate which tier of security to activate according to the decisions made by the RSE.

Algorithm 12 describes the main steps of reconfiguration. The RSE periodically pings the SAGs to gather data about system characteristics (line 2). This is called the "security heartbeat". After gathering battery level and NoC congestion information, the RSE then decides which security tier and parameters to activate based on its security policy and passes that data to the SAGs, who set the RRGs (lines 11-14). In addition to decisions made at each security heartbeat, an SAG can also interrupt the RSE if a potential security threat is detected (line 6). RSE will follow the same process and set the RRGs. RSE and SAGs communicate using a separate NoC, named the "service NoC", that facilitates all packets transferred between the RSE and SAGs without interfering with the data transferred between IPs. The IPs read the RRGs to identify which security tier to activate together with its parameters and configures

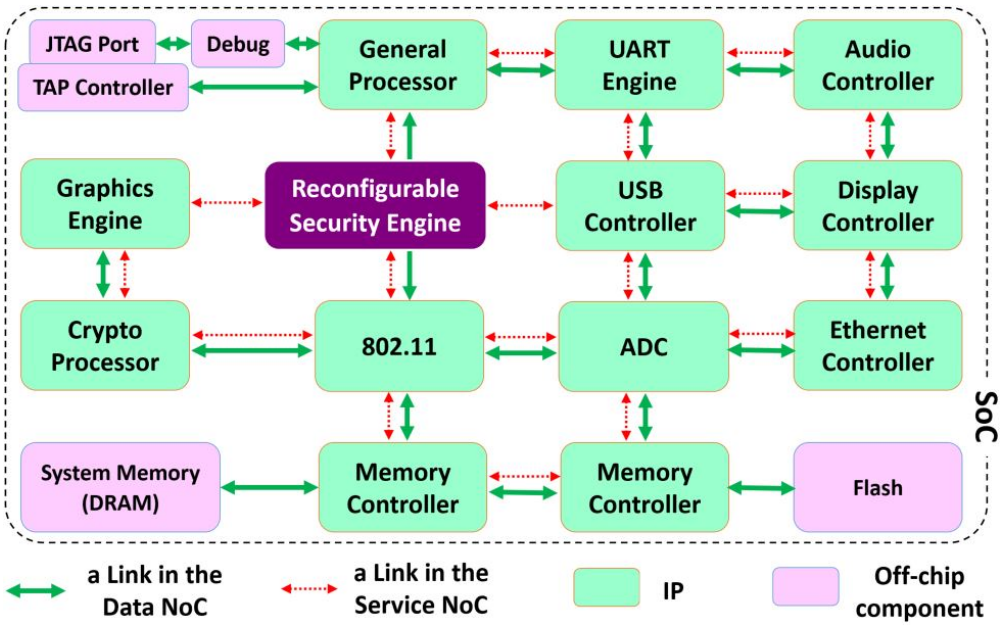


Figure 9-5. Example SoC including an RSE. Figure 9-4 shows a zoomed-in and more detailed version of the same architecture considering only four IPs.

security accordingly (lines 15-24). When a packet is injected into the NoC, it first goes through the security mechanisms depending on what tier is activated (lines 25-27).

This approach allows easy decoupling of the RSE, SAGs, security policy, security tiers and reconfigurable parameters so that each component can be modified independently at design time depending on the system requirements. The next three sections describe the components of tier-based security countermeasures - encryption (Section 9.4.2), authentication (Section 9.4.3) and DoS attack detection & localization (Section 9.4.4). A list of notations used to illustrate our approach is listed in Table 9-2.

9.4.2 Reconfigurable Encryption

To encrypt packets in real-time embedded systems, the encryption scheme should support high-speed encryption with low costs and latency. To achieve this, the operation mode of the encryption scheme must support pipelined and parallelized implementations. Furthemore, due to the nature of packets transferred and routing protocols used in the NoC, some of the packet fields such as addresses, sequence numbers and ports need to be transferred in plaintext. These fields are mainly the header fields of the packet. This leads to the requirement of an AEAD

Table 9-2. Notations used to illustrate our approach.

Notation	Description
$E_K(M)$	a message M encrypted using the key K
$A \parallel B$	concatenation of two bit strings A and B
$A \oplus B$	bitwise XOR of two bit strings A and B
$\{q\}_d$	d-bit representation of binary value q (e.g., if $d = 4$, $\{1\}_d = 0001$)
$MSB_u(S)$	gives the most significant (leftmost) u bits of S
$len(A)$	number of bits in A
0^u	string of u zero bits
$X \cdot Y$	multiplication of two elements $X, Y \in \quad GF(0^n)$ where GF corresponds to a Galois Field.

(Authenticated Encryption with Associated Data) scheme. According to our threat model and proposed tier-based security model, encryption and authentication should be decoupled. Therefore, an authenticated encryption scheme that allows isolation of the two stages is required. Furthermore, the architecture should allow easy plug-and-play of security primitives that allows the selection of reconfigurable parameters. To cater to these requirements, we use the "Counter Mode (CM)" in our experiments. Figure 9-6 shows an overview of CM including both encryption and authentication components. It is evident from the setup that the framework supports easy decoupling of encryption and authentication allowing activation of encryption only (tier 1), or both encryption and authentication (tier 2) through the values written in RRGs during runtime. In this section, we present how the encryption scheme in CM is implemented in NoC and Section 9.4.3 describes the NoC implementation of authentication.

Let $m_1, m_2, ..., m_{b-1}, m_b^*$ denote a sequence of b bit strings that construct the plaintext. Each bit string in the sequence, also known as a data block, has length n, except for m_b^* with length u where $1 \leq u \leq n$. This gives that the total length of plaintext in bits is $(b-1) \times n + u$. The ciphertext associated with this sequence follows the form $c_1, c_2, ..., c_{b-1}, c_b^*$ where each block is n bits long except for the final block c_b^* which is u bits long.

The encryption algorithm is shown in Algorithm 13. Each blockcipher in CM encrypts the string $IV \parallel \{q\}_d$ using a symmetric key K (line 5) where IV refers to the initialization vector

Algorithm 12 Main steps in security reconfiguration

```
    Send security heartbeat periodically
 1: if timer > securityHeatBeatPeriod then
 2:     send security heartbeat to all SAGs and gather data -D
 3:     reconfigureSecurity(D)
 4:     restartTimer()
 5: end if
 6: if upon event potentialAttack == TRUE: then
 7:     get data sent by SAG - D                    ▷ get data sent by SAG with interrupt
 8:     reconfigureSecurity(D)
 9:     restartTimer()
10: end if
    Major steps of reconfigure security function
11: procedure RECONFIGURESECURITY(D)
12:     T_n, P_n ← selectSecurityTier(D)    ▷ select one from T_1, T_2, T_3 and relevant parameters
13:     send selected security tier (T_n) and parameters (P_n) to SAGs
14:     setReconfigurationRegisters(T_n, P_n)
15:     T_n, P_n ← readReconfigurationRegisters()
16:     if T_n == T_1: then
17:         Q ← encryption(P_n)
18:     else if T_n == T_2: then
19:         Q ← encryption(P_n) + authentication(P_n)
20:     else if T_n == T_3: then
21:         Q ← encryption(P_n) + authentication(P_n)
22:         monitorDoS(P_n)
23:     end if
24: end procedure
    Send packets in to the NoC
25: procedure SENDPACKETS(M)
26:     send Q(M)
27: end procedure
```

which is a nonce[1] . The output r_q of the block cipher is XORed with the plaintext m_q sent to that block (line 6). The final block, which can potentially have less bits (u), is XORed with the most significant u bits of the block cipher output (line 8). The outputs are concatenated to create the ciphertext of length $(b-1) \times n + u$ (line 9). The decryption process is the exact

[1] a nonce is a random string which is distinct for each invocation of the encryption operation for a fixed key.

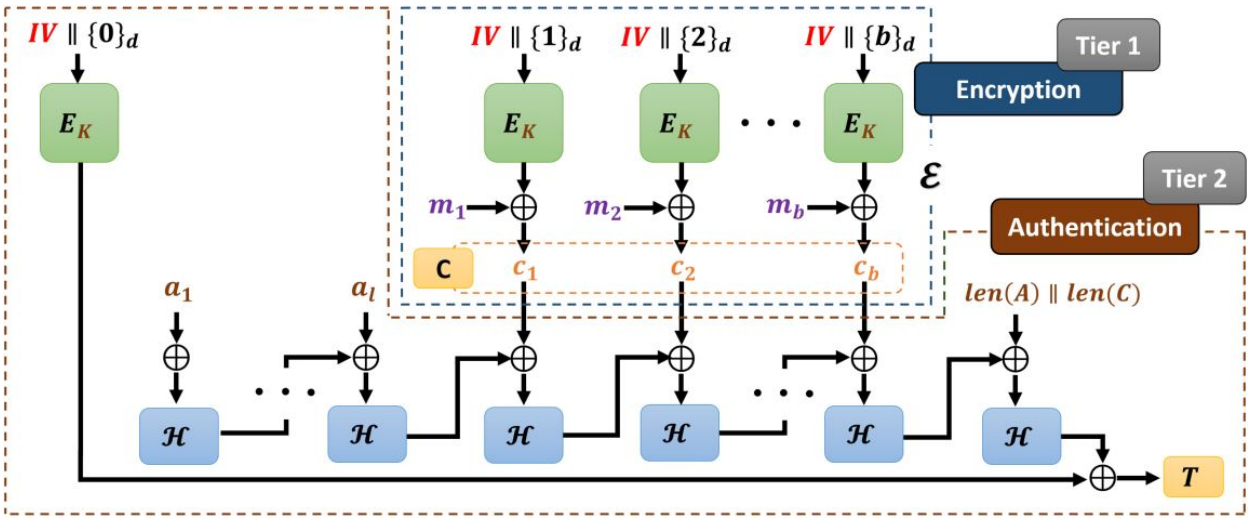


Figure 9-6. Encryption and Authentication in CM

inverse of this and is omitted in this chapter to save space. In our experiments, we used the Galois Counter Mode (GCM) which uses the same setup and AES as the block cipher together with Galoish hash as the hash function. Complete details of GCM can be found in [198].

Algorithm 13 Encryption in counter mode

Inputs: plaintext to encrypt $M = m_1 \parallel m_2 \parallel ... \parallel m_b^*$
Output: ciphertext corresponding to the plaintext C
procedure ENCRYPTION(P_n = {block cipher E_K, key K, Initialization Vector IV})
    for $q = 1, ..., b-1$ **do**
        $r_q \leftarrow E_K(IV \parallel \{q\}_d)$
        $c_q \leftarrow r_q \oplus m_q$
    end for
    $c_b^* \leftarrow m_b^* \oplus MSB_u(E_K(IV \parallel \{b\}_d))$
    $C \leftarrow c_1 \parallel c_2 \parallel ... \parallel c_b^*$
    return C
end procedure

9.4.3 Reconfigurable Authentication

While encryption makes sure that an eavesdropper cannot read the sensitive data, authentication is required to ensure that the adversary doesn't corrupt/spoof the packets. To address this, we use a hash-based Message Authentication Code (HMAC). With HMAC, the receiver is able to verify a message by checking a tag appended to the end of the packet by the source. The receiver can re-compute the original authentication tag and check whether

both tags match to see that the message has not been changed during NoC traversal. The tag is computed by using a hash function that takes the message to be authenticated and a key as inputs. In our approach, since the encrypted data is used as a part of the message to be authenticated, we are following the Encrypt-then-MAC authentication technique, which is more secure than Encrypt-and-MAC [199].

In our AEAD scheme, the associated data was not included in the encryption process. However, associated data (A) should be used when calculating the tag. Similar to M and C, A can also be denoted as a sequence of bit strings $a_1, a_2, ..., a_{l-1}, a_l^*$. Each bit string in A has a length of n, except for the last block, a_l^*, with length v, where $1 \leq v \leq n$. It follows that a_l^* can be a partial block and the total length of A in bits is $(l-1) \times n + v$. If $S_q = IV \parallel \{q\}_d$, the authentication tag (T) can be calculated as;

$$T = MSB_t(\mathcal{H}(H, A, C) \oplus E_K(S_0)) \tag{9-1}$$

where $|T| = t$ and $\mathcal{H}(H, A, C) = X_{l+b+1}$ [198]. The variable X_i for $i = 0, 1, ..., l+b+1$ is defined as;

$$X_i = \begin{cases} 0 & \text{for } i = 0 \\ (X_{i-1} \oplus a_i) \cdot H & \text{for } i = 1, ..., l-1 \\ (X_{l-1} \oplus (a_l^* \parallel 0^{b-v})) \cdot H & \text{for } i = l \\ (X_{i-1} \oplus c_i) \cdot H) & \text{for } i = l+1, ..., l+b-1 \\ (X_{l+b-1} \oplus c_l^* \parallel 0^{b-u}) \cdot H & \text{for } i = l+b \\ (X_{l+b} \oplus (len(A) \parallel len(C)) \cdot H) & \text{for } i = l+b+1 \end{cases}$$

where $\mathcal{H}$ is the hash function that takes the hash key H as one of the inputs. The t-bit tag is then appended to the packet and injected into the network. Any tampering done to the packet will cause the tag verification at the receiver's end to fail resulting in the packet being discarded and a re-transmission of the packet from the source. This will make sure

that the corrupted/spoofed packets will be discarded by NIs before they reach the IPs. The tag calculation method given in Equation 9-1 is according to the Galois hash function used in our experiments. Other commonly used hash functions for message authentication include SHA-256 [200] and MD5 [201].

9.4.4 Reconfigurable DoS Attack Detection and Localization

We implement the DoS attack detection and localization mechanism proposed in [85]. The previous work is a monolithic functionality, whereas in this chapter, we propose a reconfigurable DoS attack detection and localization algorithm. Moreover, we integrate it with reconfigurable encryption and authentication. The basic idea is to statistically analyze network traffic and to model communication patterns. Using the model, two curves are obtained that capture system characteristics. (i) Upper bounds of packet arrival curves (PAC) are calculated at each router and (ii) destination packet latency curves (DLC) are constructed at each IP. PAC bounds are used to detect DoS attacks and once an attack is detected, DLCs are used to localize the MIP.

An overview of this approach together with the parameters passed from our reconfigurable architecture is shown in Algorithm 14. The detection and localization mechanisms are implemented in such a way that localization can be disabled without affecting the detection mechanism. This is defined by the "DoSTier" parameter. In that case, an ongoing attack will be detected, but the MIP will not be localized. While this approach gives better performance and energy efficiency, the MIP can launch the attack again unless it is diagnosed separately. Similarly, to achieve improved energy efficiency, the detection mechanism at the routers can sleep periodically. The sleep time is defined by the "detectionInterval" parameter. In such a scenario, energy efficiency will improve while compromising with delays in DoS attack detection. An SoC running tasks with soft-deadlines can afford to have a DoS attack detection mechanism that is not always active. However, if there are tasks with hard-deadlines, it is better to detect immediately (no sleeping) to avoid delays caused by DoS attacks. The next

two subsections describe the two major steps - detection and localization as well as PACs and DLCs in detail.

Algorithm 14 DoS attack detection and localization mechanism

```
1: procedure MONITORDOS(parameters P_n = {DoSTier, detectionInterval})
2:     if DoSTier == detectOnly then
3:         detectDoS(detectionInterval)      ▷ start packet monitoring at routers and check for PAC bound violations
4:     end if
5:     if DoSTier == detectAndLocalize then
6:         detectDoS(detectionInterval)      ▷ start packet monitoring at routers and check for PAC bound violations
7:         localizeDoS()  ▷ if a potential attack is detected, initiate localization mechanism to pinpoint the MIP
8:     end if
9: end procedure
```

9.4.4.1 DoS attack detection using PAC bounds

Upon arriving at a router (r), a packet is seen as an event and can be recorded with arrival curves [177]. The packet stream (P_r) comprises of all of the packets that arrive at r during the execution of a particular program. Figure 9-7 illustrates comparison of two different packet streams, one normal and one compromised, over the time interval $[1, 17]$. P_r (blue) shows the normal stream of packet arrivals, and $\widetilde{P_r}$ (red) shows a compromised stream with an influx in packets over the same time. For some half-closed interval, $[t_a, t_b)$, the total number of packets passing through r is called the packet count ($N_{p_r}[t_a, t_b)$) which is defined in Equation 9-2. The parameters needed to calculate $N_{p_r}[t_a, t_b)$ are $N_{p_r}(t_a)$ and $N_{p_r}(t_b)$, which are the maximum number of packets before time t_a and time t_b, respectively.

$\forall t_a, t_b \in \mathbb{R}^+, t_a < t_b, n \in \mathbb{N}$:

$$N_{p_r}[t_a, t_b) = N_{p_r}(t_b) - N_{p_r}(t_a) \tag{9-2}$$

Then, we construct the upper PAC bound ($\lambda^u_{p_r}(\Delta)$) for every router from the previously collected packet arrival data. In order to construct an upper bound, the maximum number of

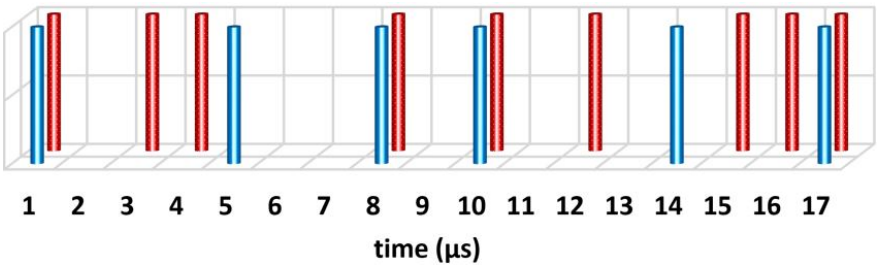


Figure 9-7. Two sample event traces where the blue trace shows packet arrivals at a router under normal operation (P_r) and the red trace shows packet arrivals in the presence of a DoS attack ($\widetilde{P_r}$).

arrivals is necessary for any time interval $\Delta(= t_b - t_a)$. Equation 9-3 defines how this is done by sliding a window of length Δ over P_r to calculate the maximum number of packets arrivals within that window.

$$\lambda^u_{p_r}(\Delta) = \max_{t \geq 0}\{N_{P_r}(t + \Delta) - N_{P_r}(t)\} \tag{9-3}$$

The process is repeated for several fixed Δ to construct the upper PAC bound. Once the upper PAC bound is constructed, then it can be used in detecting abnormal behaviors in real time by the "Leaky Bucket Algorithm". Figure 9-8 shows a PAC bound and two PACs corresponding to P_r and $\widetilde{P_r}$ from Figure 9-7. It illustrates how $\widetilde{P_r}$, the compromised stream, goes beyond the shaded region indicating there is a DoS attack happening.

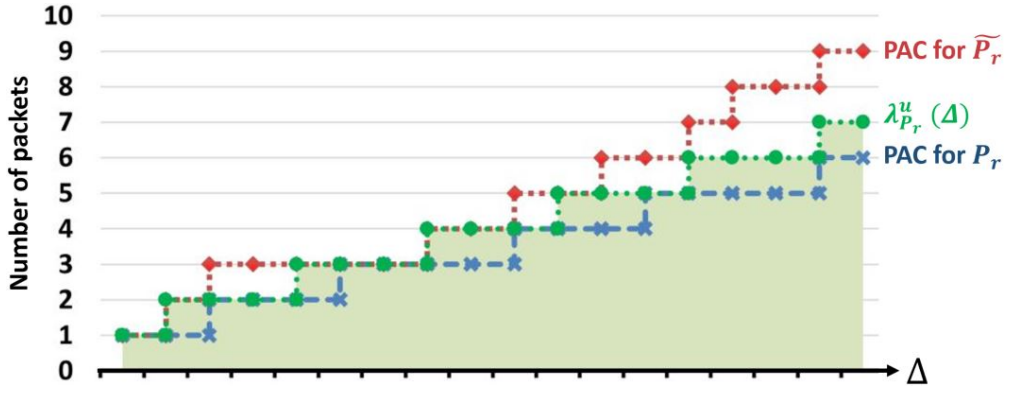
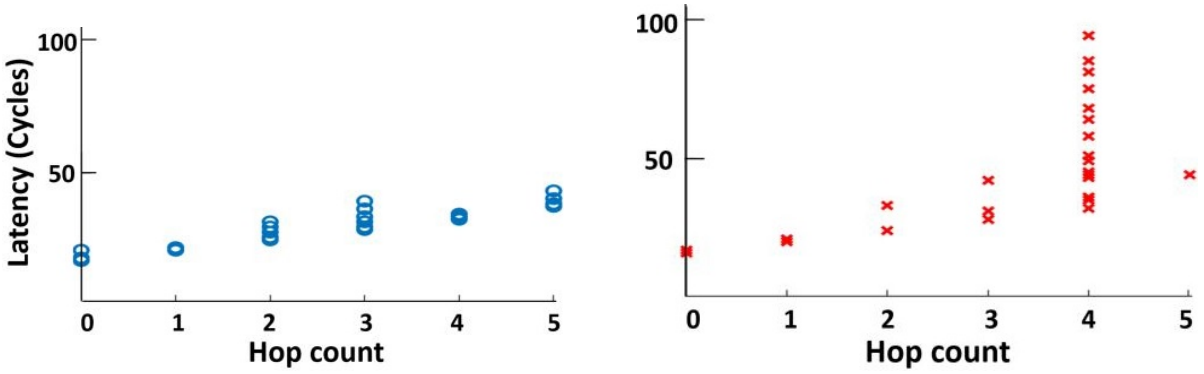


Figure 9-8. Graph showing upper bound of PACs ($\lambda^u_{p_r}(\Delta)$). The green line with round markers show the PAC bound whereas the normal operational area is shaded in green.

9.4.4.2 DoS attack localization using PAC bounds and DLCs

The localization method uses DLCs in addition to PAC bounds. While every router along the path constructs PACs, every destination IP constructs a DLC. Figure 9-9 shows two examples of DLCs with Figure 9-9(a) being normal operation and Figure 9-9(b) corresponding to an attack scenario. The DLCs capture the latency of a packet (y-axis) from source to destination D_i against the number of hops traversed by the packet from source to destination (x-axis). The distribution of latencies against each hop count follows the normal distribution, which is represented by its mean and variance. The mean and variance of the latency distribution of packets travelling k hops to reach D_i are denoted by $\mu_{i,k}$ and $\sigma_{i,k}$, respectively. The packet header holds the source and hop count information that the destination will extract for profiling. From this, the destination constructs a graph capturing the latency of packets from source to destination against the number of hops. Mean and variance for the distribution at each hop count is calculated after every packet has arrived.



A Normal operation

B Attack scenario

Figure 9-9. Two sample destination packet latency curves (DLC) constructed at an IP. Under normal operation, the variance of the distribution is small, whereas in a DoS attack scenario, it can be large.

Should a violation be flagged during the detection phase, the local IP attached to that router initiates the diagnosis. By referring to its DLC, it finds the packets that have suspicious (longer than usual) latencies using the parameterized $\mu_{i,k}$ and $\sigma_{i,k}$ values. The local IP then uses the source address of the delayed packets to get the congestion data from the other

routers in that path. We can only conclude that the source address of the delayed packets is a candidate MIP. If we conclude that the source address of the delayed packets is where the attack is originated can lead to many false positives. Therefore, the method relies on the victim pinpointing the attacker and other IPs removing the false positives. The behavior of each router during DoS attack localization is given in Algorithm 15.

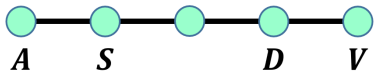


Figure 9-10. Illustrative example with local IP (D), attacker IP (A), victim IP (V), and the candidate MIP (S) as found by D.

We explain the behavior of the algorithm using Figure 9-10. The attacker IP, A, launches a DoS attack at V and two other IPs, S and D, are located along the same congested path. Routers of D and V both will flag a potential attack (lines 1-2) and check the DLC for candidate MIPs. Even though S and D are not the attacker or the victim, packets originating from S with destination D will be delayed since they are on the congested path. Therefore, the router of S will be flagged as a candidate MIP by D (lines 3-7). As a result, the router of S will receive a message from the router of D indicating that its local IP is the attacker, which will cause the flag to be set to 1 (lines 8-12). However, S will receive another message from V, since S is on the path from V to A, indicating that A is the potential attacker. This will cause the flag at S to be changed to 2 (lines 13-15). The router of A will only receive the message from V which will cause the flag to remain at 1. When the timeout occurs, flag at S is set to 2, and therefore, no action is taken. However, the router of A has a flag set to 1, and therefore, a broadcast is sent indicating that A is the attacker (lines 16-20).

The overview of both DoS attack detection and localization is shown in Figure 9-11. The attack detection phase occurs first and is shown in the left part of the figure. The localization of the MIP occurs after detection as shown in the right part of the figure. The complete methodology is described in [85]. Unlike the work done by Charles et al. [85], we use the

Algorithm 15 localizeDoS( ): event handlers for routers

upon event attacked == TRUE:
send a signal to local IP

upon receiving address of the candidate MIP S from local IP:
send a query to the router of S for its congestion status
if S is congested **then**
    sends a *diagnostic message* $< S, D >$ to all routers in the path from S to D indicating that S is the potential attacker
end if

upon receiving a diagnostic message $< S, D >$ **from port** p_i**:**
start TIMEOUT if all $flag == 0$
if S is local IP and $flag[p_i] == 0$ **then**
    $flag[p_i] = 1$ ▷ local IP is the MIP
end if
if S is not local IP **then**
    $flag[p_i] = 2$ ▷ local IP is not the MIP
end if

upon event TIMEOUT:
if $flag$ contains 1 **then**
    broadcasting that its local IP is the attacker
    RESET
end if

upon event RESET:
$flag[p_i] = 0$ for all ports p_i

service NoC to pass messages between IPs when localizing the MIP to reduce the localization time.

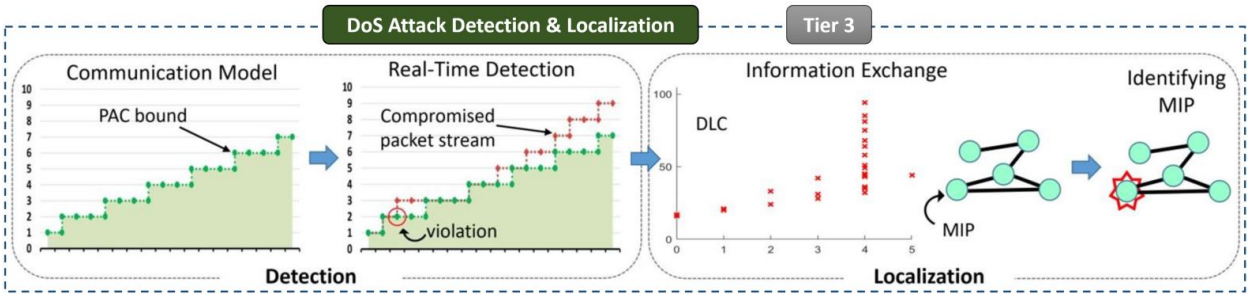


Figure 9-11. Overview of the DoS attack detection and localization framework.

9.5 Experiments

In this section, we first present the experimental setup used to evaluate our framework and then show the performance and energy data for different security tiers. Finally, we discuss the area overhead and security guarantees provided by the architecture.

9.5.1 Experimental Setup

Our experimental setup was built using the gem5 cycle-accurate full-system simulator [152]. We modeled an 8×8 Mesh NoC-based SoC with 64 IPs. GARNET2.0 detailed on-chip interconnection network model was used for the NoC after modifying the default garnet model to include our reconfigurable security architecture [147]. The delay for encryption/decryption and authentication was assumed to be 12 cycles [98]. GARNET2.0 uses the routing infrastructure provided by gem5's ruby memory system model. We set the number of pipeline stages in the router to be 3 and each link is assumed to consume 1 cycle to transfer a packet between neighboring routers. Each message from an IP goes through the security operations implemented at the NI and is then divided into flits (flow control units) before being injected into the NoC through its local router. The NoC then routes the packet depending on the routing protocol (XY routing in our experimental setup) and the NI at the destination performs decryption and tag validation before sending the message to the destination IP. The output statistics of the gem5 simulation were fed to the McPAT power modeling framework to obtain power consumption [151].

We tested the system using 4 real benchmarks (FFT, RADIX, FMM, LU) from the SPLASH-2 benchmark suite [142] and 6 synthetic traffic patterns (uniform random (URD), tornado (TRD), bit complement (BCT), bit reverse (BRS), bit rotation (BRT), transpose (TPS)). When running both real benchmarks and synthetic traffic patterns, each IP in the top (first) row of the Mesh NoC instantiated an instance of the task. Real benchmarks used 8 memory controllers that provide the interface to off-chip memory which were connected to the bottom eight IPs. As synthetic traffic patterns don't use memory controllers, the destination of injected packets were selected based on the traffic pattern. For example, uniform random

selected the destination from the 8 IPs at the bottom row with equal probability. Source and destination modeling was done this way to mimic the secure and non-secure zones. When simulating DoS attacks, a MIP that is randomly placed in the middle rows (non-secure zone) injected more packets into the NoC targeted at one of the destination IPs, which receive high traffic from legitimate requests. According to our architecture model, the IPs in the top row (secure zone) communicate with the IPs in the bottom row (secure zone) through the other 6 rows (non-secure zone) of IPs. Our approach will work the same for any other secure, non-secure zone selection and MIP placement. Table 9-3 shows the reconfigurable parameters selected in our experiments.

Table 9-3. Reconfigurable parameter values used in our experiments.

Security Primitive	Reconfigurable Parameters
Encryption (Tier 1)	AES Blockcipher, 128-bit key, 128-bit block, 96-bit IV
Authentication (Tier 2)	Galois Hash, 128-bit key, 128-bit input
DoS attack detection and localization (Tier 3)	Detection and localization both active, Detection always active without sleeping

These choices were motivated by the capabilities of the simulator as well as the lightweight nature of IoT and embedded devices.

9.5.2 Performance Results

To evaluate the execution time for each application, we simulated the setup with different security levels. Figure 9-12 and Table 9-4 show results for four levels of security when running real benchmarks.

- **No-Sec** - NoC without implementing any security.
- **Tier1** - Tier 1 security implemented. Encryption only.
- **Tier2** - Tier 2 security implemented. Encryption and authentication
- **Tier3** - Tier 3 security implemented. Encryption, authentication and DoS attack detection & localization.

Figure 9-12B shows NoC delay (end-to-end NoC traversal delay) of different security tiers. Compared to No-Sec, 40%, 57% and 58% more delay is observed on average across

all benchmarks in Tier1, Tier2 and Tier3, respectively. Execution time is compared in Figure 9-12C and it shows a similar trend. Tier1, Tier2 and Tier3 take 7%, 12.7% and 13.2% more time to execute each simulation, respectively. The impact of security features is less in total execution time since it includes instruction execution, memory operations, etc., in addition to NoC traversal delay. Difference between performance in Tier2 and Tier3 is very small (0.5% in execution time) since the DoS attack detection mechanism can run in parallel to normal router computations once separate hardware is implemented. Charles et al. reported that there is no performance overhead in their 5-stage router pipeline. However, since we have implemented a 3-stage router pipeline which makes the normal router computations to take place faster, DoS attack detection can take a bit longer depending on crossbar contention at that time, and therefore, we observe a slight delay.

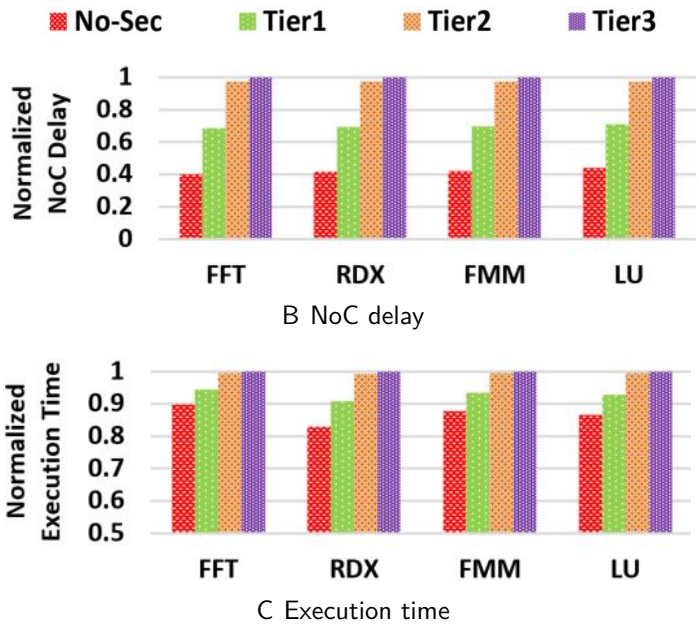


Figure 9-12. NoC delay and execution time comparison across different security levels using real benchmarks.

The same experiments were run on synthetic traffic patterns and results are shown in Figure 9-13. We can only capture NoC delay when running synthetic traffic patterns since running synthetic traffic patterns do not include instruction execution and memory operations. We observe 50%, 66% and 67% more NoC delay on average in Tier1, Tier2 and Tier3,

Table 9-4. Execution time comparison in terms of number of clock cycles across different security levels using real benchmarks.

	FFT	RDX	FMM	LU
No-Sec	3.444E+08	2.419E+10	8.807E+09	3.684E+09
Tier1	3.638E+08	2.669E+10	9.422E+09	3.968E+09
Tier2	3.833E+08	2.918E+10	1.004E+10	4.251E+09
Tier3	3.849E+08	2.939E+10	1.009E+10	4.275E+09

respectively, when compared with No-Sec. Both Figure 9-12 and Figure 9-13 show us that added security comes at the expense of performance. Therefore, the security level has to be reconfigured depending on the use-case scenario.

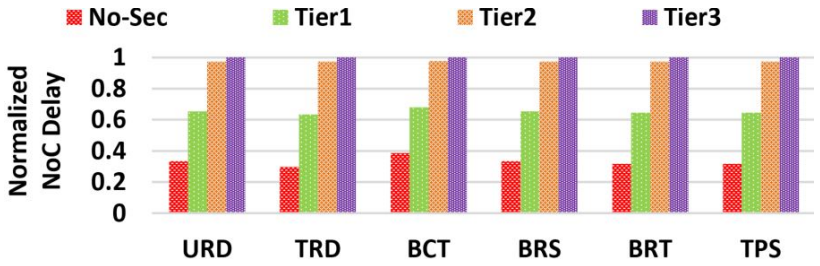


Figure 9-13. NoC delay comparison across different levels of security when running synthetic traffic patterns.

9.5.3 Overhead Analysis

This section provides details on area and power overhead of our proposed framework. It also discusses overhead associated with various components including service NoC, RSE as well as security tier changes.

9.5.3.1 Area overhead

Additional hardware is required to implement our reconfigurable security architecture. To evaluate this area overhead in comparison with the NoC that does not implement any security (No-Sec), we implemented the security tiers using Verilog. We modified the RTL of an open-source NoC architecture [185] and conducted our experiments using Synopsys Design Compiler with 90nm Synopsis library (saed90nm). Results are shown in Table 9-5. Area overhead was calculated for each security tier. For example, area overhead for Tier2 includes overhead introduced by encryption as well as authentication hardware. Since our proposed

framework requires all the features to be integrated so that the RSE can select which tier to activate, the total area overhead is the overhead to implement Tier3 security, which is 6%. If a certain SoC designer decides to only integrate features in Tier1 (encryption), overhead would be 4.2%.

If Tier3 is implemented, in addition to the additional hardware required at routers and NIs, each IP stores and processes the DLCs. The result of $\mu_{i,k} + 1.96\sigma_{i,k}$ is calculated and stored for each hop count in the DLC as a 4-byte integer. This aggregates to a total memory space of $1 \times m \times 4$ parameters to store the DLC, where m is the maximum hop count between two IPs in the NoC. It is safe to assume that this additional memory space is negligible since the IPs typically have much more bandwidth than any other NoC component.

Table 9-5. Area occupied by security tiers.

	Tier 1	Tier 2	Tier 3	Tier 1 (Overhead)	Tier 2 (Overhead)	Tier 3 (Overhead)
Area	609696 μm^2	618473 μm^2	620228 μm^2	4.2%	5.7%	6%

9.5.3.2 Power overhead

The power overhead is introduced by the additional computations required to implement the reconfigurable security architecture. Compared to No-Sec, each packet injected into the network will have to go through encryption when Tier1 is enabled. At the destination, the inverse process - decryption takes place. These processes consume extra power. Tier2 consumes extra power for the tag computation and validation part and Tier3, for constructing DLCs at each IP and monitoring DoS attacks at each router. The gem5 output statistics were fed into the McPAT power modeling framework to obtain power consumption. The NoC power model in McPAT was modified according to the work done by Ogras et al. [135]. Power consumption when running the four real benchmarks (FFT, RADIX, FMM, LU) were recorded and average power consumption is compared in Table 9-6 together with power overhead introduced by each security tier. Tier1 has a power overhead of 3.2%, which consists of overhead for encryption. Note that each tier includes capabilities of the tier below. In

other words, 4.8% power overhead in Tier2 includes both encryption (3.2% in Tier 1) and authentication (1.6%) power overhead. Similarly, Tier3 consumes 7.9% power overhead, which includes the 4.8% overhead for Tier2 and an additional 3.1% for DoS detection and localization. The results are consistent with the previous studies on lightweight NoC encryption done by Sepúlveda et al. [90].

Table 9-6. Power consumption of our approach.

	Tier 1	Tier 2	Tier 3	Tier 1 (Overhead)	Tier 2 (Overhead)	Tier 3 (Overhead)
Power	5304 mW	5387 mW	5546 mw	3.2%	4.8%	7.9%

9.5.3.3 Overhead of service NoC

The Service NoC proposed in our architecture is responsible for transferring packets intended for the following purposes;

- DoS attack localization
- Communication between SAGs and RSE
- Key distribution for encryption and authentication

The proposal to use a separate NoC instead of using one NoC for all purposes was motivated by state-of-the-art commercial SoCs that implement multiple physical NoCs to carry different types of packets [4, 17]. The Intel Knights Landing (KNL) architecture features four parallel NoCs [4] and has been widely deployed in the Intel Xeon processor family. The Tilera TILE64 architecture comprises of five parallel 2D Mesh NoCs, each used for a different purposes such as communication with main memory, communication with I/O devices, and user-level scalar operand and stream communication between tiles [17].

The trade-off here is performance versus area. When many different types of packets are used in the NoC, the packet must contain data to distinguish between those types. Existing buffer space has to be shared between packet types. Both these concerns add performance overhead and when scaling upto 64 IPs, the overhead becomes significant. On the other hand, contrary to intuition, additional wiring between nodes incur minimal overhead as long

as the wires stay on-chip due to the advancements in fabrication processes. Furthermore, the more expensive and scarce commodity is the on-chip buffer area compared to wiring bandwidth. If virtual channels are used for different types of packets [113] and buffer space is shared, the increased buffer spaces and logic complexity will equal to that of another physical network. Yoon et al.'s work provides a comprehensive analysis about the trade-offs between having virtual channels and several physical NoCs [172]. Using their analysis that fits the NoC parameters we have chosen, the area and power overhead of having two physical NoCs compared to one NoC are 6% and 7%, respectively.

9.5.3.4 Overhead of RSE implementation

The RSE is a dedicated IP on the SoC that decides the security tier to be used based on the security policies. The policy engine can be implemented as a finite state machine (FSM) where the security tiers are the states and state transitions happen depending on the policies. In our work, we have discussed a specific implementation where RSE decides which security tier and parameters to activate based on battery level and NoC congestion information. The implementation of such a finite state machine incurs negligible area and power overhead [202].

9.5.3.5 Overhead of changing security tiers

It is worthwhile discussing what happens to the in-flight packets on the NoC during a security tier change. When changing from Tier1 to Tier2, the transition forces an authentication tag to be included in the NoC packets. However, the in-flight packets when the transition happens does not contain an authentication tag since they were injected when the architecture was in Tier1. Therefore, any packet that does not contain an authentication tag will be dropped. Since we do not expect security to be reconfigured frequently, the performance overhead due to the dropped packets is negligible. In fact, security reconfiguration is expected to be less frequent compared to traditional reconfiguration techniques such as dynamic voltage scaling (DVS) or dynamic cache reconfiguration (DCR). In DCR or DVS, the reconfiguration frequency depends on the length of a phase in a task, which is in the order of milliseconds or seconds. We envision that security reconfiguration frequency will be in the

order of minutes of even hours. To quantify the performance overhead for dropping packets, we profiled the maximum number of in-flight packets at any given time when Tier 1 is active. Table 9-7 shows the results as a comparison of total number of packets injected when running each benchmark.

Table 9-7. Maximum number of packets in flight at any given time compared to total number of packets injected when running each real benchmark.

	FFT	RDX	FMM	LU
Total number of packets injected	809,632	103,987,824	25,629,248	11,820,880
Maximum number of packets in Flight at any given time	532	569	585	630
Max # packets in flight as a percentage of total # of packets	0.0657%	0.0005%	0.0023%	0.0053%

Dropping packets will not affect the accuracy of operation, since the IPs which injected the dropped packets will re-transmit the requests after not receiving a response. Such re-transmission mechanisms are already in place for NoC error correction protocols [203]. Note that packet dropping is not required when transitioning from Tier 2 to Tier 3 (or vice versa) due to the nature of the DoS attack detection mechanism. We have added this discussion in Section 9.5.3.5.

9.5.4 Security Analysis

In this section, we discuss the security guarantees of different security tiers.

Tier1: implements encryption only. Therefore, the secrecy of packets is ensured while the integrity of packets is not. An eavesdropper on the NoC will be unable to read the critical data in a packet unless it manages to break the cipher. The security of the cipher depends on the security of the operation mode, counter mode in this case, as well as the block cipher. Each block in counter mode is treated independently while encrypting. In such a setup, using the same $IV \parallel \{q\}_d$ string with the same key K can cause the "two time pad" situation. In our method, using a nonce as the IV for each encryption addresses this. Further security can be ensured by setting the string to $IV \parallel seq_j \parallel q$ where q corresponds to the block cipher ID and seq_j represents the sequence number of the j^{th} packet. It gives per message and per block

variability and ensures that the value is a nonce. The use of $IV \parallel seq_j \parallel q$ string allows reusing the IV and it can be reset after a certain number of encryptions. GCM uses AES as its block cipher. AES has been shown to be resistant against all known cryptographic attacks and is yet to be broken [204].

Tier2: adds another layer of security on top of encryption by enabling authentication. This addresses the issue of data integrity. The authentication tag validation relies on the fact that unless the hash key is known, no other key and input string combination should produce the same hash digest. If this condition fails, an adversary will be able to alter the packet content, re-generate a tag for that string and replace the existing tag with it. Then the corrupted packet will be validated as a legitimate packet. To ensure this doesn't happen, the chosen hash function has to be collision resistant. Our choice of hash function - Galois hash adheres to this criteria and is also pre-image and secondary pre-image resistant [198].

Tier3: contributes the last layer of security of our framework - DoS attack detection & localization. To evaluate the efficiency of the approach, we ran simulations in the presence of one MIP placed at random in the middle rows (non-secure zone). The MIP injected more packets into the NoC targeted at one of the destination IPs. The packet stream periods and attack periods were selected at random. Packet steam periods were assigned a value between 2 and 6 μs at random and attack periods were assigned a random value between 10% and 80% of the packet stream period. Experiments were conducted using the six synthetic traffic patterns and random placements of MIP launching the DoS attack. Out of the collected traces, 10 of them were selected such that the test cases include all synthetic patterns and applicable MIP placements. Figure 9-14 shows the detection time for the 10 test cases. The results show that the detection time depends on the attack period and is approximately twice the attack period. This confirms that DoS attack detection can be done in real-time. The final step of Tier3, DoS attack localization can be done in real-time as well, as shown in Figure 9-15. The efficiency of DoS attack localization was evaluated by measuring the time

between detecting the attack and localizing the malicious IP. Figure 9-15 shows the results of our experiments using the same ten test cases running synthetic traffic patterns.

The detection time and localization time both depend on characteristics of the NoC as well as the position of victim/malicious IPs in the NoC. The proposed method detects a DoS attack when the number of packet arrivals within a given time window exceeds the upper bound. The time taken for this to happen depends on the the constructed upper bound, packet arrival trends at routers along the path of the DoS attack, attack period and packet stream period during normal operation. If the upper bound is tight during normal operation for a particular time window, it only takes few additional packets to violate it. Therefore, some test cases can exceed the upper bound quickly leading to detection times being very close to the packet stream period. Some can take longer to exceed the interval since within that time window, the upper bound was not violated.

The localization time depends heavily on the time it takes for the diagnostic packets to traverse from the IPs connected to the routers that flagged the attack to the potentially malicious IP. The localization time varies for each topology and victim/malicious IP placement. For example, if we used a Point2Point topology, localization needs diagnostic message to travel only one hop, whereas a Mesh may require multiple hops. Therefore, localization is faster in Point2Point compared to a Mesh. In general, the localization time is less compared to detection time because the localization process completes once the small number of diagnostic packets reach all the potentially malicious IPs, whereas detection requires many packets before violating a PAC bound during runtime.

The results are consistent with the work done in [85]. Charles et al. has shown that the framework is capable of detecting and localizing DoS attacks across different topologies and deterministic routing protocols [85]. Therefore, it is a perfect fit for real-time IoT applications.

9.6 Summary

In this chapter, I presented a reconfigurable security architecture which allowed enabling/disabling of security levels (tiers) depending on the use case scenario. Security cannot

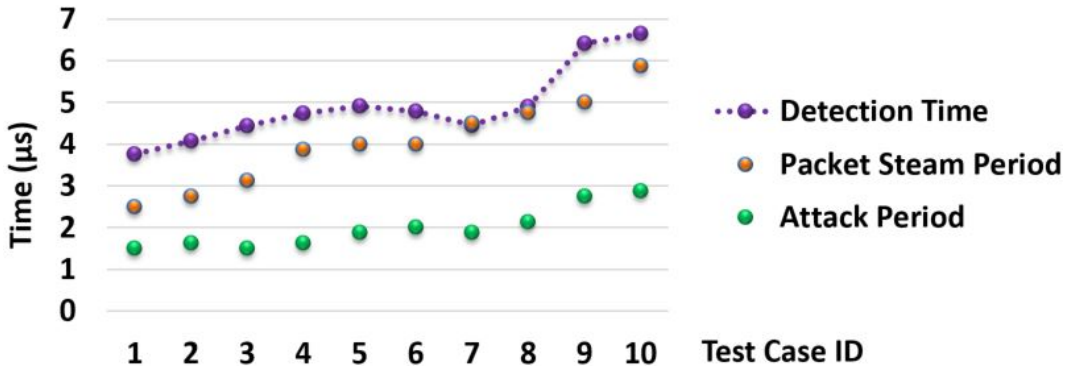


Figure 9-14. DoS attack detection time for 8×8 Mesh topology in the presence of one MIP.

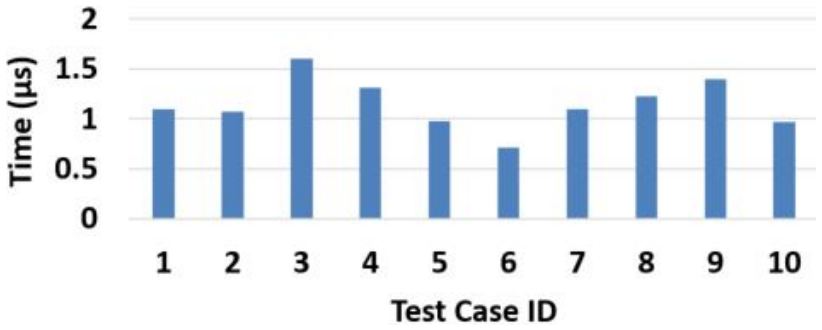


Figure 9-15. DoS attack localization time for 8×8 Mesh topology in the presence of one MIP.

be considered alone in resource constrained IoT devices. The interoperability constraints - performance, energy efficiency and area should be taken into account when deciding the level of security required. I introduced a tier-based security architecture and proposed an efficient reconfiguration mechanism that allows monitoring system characteristics and decides which security mechanism(s) to activate based on security policies. The proposed tier-based security mechanisms comprises of encryption, authentication and DoS attack detection & localization. Experimental results discussed how different tiers can affect the interoperability constraints as well the security guarantees. My reconfigurable security architecture is lightweight and provides real-time security guarantees. Therefore, it is ideal for resource-constrained IoT devices that has dynamic requirements and long application life.

CHAPTER 10
DIGITAL WATERMARKING FOR DETECTING MALICIOUS IPS

Eavesdropping attacks have become a major concern with a recent occurrence of a hardware security breach due to third-party vendors aiming at industrial espionage raising concerns across top US authorities [53]. The attack was facilitated by a hardware Trojan that acted as a covert backdoor and spied on computer servers used by more than 30 companies in USA, including Amazon and Apple. In this chapter, I propose a digital watermarking-based malicious intellectual property (IP) detection mechanism for eavesdropping attacks.

Specifically, I consider the following attack scenario. A hardware Trojan integrated in the NoC IP launches an attack to eavesdrop on the NoC packets. The goal is to exfiltrate information while remaining hidden, and thus the Trojan will not perform any action that would reveal its presence, such as corrupting packets to cause SoC malfunction (data integrity attacks) or degrade performance causing denial-of-service (DoS) attacks. Previous work has explored the most effective way of launching an eavesdropping attack in NoC, considering attack effectiveness and difficulty to detect the Trojan. It identified Trojan(s) inserted in NoC component(s) colluding with another malicious IP(s) as the strongest attack model. An illustrative example of this scenario is shown in Figure 10-1, where a hardware Trojan-infected router and an accomplice application launch an eavesdropping attack where the infected router copies packets passing through it and sends them to the accomplice application running on another malicious IP. This hardware-software collusion attack is similar to the Illinois Malicious Processor (IMP) [50]. This setting and related threat models have been the focus of [46] as well as several prior studies [49, 88–91, 95, 168].

NoC security research has proposed authenticated encryption (AE) as a solution to eavesdropping attacks [88, 90, 91]. With AE, packets are encrypted to ensure confidentiality and an authentication tag is appended to each packet to ensure integrity (and detect re-routed packets). However, the use of AE as the defense to eavesdropping attacks is sub-optimal for two reasons. First, it incurs significant performance degradation on resource-constrained

devices (as I show experimentally in Section 10.2). Second, authentication tags may be unnecessarily complex if used only for the purpose of detecting eavesdropping attackers who seek to remain undetected as long as possible — and thus are unlikely to interfere with data integrity.

In this chapter, I address a fundamental question: *is it possible to replace authenticated encryption with a lightweight defense while maintaining security against eavesdropping attacks?* Specifically, I propose to replace the costly computation of authentication tags with a lightweight eavesdropping attack detection mechanism based on digital watermarking. The attack detection capabilities achieved by digital watermarking is coupled with encryption to ensure data confidentiality.

The remainder of the chapter is organized as follows. Section 10.1 describes the threat model in detail. Section 10.2 motivates the need for my work. Section 10.3 introduces my watermarking-based attack detection method. Section 10.4 provides theoretical guarantees on performance and security of my approach followed by experimental results in Section 10.5. Section 10.6 discusses additional security considerations. Finally, Section 10.7 summarizes the chapter.

10.1 Background and Threat Model

In this section, we present a background on digital watermarking and discuss the threat model in detail.

10.1.1 Digital Watermarking

The process of hiding information related to digital data in the data itself is called digital watermarking. It has been widely used in domains such as broadcast monitoring, copyright identification, transaction tracking, and copy control. For example, in the movie industry, a unique watermark can be embedded in every movie. If the movie later gets published on the internet illegally, the embedded watermark can be used to identify the person who leaked it. In network flow watermarking, watermarks are embedded into the packet flow using packet content [205], timing information [206] or packet size [207]. This can be used for

tracing botmasters in a botnet [208], tracing other network-based attacks [209] and service dependency detection [210].

10.1.2 Threat Model

The global trend of distributed design, validation and fabrication has raised concerns about security vulnerabilities. Malicious implants, such as hardware Trojans, can be inserted into the RTL or into the netlist of an IP core with the intention of launching attacks without being detected at the post-silicon verification stage or during runtime [7]. Insertion of Trojans can happen in many places of the long, distributed supply chain such as by an untrusted CAD tool or designer or at the foundry via reverse engineering [43]. As evidence of the globally distributed supply chain of NoC IPs, iSuppli, an independent market research firm, reports that the FlexNoC on-chip interconnection architecture [40] is used by four out of the top five Chinese fabless semiconductor OEM (original equipment manufacturer) companies [47]. In fact, Arteris, the company that developed FlexNoC, achieved a sales growth of 1002% over a three-year time period through IP licensing [48]. Therefore, there is ample opportunity for attackers to integrate hardware Trojans in the NoC IP and compromise the SoC. NoC IPs are ideal candidates to insert hardware Trojans due to several reasons: i) the complexity of NoC IPs makes it extremely difficult to detect hardware Trojans during functional verification as well as runtime [49], ii) extracting data from NoC packets allows attackers to obtain confidential information without relying on memory access or hacking into individual IPs, and iii) the distributed nature of NoC components across the SoC makes it easier to launch attacks.

We focus on eavesdropping attacks, also known as snooping attacks, which pose a serious threat to applications running on many-core SoCs. IPs that are integrated on the same SoC use the NoC IP when communicating through message passing as well as through shared memory. For example, the Intel Knights Landing architecture prompts memory requests/responses from cores to traverse the NoC for shared cache look-ups and for off-chip memory accesses [4]. Therefore, eavesdropping on data transferred through the NoC allows adversaries to extract confidential information.

Adversarial model. In this chapter, we consider an adversary consisting of a hardware Trojan-infected router and a colluding malicious application running on an IP. The goal of the adversary is to exfiltrate confidential information by observing NoC traffic *without being detected*. Remaining hidden is key for the adversary to exfiltrate as much information as possible. Because the adversary must remain hidden, we assume that the adversary does not interfere with the normal operation of the NoC. For example, this means that the adversary does not modify the content of packets (attack on integrity) or cause large delays in processing of packets (denial-of-service) as either would likely lead to detection.

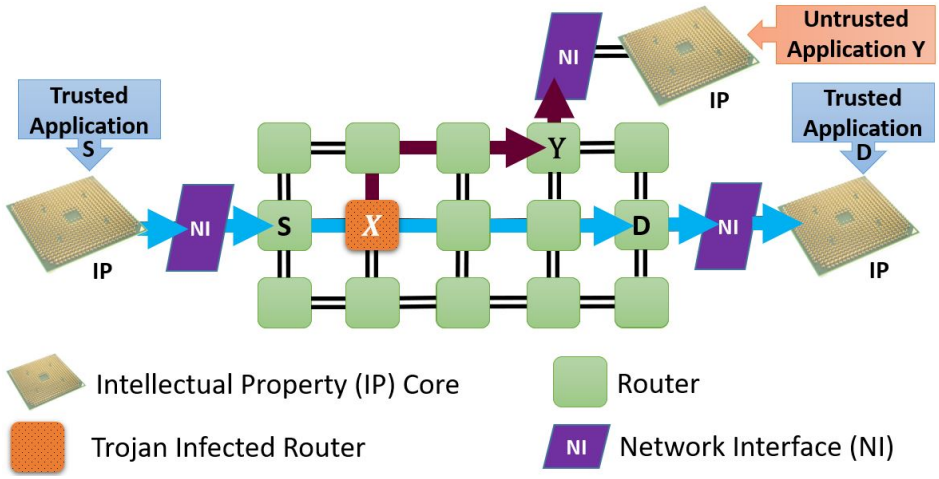


Figure 10-1. Illustration of an eavesdropping attack through colluding hardware and software.

Attack scenario. Eavesdropping attacks by malicious NoC IPs rely on the hardware Trojan creating duplicate packets with modified headers (specifically, destination address in the header) and sending them into the NoC for an accomplice application to receive them [46, 49]. Figure 10-1 shows an illustrative example. We consider a commonly used 2D Mesh NoC topology where IPs are connected to the NoC, more specifically to the router, via a network interface (NI). When the NI receives a message from the local IP, the message is

packetized and injected into the network.[1] Packets injected into the NoC are routed using the hop-by-hop, turn-based XY routing algorithm and received by the destination router. The NI then combines the packets to form the message which is passed to the intended destination IP. In our example (Figure 10-1), two trusted applications running in nodes S and D are communicating with each other, and an eavesdropping attack is launched to steal confidential information. The attack is carried out by two main components: i) a Trojan-infected router, and ii) an IP running a malicious application. The malicious router (X) copies packets passing through it and sends them to the IP running the malicious program at node Y, which reads the confidential information. To facilitate this attack, several steps should be carried out by the attacker. First, the hardware Trojan is inserted by the third-party NoC IP provider during design time. The Trojan is designed such that it can act upon commands sent by the malicious application. Once the SoC is deployed, the malicious application sends commands at a desired time to launch the attack. The Trojan then starts copying and sending packets to the malicious application. The malicious application can also send commands to pause the attack to avoid being detected.

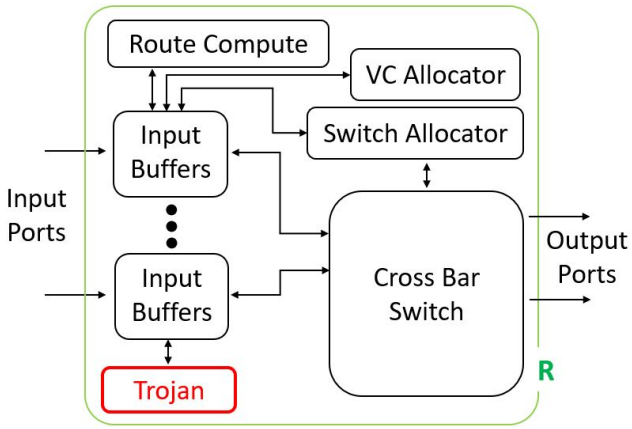


Figure 10-2. Router infected with a hardware Trojan.

[1] Most NoCs facilitate flits, which is a further breakdown of a packet used for flow control purposes. We stick to the level of packets for the ease of explanation as our method remains the same at the flit level as well.

Figure 10-2 shows a block diagram of a router design infected with the Trojan that launches the attack described in our threat model [46]. The Trojan copies packets arriving at the input buffer, changes the header information so that the new destination of the packet is where the malicious application is (node Y according to our illustrative example) and inject the new packet back to the input buffers so that it gets routed through the NoC to reach Y. The Trojan does not tamper with any other part of the packet, except for the header to re-route the packet, due to two reasons: i) the goal is to extract information, so corrupting data defeats the purpose, and ii) corrupting data increases chances of the Trojan getting detected. Since the original packet is not tampered with and is routed to the intended destination D, the normal operation of the SoC is preserved. The Trojan also has a very small area and power footprint. Ancajas et al. [46] used a similar threat model in their work and reported 4.62% and 0.28% area and power overheads, respectively, when compared with the router design without the Trojan. The performance overhead when copying and routing packets to the malicious application is less than 1% [46]. Therefore, the likelihood of the Trojan being detected is very small unless additional security mechanisms (such as the one proposed in this chapter) are implemented.

10.2 Motivation

AE is a widely accepted countermeasure against eavesdropping attacks. Details of how AE can be implemented at the NoC level is given in Section 9.4.2. Encryption provides packet confidentiality and authentication is capable of detecting re-routed packets. Since the header is modified by the hardware Trojan in order to re-route the packet to the malicious application, the authentication tag validation fails and the attack is detected. To analyze the performance overhead introduced by an AE scheme, we ran FFT, RADIX (RDX), FMM and LU benchmarks from the SPLASH-2 benchmark suite [142] on an 8×8 Mesh NoC-based SoC with 64 IPs using the gem5 simulator [18] considering two scenarios:

- **Default-NoC**: Bare NoC that does not implement encryption or authentication.

- **AE-NoC**: NoC that uses an authenticated encryption scheme with a setup similar to Figure 9-6.

More details about the experimental setup is given in Section 10.5.1. Results are shown in Figure 10-3. A 12-cycle delay was assumed for encryption/decryption and authentication tag calculation when simulating AE-NoC according to the evaluations in [97]. The values are normalized to the scenario that consumes the most time. AE-NoC shows 59% (57% on average) increase in NoC delay (average NoC traversal delay for all packets) and 17% (13% on average) increase in execution time compared to the Default-NoC. The overhead for security has a relatively lower impact on execution time compared to the NoC delay since the execution time also includes the time for executing instructions and memory operations (in addition to NoC delay). NoC delay in Default-NoC case is caused by delays at routers, links and the NI. In AE-NoC, in addition to those delays, encryption/decryption delays and authentication tag calculation/validation delays are added to each packet. Additional delays are due to complex encryption/decryption operations and hash calculations for authentication.

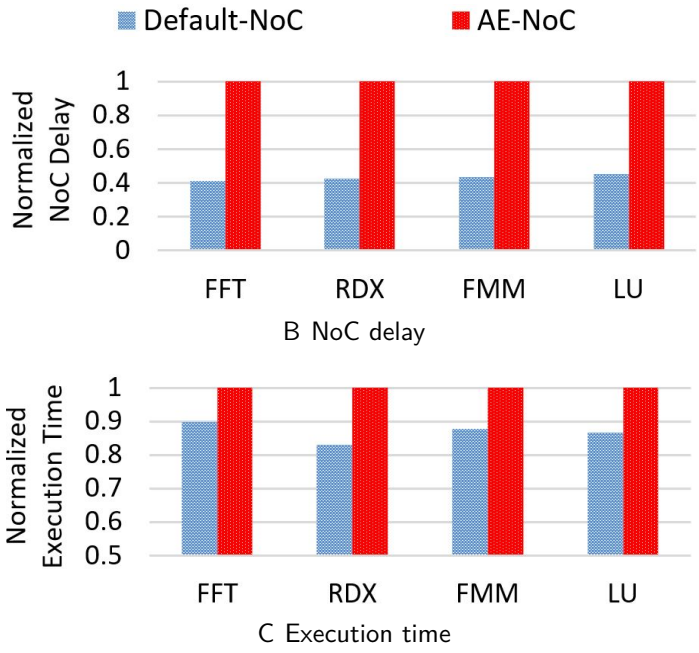


Figure 10-3. NoC delay and execution time comparison across different levels of security for four SPLASH-2 benchmarks.

When security is considered, Default-NoC leaves the data totally vulnerable to attacks, whereas AE-NoC ensures confidentiality and data integrity. For systems with real-time requirements, an execution time increase of 17% to accommodate a security mechanism is unacceptable. Furthermore, validating the authentication tag for each packet contributes to the SoC power consumption. Since the Trojan is rarely activated and only the packet header is modified (packet data is not corrupted) to avoid detection, authenticating each packet becomes inefficient in terms of both performance and power consumption [88]. Clearly, authenticating to detect re-routed packets introduce unnecessary overhead. It would be ideal if the security provided by AE-NoC could be achieved while maintaining the performance of Default-NoC. However, in resource-constrained environments, there is always a trade-off between security and performance.

In this chapter, we propose a novel digital watermarking-based security mechanism that incurs minimal overhead while providing high security. Our approach replaces authentication by watermarking. Encryption is used to ensure data confidentiality. Our method achieves a better trade-off than: (1) no authentication that is vulnerable to credible Trojan attacks, and (2) authenticated encryption, which incurs performance degradation prohibiting their use in applications with real-time constraints.

10.3 NoC Packet Watermarking

In this section, we first present a few key definitions and concepts used in our proposed watermarking construction. We then describe our lightweight eavesdropping attack detection mechanism based on digital watermarking.

10.3.1 Definitions

In this section, we introduce two important definitions that would be used in the rest of the chapter.

10.3.1.1 Hoeffding's inequality

Let $\{X_1, ..., X_n\}$ be a sequence of independent and bounded random variables with $X_i \in [a, b]$ for all i, where $-\infty < a \leq b < \infty$. Then;

$$\Pr\left[\left|\frac{1}{n}\sum_{i=1}^{n}(X_i - \mathbb{E}[X_i])\right| \geq t\right] \leq e^{\left(-\frac{2nt^2}{(b-a)^2}\right)}$$

for all $t \geq 0$ [211]. By Hoeffding's Lemma, which says if $X_i \in [a, b]$ then $\mathbb{E}\left[e^{\lambda X}\right] \leq e^{\lambda^2(b-a)^2/8}$ for any $\lambda \geq 0$, a random variable bounded in $[a, b]$ is sub-Gaussian with variance proxy $\sigma^2 = \frac{(b-a)^2}{4}$. Therefore;

$$\Pr\left[\left|\frac{1}{n}\sum_{i=1}^{n}(X_i - \mathbb{E}[X_i])\right| \geq t\right] \leq e^{\left(-\frac{nt^2}{2\sigma^2}\right)} \tag{10-1}$$

10.3.1.2 Bounds for binary codes

Let $\mathcal{C}$ be a binary code of length w, size M (i.e., having M codewords) and minimum Hamming distance δ between any two codewords denoted by (w, M, d). The distance distribution of $\mathcal{C}$ can be calculated as;

$$B_i = \frac{1}{M}\sum_{c \in \mathcal{C}} |c' \in \mathcal{C} : \mathcal{D}(c, c') = i|, 0 \leq i \leq n$$

It is clear that $B_0 = 1$ and $B_i = 0$ for $0 < i < d$ [212].

Let $A(w, d)$ represent the maximum number of codewords M in any binary code of length w and minimum Hamming distance d between codewords. Finding optimum $A(w, d)$ for a given w and d is an NP-Hard problem [213]. However, exact solutions are known for few combinations of values and in the general case, upper and lower bounds of the maximum number of codewords are known [214, 215].

10.3.2 Overview

We call the flow of packets sent from one IP (source) to another IP (destination), a "packet stream". Our detection mechanism relies on the following assumptions about the architecture and threat model.

- The Trojan does not tamper with the legitimate packet content as this may reveal its presence (Section 10.1.2). The Trojan only modifies the header of duplicated packets to change the destination (data fields of the duplicated packets are not tampered with) and it allows the legitimate packets to pass as usual.
- Packets are not dropped by intermediate routers and the order of packets in a packet stream is kept constant. This is reasonable as deadlock and livelock free XY routing is used together with FIFO buffers [49].
- When the attacker injects copied packets into the NoC, all the packets can get delayed due to congestion. While this delay is random, the maximum delay is bounded. We explore this assumption in detail in Section 10.4.2.

Our proposed approach is to embed a unique watermark into every packet stream. Figure 10-4 shows an overview. We propose to include the watermark encoder and decoder at the NI of each node. It is reasonable to assume that the NI can be trusted since it acts as the interface between all the IPs in the SoC and the NoC IP, and is typically designed in-house [95, 97]. The NI at source S encodes the watermark and the NI at destination D decodes it to identify that the packet stream is valid, or in other words, the packets in the packet stream are intended to be received by D. This process is followed by each source/destination pair in the NoC. In case of an attack, the watermark decoded by the NI of the receiving node (node Y according to our illustrative example), will be invalid and a potential attack is flagged. To ensure this behavior, the watermarking mechanism must have the following characteristics:

1. The watermark is unique to each packet stream.
2. There is a shared secret between S and D, which is "hard" for any other node to guess or deduce.

In addition to watermarking, we rely on encryption/decryption modules implemented at the NIs. The watermark is embedded in the encrypted packets and is decoded before the decryption process. Encrypting packets is required to provide data confidentiality during packet transfers and due to the nature of our watermarking scheme that allows the malicious application to receive some packets before detecting the attack. Proposing an encryption mechanism is beyond the scope of this chapter and several previous work have already

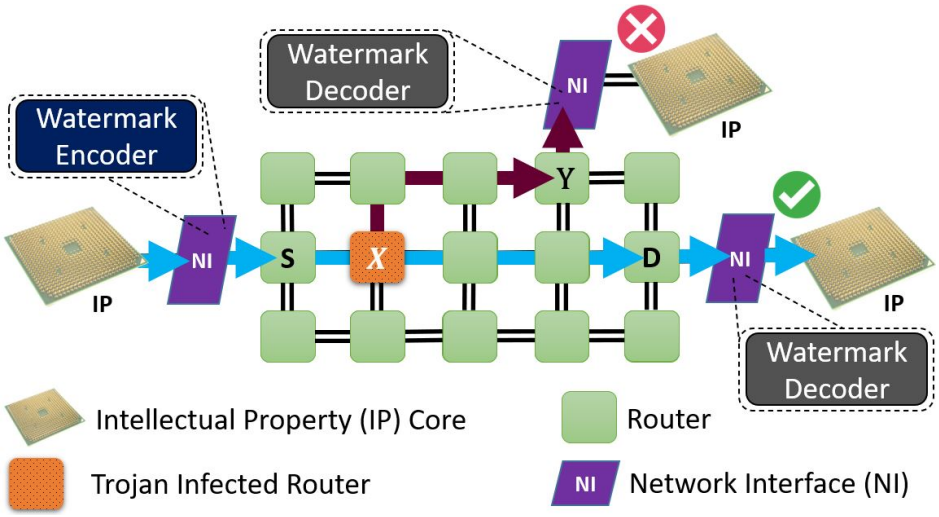


Figure 10-4. Overview of the watermarking scheme where the watermark encoder and decoder are implemented at the NI.

proposed NoC-based SoC architectures with encryption/decryption modules implemented at the NI [95, 97, 98]. Our proposed watermarking scheme can be implemented on top of those solutions. The performance improvement is achieved by replacing the authentication scheme with our lightweight digital watermarking scheme. The following sections describe our approach in detail. First, we outline the concept behind probabilistic NoC packet watermarking (Section 10.3.3), and then discuss the operation of the watermark encoder and decoder in detail (Section 10.3.4). Finally, we outline an effective method for managing secrets shared between nodes (Section 10.3.5).

10.3.3 Probabilistic Watermarking Concept

The watermark ω_{SD} is embedded by the NI of S before the packets are injected into the NoC. We use a timing-based watermark (as opposed to size or content-based) for three reasons; (i) timing alterations are harder to detect by an attacker, (ii) it allows a lightweight implementation as it is easy to manipulate, and (iii) it does not alter the packet content allowing encryption schemes to be implemented together with watermarking [216]. The watermark is embedded by slightly delaying certain packets in the stream. If ω_{SD} is unique, it should be correctly decoded at the NI of destination D with high probability. In contrast, the probability of decoding ω_{SD} as valid at any other NI should be very low.

Given n packets of a packet stream P_{SD} such that;

$$P_{SD} = \{p_{SD,1}, p_{SD,1}, ..., p_{SD,i}, ..., p_{SD,n}\}$$

the inter-packet delay (IPD) between any two packets can be calculated as $\tau_{SD,i,i+1} = t_{SD,i+1} - t_{SD,i}$ where $t_{SD,i}$ is the timestamp of the packet $p_{SD,i}$. Without loss of generality, for the ease of illustration, we will remove "SD" from the notation and denote the packet stream P_{SD} as P and IPD $\tau_{SD,i,i+1}$ as τ_i.

The encoder selects $2m$ packets $\{p_{r_1}, p_{r_2}, ..., p_{r_{2m}}\}$ out of the n packets of packet stream P. The selected packets are paired with another $2m$ packets (outside of the initially selected $2m$ packets) to create $2m$ pairs such that each pair is constructed as $\{p_{r_z}, p_{r_z+x}\}$ where $x \geq 1$ and $z = 1, ..., 2m$. Therefore, it is assumed that the packet stream has at least $4m$ packets. The IPD between each pair of packets can be calculated as;

$$\tau_{r_z} = t_{r_z+x} - t_{r_z} \tag{10-2}$$

Given that the $2m$ packets are selected independently and randomly, we model the IPDs as independently and identically distributed (IID) random variables with a common distribution. The IPD values are then divided into 2 groups. Since we had $2m$ pairs of packets, each group will have m IPD values. Let the IPD values of the two groups be denoted by τ_k^1 and τ_k^2 ($k = 1, ..., m$), respectively. It follows that both τ_k^1 and τ_k^2 are IID. Therefore, the expected values μ (and the variances) of the two distributions are equal. Let Δ be the average difference between the two IPD distributions:

$$\Delta = \frac{1}{m} \cdot \sum_{k=1}^{m} \frac{\tau_k^1 - \tau_k^2}{2} \tag{10-3}$$

Then, we can calculate the expected value and variance of Δ:

$$\mathbb{E}[\Delta] = \mathbb{E}\left[\tau_k^1\right] - \mathbb{E}\left[\tau_k^2\right] = 0 \ , \qquad \mathrm{Var}(\Delta) = \frac{\sigma^2}{m} \ .$$

Where σ^2 is the variance of the distribution $\frac{\tau_k^1 - \tau_k^2}{2}$. In other words, the distribution of Δ is symmetric and centered around zero. The parameter m is referred to as the "sample size".

The core idea of our watermarking approach is to intentionally delay a selected set of packets to shift the Δ distribution left or right to encode the watermark bits in the timing information of the packets. Specifically, the distribution of Δ can be shifted along the x-axis to be centered on $-\alpha$ or α by decreasing or increasing Δ by α, where α is called the "shift amount". As a result, the probability of Δ being negative or positive will increase. Concretely, to embed bit 0, we decrease Δ by α. To embed bit 1, we increase Δ by α. Decreasing Δ can be done by decreasing each $\frac{\tau_k^1 - \tau_k^2}{2}$ by α (Equation 10-3). Decreasing $\frac{\tau_k^1 - \tau_k^2}{2}$ can be achieved by decreasing each τ_k^1 by α and increasing each τ_k^2 by α. It is easy to see that increasing Δ can be done in a similar way. Decreasing or increasing one IPD (τ_k^1) is achieved by delaying the first packet or the second packet of the pair, respectively.

The encoded watermark can be detected by calculating Δ and checking if Δ is positive or negative. If $\Delta > 0$, bit 1 is decoded. Otherwise (if $\Delta \leq 0$) bit 0 is decoded. This scheme can be extended to a w-bit watermark (ω_{SD}) by repeating the above process w times. During the decoding process, a w-bit watermark (ω'_{SD}) is extracted from the packet stream and if the hamming distance between ω_{SD} and ω'_{SD} is lower than a pre-defined error margin δ, we can conclude that the watermark embedded at the source S is detected at the receiver. If the watermark does not match, an attack is flagged.

Figure 10-5 shows the distribution of Δ and the corresponding distribution after shifting it by $\alpha > 0$. Since our scheme is probabilistic, there is a probability that the embedded watermark bits will be incorrectly decoded, thus leading to false alarms (false positives) or missed detection (false negatives). This is because for any $\alpha > 0$, a small portion of the distribution of Δ falls outside the range $(-\infty, \alpha]$. Therefore, if we embed bit 0, there is a small probability that the bit will be incorrectly decoded as 1. It can be seen that this probability is the same as the probability that a sample from the unshifted distribution takes a value outside the range $(-\infty, \alpha]$. Similarly, a bit encoded to be 1 can be decoded incorrectly

because samples from Δ have a small probability of falling outside the range $[-\alpha, \infty)$. However, we can tune parameters m (sample size), α (shift amount) and δ (error margin) to achieve a very high (nearly 100%) decoding success rate (Section 10.5).

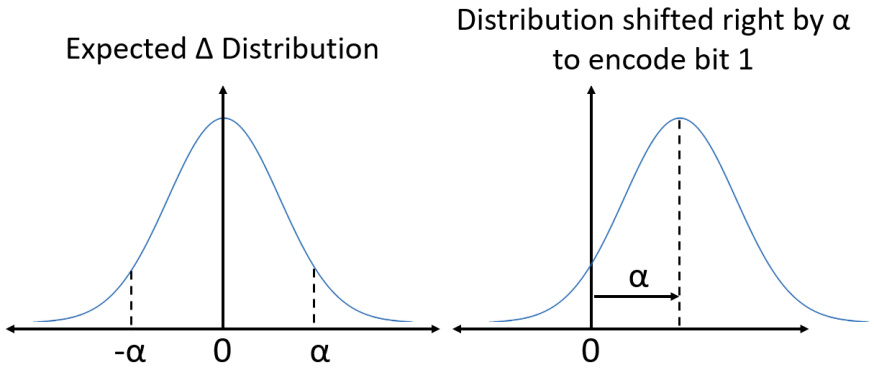


Figure 10-5. Example showing the Δ distribution shifted by α.

To provide formal guarantees, we define the "bit decoding success rate" (BDSR) as the probability of the embedded watermark bit being decoded correctly (for a shift amount of α). We denote this quantity by $\Pr[\Delta < \alpha]$. Note that the BDSR also depends on m and σ^2, but this is not explicit in the notation $\Pr[\Delta < \alpha]$ because it is implicitly captured by Δ. We now give an illustrative example to further explain this concept.

Illustrative Example: Figure 10-6 shows a sample packet stream in the time domain with packet injection times. For ease of explanation in this example, m is set to one and therefore, two packets ($2m$) are selected from the packet stream (P_{r_1} and P_{r_2}). Both packets are paired with two other packets that are x (=3) packets away in the packet stream (P_{r_1} with P_{r_1+3} and P_{r_2} with P_{r_2+3}). The IPD between each pair is calculated as $\tau_{r_1} = t_{r_1+3} - t_{r_1}$ and $\tau_{r_2} = t_{r_2+3} - t_{r_2}$. The two IPD values are then divided into two groups and Δ calculated according to Equation 10-3 as $\frac{\tau_{r_1} - \tau_{r_2}}{2}$ (sum for all m and division by m not shown since $m = 1$). We repeated the process using a packet stream that had more than 3000 packets obtained by running a simulation using the gem5 architectural simulator [18] on a real benchmark. An 8×8 Mesh NoC was modelled using the Garnet2.0 [139] interconnection network model. The node in the top left corner (node S) ran the RADIX benchmark from the

SPLASH-2 benchmark suite [142]. One memory controller was modelled and attached to the node in the bottom right corner (node D) so that the memory requests always traverse from S to D. Figure 10-7 shows the histogram collected at the NI of S for the distribution of Δ with $m = 1$ and $x = 3$. Packets were collected at random with the above parameter values to plot Δ. We can observe from Figure 10-7 that the distribution closely approximates the distribution we expected. The calculated sample mean ($\mathbb{E}[\Delta]$) for this particular example was 0.0053, which is very close to zero. Increasing the number of selected packets ($2m$) further increases the likelihood of the sample mean being zero.

The next section describes the details of the watermark encoder and decoder operations.

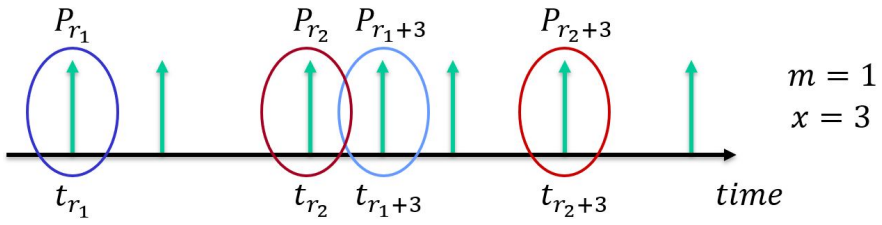


Figure 10-6. Sample packet stream with $m = 1$ and $x = 3$.

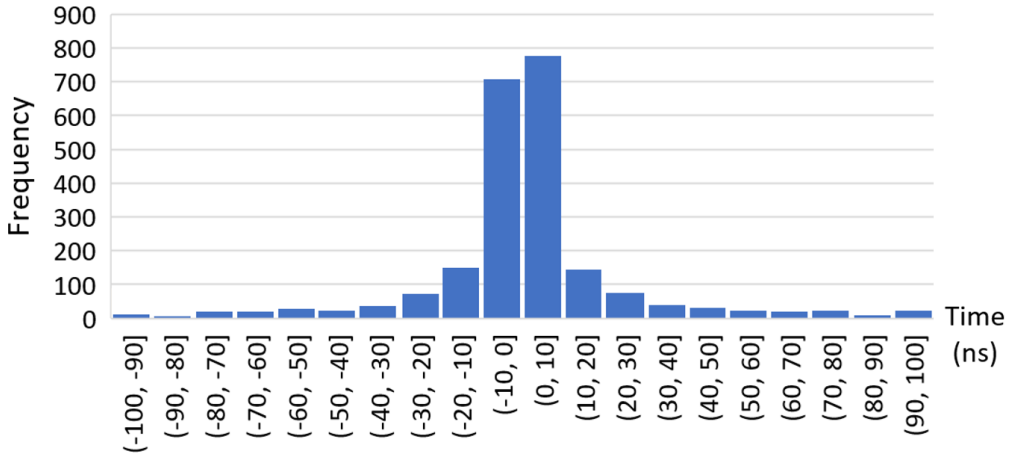


Figure 10-7. Distribution of Δ with $m = 1$ and $x = 3$.

10.3.4 Watermark Encoder and Decoder

As outlined in Section 10.3.2, our watermarking scheme includes a shared secret between S and D, which is "hard" for any other node to guess or deduce. In addition, several

parameters are shared between S and D. Specifically, S and D share the tuple $\langle m, \alpha, w_{SD}, \mathbb{K} \rangle$. The first three parameters were introduced in Section 10.3.2 as the sample size (m), the shift amount (α), and the unique watermark that represents P_{SD} (w_{SD}). The length of w_{SD} (w) can be derived from w_{SD}. In addition, $\mathbb{K}$ is a secret which is used to derive a key for the encryption scheme and a seed $\mathbb{S}$ using a key derivation function. $\mathbb{S}$ is used to seed the pseudo-random number generator which selects the $2m$ IPDs. We assume that the attacker does not know the watermark w_{SD} or secret $\mathbb{K}$, but may know m and α.

10.3.4.1 Watermark encoding process

When the watermark encoder, which is integrated in the NI of node S, receives packets from its local IP with the destination node D, it encodes the watermark according to the process outlined in Section 10.3.3 and the shared secret between S and D. The selection of the IPDs that construct the Δ distribution needs to be deterministic so that the process is identical for the watermark encoder and decoder, and it needs to ensure that an attacker cannot replicate the same behavior. To achieve this, we need a method to pair packets deterministically based on the shared secret, but that appears uniformly random to the attacker (who does not know the shared secret). We propose to implement this using a pseudo-random number generator (PRNG) seeded (i.e., initialized) with $\mathbb{S}$ (or something derived from it). This ensures that the encoder and decoder produce the *same* sequence of random numbers. Further, an attacker (who does not know the seed) cannot predict the next PRNG output, even with the knowledge of the previous output [217].

Let $\mathcal{F}$ denote the selection function that given a packet stream, selects and divides $2m$ IPDs into two groups, each of size m. We choose a window of packets and pair two random packets together from each window. Therefore, to construct $2m$ IPDs, $2m$ such packet windows are required. The operation of $\mathcal{F}$ used in our method is outlined in Algorithm 16. The PRNG seeded with $\mathbb{S}$ is used to randomly generate two integers r_z and x (line 2) such that $0 \leq r_z \leq W - 1$ and $0 < x$ and $r_z + x \leq W - 1$, where W is the size of the window. This can be done using rejection sampling to ensure that $r_z \neq x$ and then calling the smaller integer

r_z and the larger $r_z + x$. The packet at the index r_z (p_{r_z}) is paired with the packet that is x packets away giving the random pair $\{p_{r_z}, p_{r_z+x}\}$ (lines 5-6). The calculated IPD values are then evenly divided into two groups (lines 7-15).

Algorithm 16 Selection function $\mathcal{F}$

Input: Seed $\mathbb{S}$
Output: Two IPD groups used to encode one watermark bit

1: **procedure** $\mathcal{F}$
2: $r_z, x \leftarrow PRNG(\mathbb{S})$
3: **for all** $k = 1, ..., 2m$ **do**
4: $A \leftarrow \text{selectNextWindow}(P_{SD})$
5: $p_{r_z} \leftarrow A[r_z]$
6: $p_{r_z+x} \leftarrow A[r_z + x]$
7: $\tau_{r_z} \leftarrow t_{r_z+x} - t_{r_z}$
8: **if** k is odd **then**
9: $\tau_k^1 \leftarrow \tau_{r_z}$
10: **else**
11: $\tau_k^2 \leftarrow \tau_{r_z}$
12: **end if**
13: **end for**
14: **return** $[\{\tau_1^1, \tau_2^1, ..., \tau_m^1\}, \{\tau_1^2, \tau_2^2, ..., \tau_m^2\}]$
15: **end procedure**

Since $2m$ IPDs are required to encode a 1-bit watermark, w iterations of the procedure $\mathcal{F}$ are required to encode the w-bit watermark. When encoding one watermark bit, the distribution discussed in Section 10.3.3 holds only when each pair of packets is the same distance x apart from each other. Therefore, the same r_z and x values are used for each iteration of k. When encoding another watermark bit, another iteration of $\mathcal{F}$ is required in which another pair of r_z and x values will be generated by the PRNG. To ensure that the same r_z and x values are not generated for subsequent watermark bits, the PRNG must be seeded only once. An example to show how the selection function can be used to encode a w-bit watermark including how to select the window is given in Section 10.5.

10.3.4.2 Watermark decoding process

Node D upon examining the packet stream P_{SD}, decodes the w-bit watermark w'_{SD} by following the process outlined in Section 10.3.3 and the shared secret tuple. The decoder

concludes that the watermark is valid if the Hamming distance between w_{SD} (taken from the shared secret tuple) and w'_{SD} (decoded from the received packet stream P_{SD}) is less than or equal to the error margin δ. Formally, the watermark is valid if;

$$\mathcal{D}(w_{SD}, w'_{SD}) \leq \delta \tag{10-4}$$

where $\mathcal{D}$ is the Hamming distance between two bit strings and $0 \leq \delta \leq w$. The reason for allowing an error margin δ and not looking for an exact match is that no matter how large the shift amount α is, there is a probability that the watermark is decoded incorrectly as discussed in Section 10.4.1. Tuning parameter δ allows us to minimize this probability. In addition, as shown in Section 10.4.2, it allows us to minimize the impact of the attack.

10.3.5 Managing Shared Secrets

The watermark encoder and decoder operation introduced in Section 10.3.4 relies on shared secret tuples between nodes to make sure the watermarking scheme cannot be compromised. To facilitate this, an efficient way to generate and manage such secrets is required. Developing an efficient management mechanism is beyond the scope of this work and many previous studies have addressed this problem in several ways. One such example is the key management system proposed by Lebiednik et al. [163]. In their work, a separate IP called the "key distribution center" (KDC) handles the distribution of keys. Each node in the network negotiates a new key with the KDC using a pre-shared portion of memory that is known by only the KDC and the corresponding node. The node then communicates with the KDC using this unique key whenever it wants to obtain a new key. The KDC can then allocate keys and inform other nodes as required. Our proposed digital watermarking scheme can be integrated with a similar key generation and management mechanism.

10.4 Theoretical Analysis

In this section, we provide some mathematical guarantees about the correctness and security of the watermarking scheme which we further validate with experimental results in Section 10.5. First, we provide a bound on BDSR during normal operation (Section 10.4.1).

Then we evaluate the impact of an attacker on BDSR (Section 10.4.2). Finally, we present how the error margin δ can be selected such that it maximizes the chance of successfully decoding the watermark while minimizing the chances of an attack if the attacker is aware of our detection method (Section 10.4.3).

10.4.1 Watermark Bit Decoding Success Rate During Normal Operation

Given this watermark encoding/decoding scheme, it is clear that larger the shift amount α is, the higher the bit decoding success rate (BDSR) will be. However, having arbitrarily large α is not feasible in systems with real-time constraints. In this section, we show that we can achieve close to 100% BDSR for arbitrarily small α by changing the sample size m.

As discussed in Section 10.3.3, a watermark bit can be decoded incorrectly if at the receiver's end, $|\Delta| > \alpha$. Therefore, we should analyze the behavior of $\Pr[|\Delta| > \alpha]$. There are several well-established statistical tools for this, but in particular we can use concentration results, also known as tail bounds. Since the IPDs are bounded and independent, we can use Hoeffding's inequality (introduced in Section 10.3.1.1) and Equations (from Section 10.3.3) related to the distribution of Δ;

$$\Pr[|\Delta| \geq \alpha] \leq e^{\left(-\frac{m\alpha^2}{2\sigma^2}\right)}$$

$$\text{Using symmetry;} \qquad \Pr[\Delta < \alpha] \geq 1 - \frac{1}{2}e^{\left(-\frac{m\alpha^2}{2\sigma^2}\right)} \tag{10-5}$$

Therefore, we can observe that the BDSR is lower bounded by a value that depends on α and m. The results show that irrespective of the distribution of the IPDs, for arbitrarily small α values, we can always take the BDSR close to 100% by increasing the sample size m. In other words, no matter how small the shift amount α needs to be to abide by the timing constraints of the system, we can still achieve high BDSR by selecting more packets in each IPD group.

10.4.2 Impact of an Attack on the Bit Decoding Success Rate

Having established mathematical guarantees about BDSR during normal operation, we shift our focus to explore how BDSR of legitimate packet streams can be affected by an attack. According to the threat model, the Trojan infected router copies packets and

sends them to a malicious application running on a different IP. As a result, more packets are introduced to the network which can cause congestion. All packets in the network can be delayed because of this. Therefore, the attack can introduce additional delays to the legitimate packet streams. It is safe to assume that these additional delays are finite. If the attacker delays packets indefinitely through congestion, the attack is no longer an eavesdropping attack, but rather a flooding type of denial-of-service attack [85] that is beyond the scope of this chapter.

Given that the Trojan-infected router does not know which packets were selected by the watermark encoder (as explained in Section 10.3.4), the delay introduced by the attacker (whatever it is) on the selected IPDs is IID from the perspective of S and D. Using this insight, we can analyze Δ', which is the distribution after modifying Δ defined in Equation 10-3 with the added delays, and conclude that;

$$\Pr\left[\Delta' < \alpha\right] \geq 1 - \frac{1}{2} e^{\left(-\frac{m\alpha^2}{2(\sigma+\sigma_d)^2}\right)} \tag{10-6}$$

where σ_d is the added delay variance due to the added congestion. Observe that the only change is the increase in variance caused by the attacker. We can choose σ_d depending on the amount of congestion the attacker is willing to cause without risking being detected. Similar to the argument we made when reasoning about the BDSR using Equation 10-5, we can see that BDSR is lower bounded and by manipulating the sample size, we can make the BDSR arbitrarily close to 100%. Therefore, the impact on the watermarking detection is a bounded increase of variance on an otherwise 100% successful watermarking scheme. As the illustrative example that calculates BDSR in Section 10.5.2 outlines, the success rate can be brought very close to 100% even with the selection of a modest value for m.

10.4.3 Optimal Error Margin Selection

As discussed in Section 10.3.4, the use of the error margin δ instead of an exact match between the decoded and the expected watermark, allows us to tune δ to maximize the "watermark detection success rate" (WDSR). Unlike BDSR, which refers to the success of

decoding a single bit, WDSR considers the entire watermark with w bits. The probabilistic nature of our watermarking scheme leaves a small probability that the watermark will be incorrectly decoded irrespective of the values chosen for the parameters. While this probability is small, efficient selection of δ can push WDSR as close as possible to 100%. On the other hand, using a larger error margin also increases the success of potential attacks. Indeed, assuming that the attacker is aware of our detection strategy, the best strategy for an attacker to eavesdrop on data without being detected is to try to *forge* a watermark. If he succeeds, then the duplicated packets will be accepted as valid by the node that runs the accomplice application and our proposed watermarking-based defense will be defeated. We call the success probability of such a forging attack the "watermark forging success probability" (WFSP). The goal of the detection scheme is thus to set the parameters such that WDSR is maximized while minimizing WFSP. We explore how this can be achieved in this section.

10.4.3.1 Maximizing watermark detection rate

The probability of incorrectly decoding a bit was formalized using the metric BDSR as $\Pr[\Delta < \alpha]$. Considering symmetry, let $\vartheta = \Pr[-\infty < \Delta < \alpha] = \Pr[-\alpha < \Delta < \infty]$. Then for a w-bit watermark, probability of accurately decoding all w bits will be ϑ^w. Therefore, the expected WDSR can be calculated as;

$$\sum_{i=0}^{\delta} \binom{w}{i} \vartheta^{w-i}(1-\vartheta)^i \tag{10-7}$$

We can see that with a large δ, the expected WDSR increases. We observe from Equation 10-7 that;

$$\sum_{i=0}^{\delta} \binom{w}{i} \vartheta^{w-i}(1-\vartheta)^i \geq \vartheta^w$$

Therefore, we can make the expected WDSR larger than the desired WDSR by increasing ϑ. Revisiting Equation 10-6, we observe that ϑ can be made sufficiently close to 1 by increasing the sample size m irrespective of α, σ and σ_d. Therefore, we can conclude that in theory, it is possible to make WDSR close to 100% even with a modest error margin.

10.4.3.2 Minimizing risk of watermark forging attacks

While increasing δ can increase WDSR, larger the δ, larger the expected WFSP will be. We address this in two steps. First, we select watermarks such that under a given error margin δ, the probability that one watermark can be incorrectly decoded as another watermark (watermark collision) is minimized. Then, we discuss the case where an attacker, after knowing our detection mechanism, tries to inject duplicated packets such that the decoder at the receiver incorrectly validates the watermark (watermark forging) and accepts the duplicated packet steam as valid.

The problem of selecting distinct w-bit watermarks for each source-destination pair can be recast as the problem of selecting distinct codewords. This is a well-established problem that has been extensively studied in the information theory literature. Indeed, it is known that for any given set of distinct codewords, if the minimum Hamming distance between any two codewords is at least $2\delta + 1$, a nearest neighbor decoder will always decode correctly when there are δ or fewer errors [218]. Therefore, if the watermarks are chosen such that any two watermarks are at least $2\delta + 1$ distance apart, the probability of a watermark collision is minimal. We select the number of bits in the watermark w such that this property is satisfied using the method explained in Section 10.3.1.2. An example of how w is selected is given in Section 10.5.2.2.

Even if w is selected such that watermark collision probability is minimized, an attacker may still try to impersonate a legitimate sender. Assume that w_{SD} and w_{SY} are valid watermarks with distance $2\delta + 1$ (minimum possible distance between two watermarks) between nodes S and D and S and Y, respectively. A Trojan-infected router in the path from S to D duplicates packets and sends to an accomplice application in node Y. For Y to accept the duplicated packet stream as a legitimate packet stream coming from S, the watermark of the duplicated packet stream should match w_{SY}. We refer to this attack as a *watermark forging* attack.

Section 10.3.4 and Section 10.3.5 detailed how watermarks are kept unknown to any other parties, except for the sender and receiver in a packet stream, using shared secrets. Therefore, the attacker's method to forge a watermark can be reduced to a random bit flipping game with the goal of matching w_{SY}. Random bit flipping is achieved by randomly delaying the duplicated packets in P_{SD}. For the attacker to win the game, w_{SD} should change to w_{SY}. Since the minimum distance between any two watermarks is $2\delta + 1$, considering the error margin of δ, the minimum required number of bit flips is $\delta + 1$. Therefore, the attacker should flip at least $\delta + 1$ bits to win the game. However, flipping the wrong bits can take the target even further. Therefore, the best chance for the attacker to win the game is if it flips the correct $\delta + 1$ bits of w_{SD} to match w_{SY} (to end up within the error-margin of w_{SY}, i.e., within δ-Hamming distance of w_{SY}). The probability that the attacker flips the correct $\delta + 1$ bits at any given round of the game is thus: $\binom{w}{\delta+1}^{-1}$. Assuming the attacker plays n times, the attacker's probability of winning, or in other words, the probability of successfully forging the watermark (WFSP) at least once (after n attempts) is;

$$1 - \left[1 - \frac{1}{\binom{w}{\delta+1}}\right]^n \tag{10-8}$$

Observe that by manipulating w and δ, this probability can be made arbitrarily small. Furthermore, n cannot be arbitrarily large because if the probability of winning in the first few attempts is low, then the attacker will be detected before the attacker can successfully forge the watermark.

This allows us to conclude that we can make WDSR close to 100% and WFSP close to 0%. Equations 10-6, 10-7 and 10-8 combined give us the theoretical trade-off model between WDSR and WFSP. However, we cannot accommodate arbitrarily large m and w in practical scenarios. Therefore, in the next section (Section 10.5), we provide experimental evaluations and discuss realistic values that can be achieved under our threat model and architecture.

10.5 Experiments

In this section, we experimentally evaluate the theoretical models established in previous sections and choose the parameters that give the optimum results. The selected parameters are then used to explore the performance gain achieved by using our method compared to traditional AE based schemes.

10.5.1 Experimental Setup

We evaluated our approach by modeling an NoC-based SoC using the cycle-accurate full-system simulator - gem5 [18]. "GARNET2.0" interconnection network model that is integrated with gem5 was used to model an 8x8 Mesh 2D NoC [139]. To ensure the accuracy of our simulator model when compared to real hardware, we used the simulator framework proposed in [153], which has validated simulator results with results from the Intel Knights Landing (KNL) architecture (Xeon Phi 7210 hardware platform [137]), when setting up the experimental environment. Figure 10-8 shows an overview of the NoC-based SoC model. Each IP was modeled as a processor core executing a given task at 1GHz with a private L1 Cache. Eight memory controllers were modeled and attached to the IPs in the boundary providing the interface to off-chip memory. In case of a cache miss, the memory request/response messages were sent to/from memory controllers as NoC packets. The NoC was modeled with 3-stage (buffer write, route compute + virtual channel allocation + switch allocation, and link traversal) pipelined routers with wormhole switching and 4 virtual channel buffers at each input port. Packets are routed using the deadlock and livelock free, hop-by-hop, turn-based XY deterministic routing protocol.

Each processor core in the SoC was assigned an instance out of FFT, RADIX (RDX), FFM and LU benchmarks from the SPLASH-2 benchmark suite [142]. Each simulation round can in theory, give $\binom{64}{2} \times 2 = 4032$ packet streams (assuming two-way communication between any pair out of the 64 nodes) and the number of iterations that depended on the number of benchmarks (four in our case) can give $4 \times \binom{64}{2} \times 2 =$ 16,128 packet streams. However, depending on the address mapping, only some node pairs out of all the possible node pairs

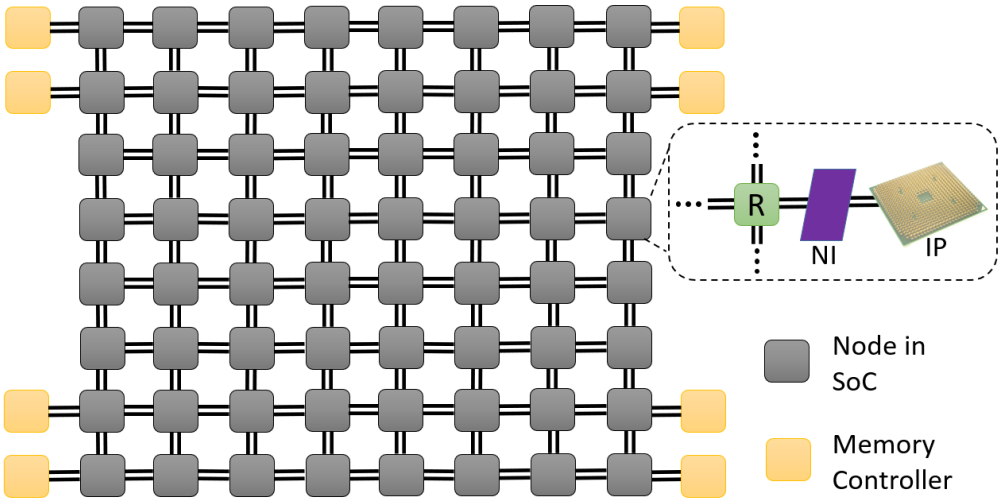


Figure 10-8. 8x8 Mesh NoC setup used to generate results.

communicate. Our simulations generated 3072 packet streams for all benchmarks between 1024 unique node pairs which we used to evaluate our method. However, to decide the number of bits in the watermark w, looking at only the number of unique node pairs is not sufficient because to avoid watermark collisions, the Hamming distance between any two watermarks should be at least $2\delta + 1$. According to Section 10.3.1.2, as δ increases, w increases as well. Therefore, more packets are required to encode the watermark and as a result, the time to detect an ongoing attack increases (more packets need to be observed before recognizing the watermark). Increasing m has a similar impact. Increasing α increases the application execution time and it takes longer to detect eavesdropping attacks. This motivates us to explore optimum parameter (m, α and δ) values such that WDSR is maximized and attack detection time, execution time as well as WFSP are minimized.

10.5.2 Parameter Tuning

We first explore m and α when encoding a single watermark bit and then extend the discussion to consider WDSR, WFSP, execution time and detection time.

10.5.2.1 Bit decoding success rate behavior with m and α

When embedding one watermark bit in a packet stream, Equation 10-5 gives a theoretical estimate of the BDSR. To compare the theoretically expected BDSR with experimental results,

we use a non-overlapping sliding window of λ packets and select $2m$ IPDs according to the method in Section 10.3.4.1. One bit is encoded in each of the 3072 selected packet steams following the same methodology and decoded at the receiver's side according to the method introduced in Section 10.3.3. $\lambda = 8$ is chosen to ensure adequate randomness in the IPD selection process. A detailed analysis of λ value selection is given in Section 10.6. We keep $\alpha = 60$ns fixed and vary m from 2 to 15. Results are shown in Figure 10-9. We compare the outcome from our experiments with the theoretical model (Equation 10-5). For example, expected BDSR for $m = 4$, $\alpha = 60$ns and $\sigma^2 = 2662$ is calculated as;

$$\Pr\left[\Delta < 60\right] \geq 1 - \frac{1}{2}e^{\left(-\frac{4\times 60^2}{2\times 2662}\right)} \approx 0.967$$

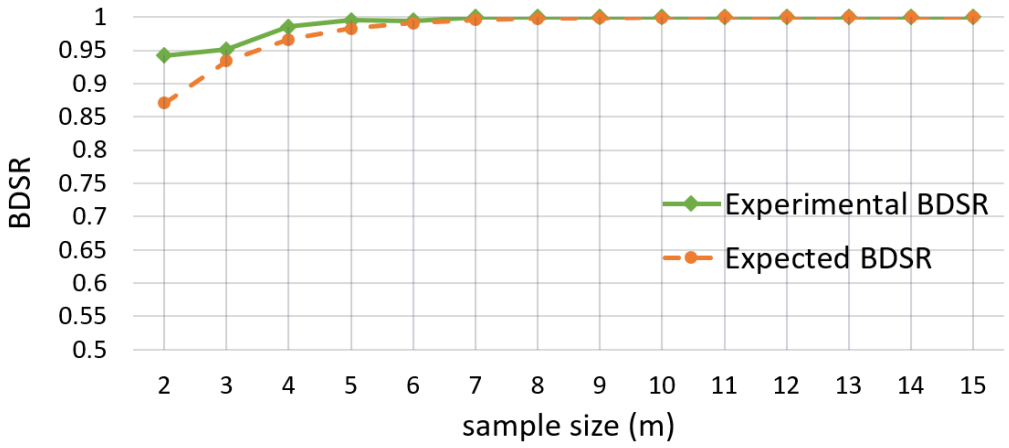


Figure 10-9. BDSR variation with sample size m. $\alpha = 60ns$.

We now fix $m = 4$ and vary α from 10ns to 100ns to explore BDSR variation with α. Figure 10-10 shows the comparison between the theoretical model (Equation 10-5) and results generated from our experiments. The experimental results in both Figure 10-9 and Figure 10-10 show that our theoretical model gives an accurate bound on BDSR. As α and m are increased, BDSR converges to 1. However, our goal is to detect any attack with high accuracy while incurring minimum performance overhead. Therefore, BDSR is not the only deciding factor. As α and m is increased, the execution time of the application/benchmark

running with our attack detection mechanism increases as well. α and m should be chosen such that this trade-off is maintained.

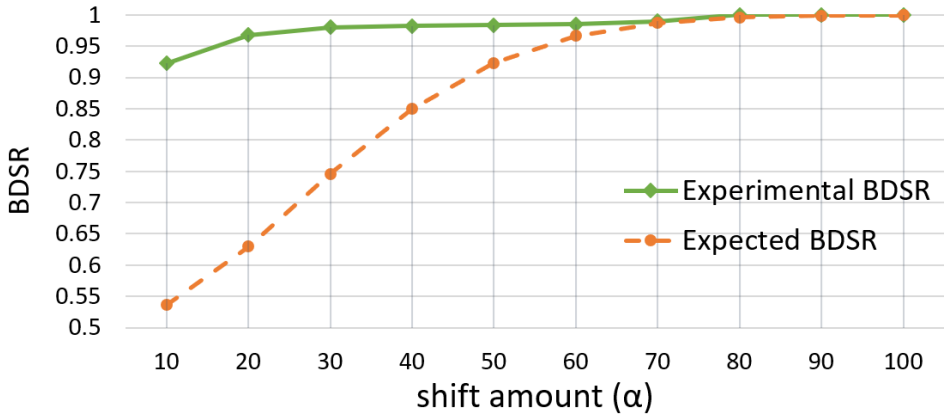


Figure 10-10. BDSR variation with shift amount α. $m = 4$.

While Figure 10-9 and Figure 10-10 show how BDSR varies with m and α, both figures had one parameter fixed while varying the other. To observe how both m and α effect the BDSR as well as the execution time, we did a grid search in the ranges $2 \leq m \leq 10$, $10 \leq \alpha \leq 80$ and $w = 20$ and eliminated cases where expected BDSR was less than 0.95 and execution time increase was more than 5%. These thresholds were chosen to achieve the optimum balance in the trade-off. Results are shown in Figure 10-11. $w = 20$ is chosen because, to provide a unique watermark for each communicating node pair (1024 in our experiments), 10 bits are required. 10 additional bits are kept to allow error margins as well as to avoid collisions. However, as discussed in Section 10.5.2.2, w can be further optimized leading to a better execution time. Execution time increase is measured as the average execution time increase as a percentage when benchmarks are run with our approach compared to Default-NoC introduced in Section 10.2. Out of the possible combinations in Figure 10-11, we pick $m = 4$ and $\alpha = 60$ as it gives an adequate trade-off for our exploration.

10.5.2.2 Choosing δ and w

With the values selected for m and α, we explore the impact of the error margin δ on WDSR. To calculate expected WDSR according to Equation 10-7, w should be decided. However, the value of w is dependant on the value we select for δ. Therefore, we explore the

α \ m	2	3	4	5	6
50	X	X	X	0.952 0.989 4.27%	0.970 0.991 4.75%
60	X	X	0.967 0.985 4.17%	0.983 0.995 4.73%	X
70	X	0.968 0.971 3.89%	0.987 0.990 4.54%	X	X
80	0.955 0.960 3.42%	0.986 0.989 4.19%	X	X	X

Figure 10-11. BDSR and execution time variation with m and α. w fixed at 20.

behavior of expected WDSR with respect to δ for several fixed w values ($w \in \{14, 16, 18, 20\}$). Results are shown in Figure 10-12. $\delta = 0$ represents exact matches between the decoded watermark and the expected watermark without using an error margin. The importance of using δ is evident when the scenario of looking for exact matches ($\delta = 0$) is compared with any other δ value. For example, for the values $\vartheta = 0.967$ and $w = 20$, WDSR with exact matches is $\vartheta^w = 51.1\%$ whereas for the same ϑ and w values with an error margin of 2 ($\delta = 2$), WDSR is 97.3%.

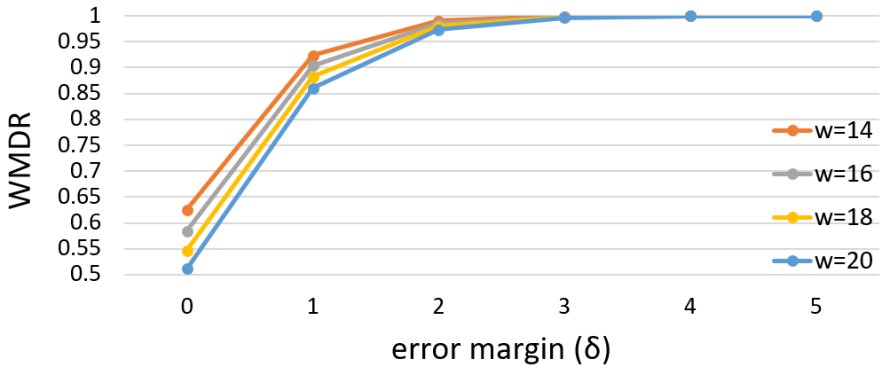


Figure 10-12. Expected WDSR variation with error margin δ for several w values. m and α fixed at 4 and 60ns, respectively.

As outlined in Section 10.4.3.2, the chosen δ value affects the chances of the attacker succeeding in a forging attack (WFSP). To evaluate the impact, we explored WDSR (Equation 10-7) and WFSP (Equation 10-8) values for different combinations of w and δ.

However not all w and δ values can co-exist if watermark collisions are to be avoided. Assume that the chosen δ value is 2. As outlined in Section 10.4.3.2, for two watermarks not to collide, they should be at least $2\delta + 1$(=5 if $\delta = 2$) Hamming distance apart. Since there are 1024 unique node pairs, we can set w as the minimum number of bits required to generate 1024 unique codewords such that the minimum Hamming distance between any two codewords is 5. In other words, we are looking for w such that $A(w, 5) \geq 1024$ according to Section 10.3.1.2. From [214], we can derive $w \geq 18$. Therefore, to ensure that there are no collisions between watermarks with an error margin of 2, at least 18 bits are required for the watermark. Similarly, we can derive $w \geq 21$, for $\delta = 3$, and $w \geq 14$ for $\delta = 1$. Since increasing w has an impact on execution time as well, for each δ value, we pick the two smallest possible w value such that there are no watermark collisions. Table 10-1 shows expected WDSR, WFSP values, experimental WDSR value and execution time increase for the selected configurations.

Table 10-1. WDSR, WFSP and execution time increase for varying w and δ. $\vartheta = 0.967$, $n = 10$

δ	w	Expected WDSR	WFSP	Experimental WDSR	Execution Time Increase
1	14	0.9238	0.1046	0.9538	3.49%
1	15	0.9139	0.0912	0.9512	3.61%
2	18	0.9797	0.0121	0.9801	3.95%
2	19	0.9765	0.0102	0.97884	4.06%
3	21	0.9955	0.0075	0.9987	4.29%
3	22	0.9946	0.0064	0.9964	4.40%

These results strongly support our claim that WFSP can be made arbitrarily small by manipulating w and δ. We observe from Figure 10-12 that WDSR converges to 1 starting $\delta = 2$. Furthermore, observing values in Table 10-1, we can pick $\delta = 2$ and $w = 18$ as a configuration that gives an adequate trade-off.

10.5.3 Performance Evaluation

With the selected parameters, $m = 4$, $\alpha = 60$, $\delta = 2$, $w = 18$, we explore the performance improvement achieved by our method compared to the traditional AE based defenses. Section 10.2 introduced two scenarios - Default-NoC and AE-NoC against which

we evaluate the performance of our approach (digital watermarking-based attack detection coupled with encryption). NoC delay and execution time comparison are shown in Figure 10-13 considering Default-NoC, AE-NoC and our watermarking based attack detection method. Our approach only increases the NoC delay by 27.9% (26.3% on average) and execution time by 5.2% (3.95% on average) compared to the default NoC whereas AE-NoC increased NoC delay by 59% (57% on average) and execution time by 17% (13% on average). Therefore, our method has the ability to significantly improve performance compared to other state-of-the-art security mechanisms intended at preventing eavesdropping attacks.

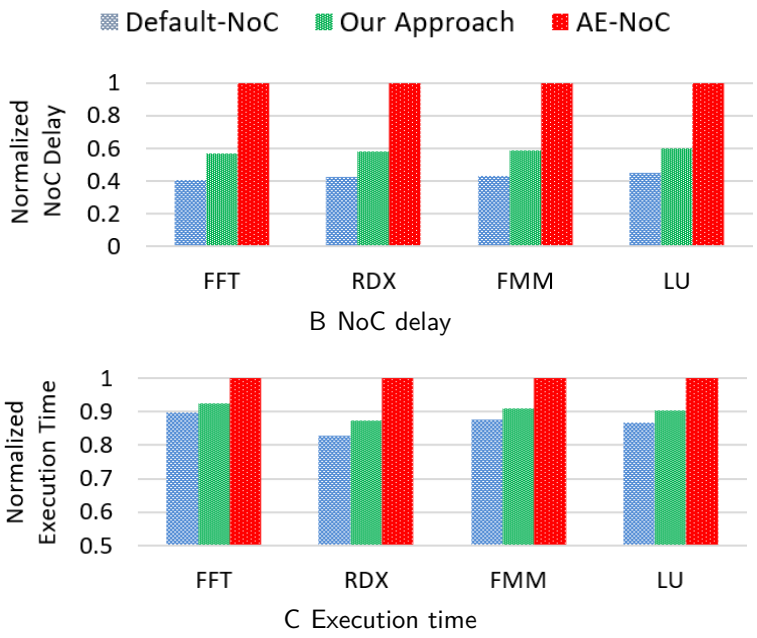


Figure 10-13. NoC delay and execution time comparison

In addition to execution time comparison, time taken to detect an ongoing attack (detection time) is also critical. Detection time is calculated as the time taken to decode the complete watermark from a packet stream. As soon as the w-bit watermark is decoded and validated, any eavesdropping attack can be detected. Table 10-2 shows detection time for each benchmark normalized to total execution time. This shows that our watermark detection scheme is capable of detecting any eavesdropping attacks in a timely manner.

Table 10-2. Attack detection time for different applications/benchmarks.

FFT	RDX	FMM	LU
6.56E-3	4.8E-5	1.9E-4	3.9E-4

In summary, these results validate our theoretical model and provide a framework to tune the parameters such that eavesdropping attacks can be detected quickly with high accuracy while providing a significant performance improvement compared to existing state-of-the-art solutions.

10.6 Discussion

The security of the watermarking scheme depends on the secrecy of some parameters (Section 10.3.4). Parameters include the watermark w_{SD} as well as the key $\mathbb{K}$ for each P_{SD}. A key distribution center (KDC) acts as a trusted dealer to distribute these parameters. In this section, we discuss security implications if some of these assumptions do not hold.

10.6.1 Eliminating the Trusted Dealer

In the absence of a trusted dealer, each communicating node pair will have to agree on a watermark and a key. While this can be facilitated by key-exchange protocols such as the Diffie-Hellman key exchange, the lack of a trusted dealer can cause duplicated watermarks (watermark collisions). If watermarks are selected uniformly at random to minimize the chances of collision, according to the birthday bound, the number of bits assigned to the watermark should be double of what is required. For example, if an 18-bit watermark is required in the presence of a trusted dealer, 36 bits are required in its absence because of the birthday bound. While our watermarking scheme can give better accuracy and less collisions for a 36-bit watermark, the execution time as well as the detection time will increase. Therefore, a designer needs to carefully select the size of the watermark to minimize the collision without violating the performance budget.

10.6.2 What Can Be Inferred from Packet Timing?

It is important to note that the watermark is encoded in the IPD values, not in the individual packet injection/received times. Furthermore, packet injection times can vary

depending on the behavior of the application as well. There can be phases in the application execution where more packets are injected to the NoC whereas in some other phases, delay between packet injections is comparatively high. Therefore, "guessing" the watermark cannot be easily accomplished by merely observing packet arrival times. Moreover, the only way for an attacker to forge the watermark successfully is to know both the watermark and the PRNG seed.

Indeed, even if the watermark could be inferred from packet timing, the PRNG seed cannot be inferred from packet timing information due to cryptographic guarantees of using a PRNG. In the next section, we assume that the watermark is known by the adversary but not the PRNG seed and analyze the probability that an attacker can forge the watermark. This probability can be reduced to a random bit flipping game (probability $= \frac{1}{2}$).

10.6.3 Watermark Is Not a Secret Anymore?

Assume that the attacker knows the watermark, but not the PRNG seed. To forge the watermark, the attacker must select the two correct packets (that forms the IPD) from each window. Observe that without the PRNG seed, the attacker's probability of correctly guessing the two packets from a given window is $1/\binom{\lambda}{2}$ (Case I). Similarly, we can derive that the probability of two packets chosen by the attacker partially overlapping with the correct two packets and the probability of the attacker not selecting either one of the two correct packets are $2(\lambda - 2)/\binom{\lambda}{2}$ (Case II) and $\binom{\lambda-2}{2}/\binom{\lambda}{2}$ (Case III), respectively. Therefore, the higher the value chosen for λ, the lower the chances of a successful attack. The probability of the attacker *not* selecting either one of the two packets correctly (Case III) goes above 0.5 at $\lambda = 8$. In the overlapping scenario, if the first packet selected by the attacker is the correct second packet (or vice versa), delaying it will give the incorrect watermark bit. However, to give a conservative estimate, we ignore that possibility and use $\lambda = 8$ so that the probability of selecting both packets incorrectly is at least $\frac{1}{2}$. This analysis shows that our watermarking scheme can be tuned to work even in scenarios with very strong security assumptions such as the watermark being leaked to the attacker. Additionally, for systems which require even

stronger security, another layer of security can be added if we rotate the watermark assigned between each pair of nodes after some number of iterations.

10.7 Summary

In this chapter, I introduced a lightweight eavesdropping attack detection mechanism using digital watermarking in NoC-based SoCs. I considered a widely explored threat model in on-chip communication architectures where a hardware Trojan-infected router in the NoC IP copies packets passing through it, and re-routes the duplicated packets to an accompanying malicious application running on another IP in an attempt to leak information. Compared to existing authenticated encryption based methods, my approach offers significant performance improvement while providing the required security guarantees. Performance improvement is achieved by replacing authentication with packet watermarking that can detect duplicated packet streams at the network interface of the receiver. I discussed the accuracy and security of my approach using theoretical models and empirically validated them. Experimental results demonstrated that my approach can significantly outperform the state-of-the-art methods.

CHAPTER 11
SECURING NOC USING MACHINE LEARNING

Malicious IPs (MIPs) can lead to a wide variety of threats such as eavesdropping attacks, data integrity attacks, etc. In this chapter, I focus on securing the system-on-chip (SoC) from MIPs that launch flooding type of denial-of-service (DoS) attacks. Previous work on mitigating flooding type of DoS attacks explored traffic latency comparison [106], traffic partitioning [107], and packet arrival monitoring [85, 100]. These approaches made an unrealistic assumption, highly predictable NoC traffic patterns, which allowed the construction of linear statistical bounds to detect DoS attacks [85, 100, 106]. Unfortunately, this assumption does not hold during many realistic scenarios that include task migration, task preemption, changing application characteristics due to major input variations, etc. In this chapter, I propose a machine learning (ML) based DoS attack detection mechanism that is capable of adapting to use cases with unpredictable NoC traffic patterns and detecting attacks with high accuracy. While ML has shown promising results for optimizing NoC power consumption [219], to the best of my knowledge, my approach is the first attempt at securing NoC-based SoCs using ML. Major contributions of this chapter are as follows;

- I propose an ML-based DoS attack detection method that trains ML models during design time and uses the trained models to classify network traffic behavior to detect flooding type of DoS attacks.
- I outline features that can be extracted from NoC traffic as well as engineered features, and experimentally evaluate the most suitable features.
- I perform a comprehensive exploration of 12 different ML models to select the best fit for the given architecture and threat models.
- My approach achieves high accuracy in DoS attack detection across different NoC traffic patterns caused by various applications and application mappings.

The rest of the chapter is organized as follows. Section 11.1 elaborates the threat model and presents related work. Section 11.2 motivates the need for ML-based detection of DoS attacks. Section 11.3 describes my ML-based DoS attack detection methodology. Section 11.4 presents the experimental results. Finally, Section 11.5 summarizes the chapter.

11.1 Threat Model

DoS attacks can happen from MIPs intentionally degrading SoC performance by flooding the NoC with packets. MIPs can target a component that is critical to SoC performance, such as a memory controller that provides the interface to off-chip memory, and inject unnecessary requests [85, 100]. As a result, the legitimate requests can experience severe delays. An example attack scenario is shown in Figure 11-1. A MIP at node 1 targets its victim at node 7 and injects additional packets. The traffic rate in routers along the routing path is increased causing NoC congestion, which leads to performance degradation and reduced energy efficiency. Since the victim receives a lot more requests than it is designed to handle, responses are delayed and that can lead to violation of task deadlines. Violation of real-time requirements can be catastrophic for safety-critical applications. A similar threat model was used in previous work that explored DoS attacks in NoC-based SoCs [85, 100, 102, 107].

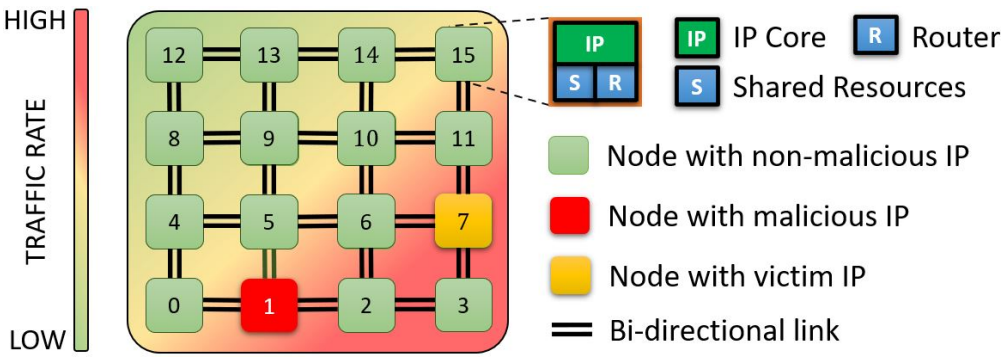


Figure 11-1. Example DoS attack from a malicious IP to a victim IP in a mesh NoC setup. The thermal map shows high traffic near the victim IP.

11.2 Motivation

To evaluate the potential of using ML to detect DoS attacks in NoC-based SoCs, we simulated both malicious and benign programs using the gem5 cycle-accurate full-system simulator [18] and extracted features from NoC traffic. A 4×4 mesh NoC was modeled using the GARNET2.0 [139] interconnection network model. The mesh consisted of 16 IP cores and 4 memory controllers as shown in Figure 11-2. Each router in the middle of the mesh is connected to four other routers in the four directions and to an IP core. When a memory

request (e.g., memory LOAD or STORE instruction) is initiated by a core during application execution, in case of a cache miss, a memory request is injected into the NoC in the form of NoC packets. Typically, the packets are further broken down into smaller units called "flits" to facilitate flow control mechanisms. The flits are routed in the appropriate virtual network (vnet), that matches the cache coherence request type, via routers and links. When the flits arrive at the memory controller, the memory fetch is initiated and once the operation is completed, the response is routed back to the original requestor. Figure 11-2 shows the architecture model used for both normal and attack scenarios. During normal execution, two processor cores ran two instances of the FFT benchmark from the SPLASH-2 benchmark suite [142]. For the attack scenario, in addition to the two applications, a MIP was modeled to inject packets at the four memory controllers uniformly, increasing the overall network traffic by 50%. More details about the experimental setup is given in Section 11.4.1.

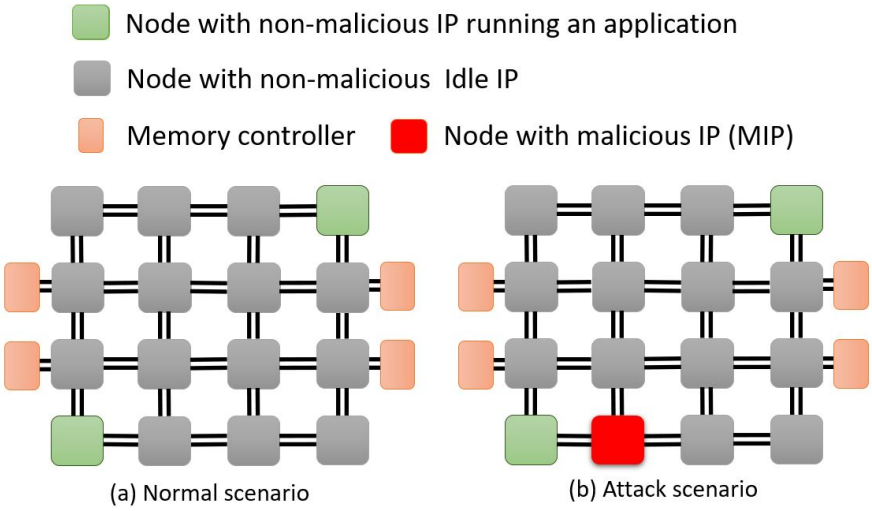


Figure 11-2. Architecture models used to extract NoC traffic features.

We extracted NoC traffic features and labeled them based on normal (target label = 0) and attack (target label = 1) scenarios. Figure 11-3 shows the correlation matrix of features extracted from NoC traffic. Each feature is denoted by a "feature ID" instead of the feature name. A detailed description of the features is given in Section 11.3. The highlighted column shows the correlation of each feature to the target label (feature ID - V). The values shown in

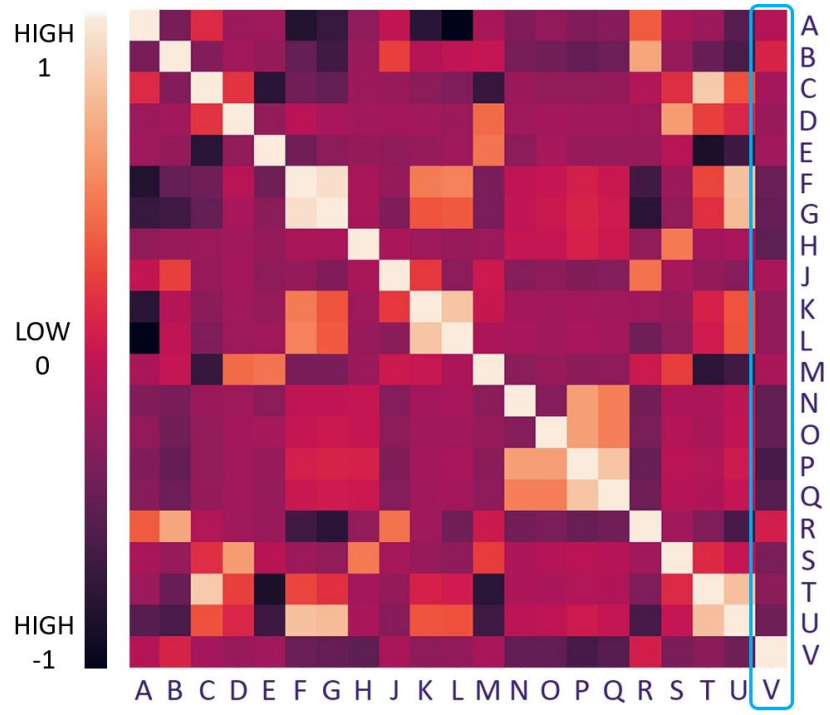


Figure 11-3. Correlation matrix of extracted NoC traffic features.

Figure 11-3 are the pairwise "Pearson Correlation Coefficients" (PCC) of all the features. PCC, calculated as:

$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X \cdot \sigma_Y}$$

gives a measure of the linear correlation between a pair of random variables (X, Y). PCC value ranges from -1 to 1. $\rho_{X,Y} = 1$ (light color shades) implies that X and Y have a linear relationship where Y increases as X increases. $\rho_{X,Y} = -1$ (dark color shades) also implies a linear relationship, but in this case, Y decreases as X increases. $\rho_{X,Y} = 0$ implies that there is no linear correlation between the variables.

We can observe the following from Figure 11-3:

- Most features are not perfectly correlated to each other and falls in the low to medium (0 ± 0.5) correlation range. In other words, the dataset does not exhibit "Multicollinearity" [220]. Multicollinearity, if existed, can severely affect performance of ML models outside of the original (training) dataset. Therefore, NoC features have the potential to train accurate classifiers.

- Since column V shows values in the range of (0 ± 0.3), we can conclude that the target label is not linearly correlated with the features. Therefore, a linear model, such as linear regression or naive bayes classifier, to differentiate normal and attack scenarios are unlikely to yield good results. Exploration of ML techniques that capture non-linear behavior, such as neural networks, decision trees or gradient boosting, is required.

These observations give us evidence that if an ML model is to be trained based on these features, it has the ability to distinguish between normal and attack traffic without causing model overfitting. Therefore, a trained ML model can potentially detect DoS attacks during runtime irrespective of the location of the MIP(s). Based on this premise, in subsequent sections, we present our ML-based runtime DoS attack detection mechanism and empirically validate our approach.

11.3 DoS Attack Detection Using Machine Learning

We propose an ML-based DoS attack detection mechanism that is trained statically during design time, and the trained models are used to detect DoS attacks during runtime. An overview of our approach is shown in Figure 11-4. During design time, NoC traffic is statically analyzed to gather the dataset that is used to train the ML models. Both normal and attack scenarios are emulated during this phase using a few known application mappings. The trained models are stored in a dedicated IP denoted as the "Security Engine" (SE). During runtime, NoC traffic data is gathered at each router using probes attached to routers and the collected data is sent to the SE using a separate physical "Service NoC". The models at the SE use data collected within a predefined time window to make inferences about the condition of the NoC. In Section 11.4, we show that our method is capable of classifying data as normal or attack, irrespective of the locations of cores running the applications and the locations of MIP(s).

Our ML-based DoS attack detection mechanism relies on the following features of the architecture model.

- Probes attached to routers can gather data from NoC packets without incurring significant performance and power overhead.
- The SoC architecture comprises of two physical NoCs: i) a Data NoC that is used to communicate between IPs for application execution, and ii) a Service NoC which transfers data collected from probes to the SE.

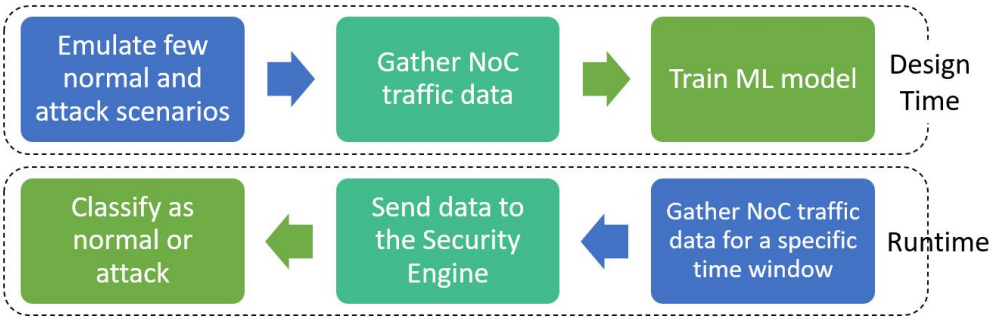


Figure 11-4. Major steps of the ML-based DoS attack detection mechanism.

The remainder of this section is organized as follows. Section 11.3.1 presents the NoC traffic features and the ML model used to make inferences. Section 11.3.2 discusses the hardware implementation to have probes connected to routers that gather data and send to the SE via the Service NoC.

11.3.1 Machine Learning Model

As outlined in Section 11.2, NoC packets/flits in our architecture model corresponds to memory requests/responses between IPs running the applications and the memory controllers. We extract information when flits are transferred through routers. The features consist of data extracted from NoC packets as well as engineered features using the extracted data. A complete list of NoC traffic features used in our exploration is shown in Table 11-1. However, as elaborated in Section 11.4.3, we experimentally eliminated some features based on feature importance[1] in an attempt to find the optimum trade-off between the least number of features and the highest model accuracy. Feature IDs of the selected features, when running the final model, are marked with a star (*) in Table 11-1.

We use "Gradient Boosting", a powerful technique to perform supervised ML classification, to classify normal and attack scenarios. It is an ensemble learner that creates the final model based on a collection of weak predictive models, decision trees in most instances, and that

[1] Feature importance gives a score to indicate how important a feature is in the decision making process of an ML model. In a trained model, the more a feature contributes to key decisions, the higher its relative importance.

Table 11-1. NoC traffic features used in our machine learning model

Feature ID	Feature Name	Feature Description
A	outport*	port used by the flit to exit the router (0-local,1-north,2-east,3-south,4-west)
B	inport	port used by the flit to enter the router (0-local,1-north,2-east,3-south,4-west)
C	cc type	cache coherence type of the packet corresponding to the flit
D	flit id	identifier used to denote each flit of a packet
E	flit type	type of flit (head, tail, body)
F	vnet	virtual network used by the flit
G	vc*	virtual channel used by the flit
H	traversal id*	identifier used to group all packet transfers related to one NoC traversal
J	hop count*	number of hops from the source to the destination
K	current hop	number of hops from the source to the current router
L	hop percentage	ratio between the current hop and the hop count
M	enqueue time*	time spent inside the router by the flit
N	packet count decr.	cumulative no. of flit arrivals within time window τ (decremented as packets arrive)
O	packet count incr.	cumulative no. of flit arrivals within time window τ (incremented as packets arrive)
P	max packet count	maximum no. of flits transferred through the router within a given time window τ
Q	packet count index	packet count incr $\times$ packet count decr
R	port index	outport $\times$ inport
S	traversal index*	cache coherence type $\times$ flit id $\times$ flit type $\times$ traversal id
T	cc vnet index	cache coherence type $\times$ vnet
U	vnet vc cc index	cache coherence vnet index $\times$ vc

results in better overall prediction capabilities due to iterative learning from each model. The key concept of the algorithm is to create new base-learners having a maximum correlation with the negative gradient of the loss function of the entire ensemble. Weaker predictive models in the ensemble are trained gradually, additively, and sequentially, and their shortcomings are identified by the use of gradients in the loss function which indicates the acceptability of the model's coefficients at fitting the underlying data. The decision to use gradient boosting for our classification was made experimentally as outlined in Section 11.4.2.

11.3.1.1 Training the ML model

The ML model is trained statically, during design time. We choose a few application mapping scenarios to train the model that includes both normal execution and attack scenarios. Figure 11-2 shows one such configuration. A list of all training and testing configurations is outlined in Section 11.4.1. Network traces are collected during application execution at each router. When flits pass through the routers, a feature vector is constructed including the selected features for each flit. All selected features are transformed using "MinMaxScaler" to fit into the range of 0 to 1, without distorting the shape of the original features. Transformed features are then used to tune the hyperparameters of the model using "Bayesian Optimization", which outputs the best-optimized list of parameters while learning from previous iterations in each iteration. This process is repeated for all 16 routers separately to train 16 models, one per each router.

11.3.1.2 Attack detection

During runtime, probes attached to the routers gather data and send to the SE for evaluation. The SE aggregates data and constructs feature vectors corresponding to each router, following a process similar to that of during model training. Let $\mathcal{M}_i$ correspond to the model trained for router r_i using gradient boosting. Feature vectors falling within a predefined time window τ_j is then used as input to each trained model, which gives a probability of an attack as the output. If $\mathcal{V}_{i,j}$ denotes the set of feature vectors constructed at r_i for τ_j, the probability of an attack is denoted by $p_{i,j}$, where $p_{i,j} \leftarrow \mathcal{M}_i(\mathcal{V}_{i,j})$. The probability is calculated as the portion of feature vectors labeled as "attack" during τ_j. If all feature vectors are classified as attack by the model, the probability is 1. If all feature vectors are classified as normal, the probability is 0. The overall probability of an attack for the time window τ_j is calculated after pooling all probabilities as;

$$\mathcal{P}_j = \frac{\sum_{\forall i}(p_{i,j} \cdot |\mathcal{V}_{i,j}|)}{\sum_{\forall i} |\mathcal{V}_{i,j}|} \tag{11-1}$$

The overall probability for the time window τ_j ($\mathcal{P}_j$) is a weighted average of probabilities from each model where the weights correspond to the number of flits transferred through each router within the given time window. If $\mathcal{P}_j$ is greater than a predefined threshold λ, an attack is flagged. This process is repeated for every τ_j during SoC operation to detect attacks that can be potentially initiated at any point in time.

Weights based on the number of flits indicate that when a model makes a decision based on a lot of data points, it can be trusted to give a more accurate result. The choice was motivated by the fact that we make no assumptions about the placement of the secure and non-secure IPs. However, if more information is available, the weighted average can be adjusted so that some models contribute more to the final decision. For example, if the locations of the non-secure (potentially malicious) IPs are known, the probabilities of models corresponding to those routers can be given more weight and it would result in a better overall performance in distinguishing normal traffic from an attack scenario. How to combine different probabilities to arrive at a single conclusion under various assumptions is well studied in the area of "Opinion Pooling", which is a part of probability theory, and can be used in our approach based on the assumptions made [221].

It is important to note that all the features we have used in our method can be extracted from the packet header or by counting flits or as a combination of header and count information. Observing the packet payload (e.g., memory data block in case of a memory data fetch packet) is not required. Therefore, our approach can be used together with other NoC security mechanisms such as encryption and authentication.[2]

[2] NoC packet encryption and authentication is typically done using "Authenticated Encryption with Associated Data" (AEAD) schemes where only the payload is encrypted and associated data, such as the packet header, is sent as plaintext to facilitate routing during NoC traversal [95].

11.3.2 Implementation of Hardware Components

Our approach relies on collecting features at routers using probes and sending the data via a separate physical NoC (Service NoC) to the SE to make inferences. In this section, we discuss the implementation of these hardware components.

11.3.2.1 Multiple physical NoCs

We identify two main types of packets to be transferred through the NoC to facilitate our ML-based DoS attack detection method: i) packets related to application execution as introduced in Section 11.2, and ii) packets related to extracted NoC features transferred from probes at routers to the SE. Instead of using different virtual networks to carry the different packets types, we propose to use two separate physical NoCs (Data NoC and Service NoC) to carry the two main types of packets. The choice is motivated by state-of-the-art commercial NoC-based SoC architectures that follow the same practise [4, 17]. Intel Knights Landing (KNL) architecture [4], widely deployed in the Intel Xeon processor family, consists of four parallel NoCs. Similarly, the TILE64 architecture by Tilera [17] features five parallel NoCs, to carry different packet types such as communication with main memory, communication with I/O devices, and user-level scalar operand and stream communication between tiles.

There is a trade-off between area and performance when considering one versus multiple NoCs. When different packet types are facilitated through the same NoC, header fields must be added to distinguish between the packets types. Furthermore, the buffer space must be shared between virtual networks. This can lead to performance degradation, specially when scaling to many-core processors. On the other hand, separate physical NoCs contribute to the area overhead. However, due to advances in manufacturing technologies, additional wiring to facilitate the NoCs incurs minimal overhead as long as the wires stay on-chip. On-chip buffer area has become the more scarce resource. If virtual networks are used, the increased buffer space due to sharing and the logic complexity to handle virtual networks can closely resemble to having a separate physical NoC. Intel and Tilera opted for separate physical NoCs for the same reasons. Yoon et al's work provides a comprehensive trade-off analysis [172]. When we

apply the analysis from [172] to fit the parameters in our work, the power and area overhead of having two physical NoCs versus one NoC are 7% and 6%, respectively.

11.3.2.2 Probes at routers and security engine

Hardware implementations for probes collecting data at routers and the SE have been explored in several prior work [101, 222]. Fiorin et al. [101] utilized probes attached to the network interfaces to collect data and send to a central processing element to detect DoS attacks. The runtime NoC monitoring and debugging framework proposed in [222] also used a similar setup where event related information is gathered at NoC routers and sent to a central unit for processing. Our security mechanism is built using a similar architecture. In our framework, the probes are event triggered on flit arrival. The probes consist of a sniffer, an event generator and an interface to the Service NoC. The sniffer extracts the features from flits and sends to the event generator to create the timestamped messages. The network interface then packetizes the messages and sends to the SE via the Service NoC. The SE completes feature engineering and combines the engineered and extracted features to construct the final feature vectors. Previous work performed detailed overhead analysis and reported minimal area overhead, for example, the probes consumed $0.05mm^2$ compared to a $0.26mm^2$ router area when synthesized with 0.13 micron technology [222]. Our overhead analysis is consistent with [222].

11.4 Experiments

This section is organized as follows. First, we describe our experimental setup (Section 11.4.1). Next, we explore several machine learning models to identify the best performing one and show why gradient boosting is the best choice (Section 11.4.2). Then, we rank feature importance according to the selected model and eliminate low priority features in an attempt to find the optimum trade-off between the number of features and model accuracy (Section 11.4.3). Finally, we show how our ML-based DoS attack detection mechanism performs across several training and testing configurations by exploring model accuracy for all the test cases in Table 11-2 (Section 11.4.4).

11.4.1 Experimental Setup

Following the realistic architecture model proposed in [153], the 4×4 mesh NoC was modeled using the "GARNET2.0" framework [139] that is integrated with the gem5 [18] cycle-accurate full-system simulator. The NoC model was implemented using X-Y routing with wormhole switching, 3-stage router pipeline (buffer write, route compute + virtual channel allocation + switch allocation, and link traversal) and 4 virtual channel buffers per input port. Each IP was modeled as a processor core executing a given task at 1GHz with a private L1 cache. Processor cores used the NoC for memory operations as outlined in Section 11.2. The four memory controllers attached to four boundary nodes of the NoC provided the interface to off-chip memory. The address space was shared equally between the memory controllers. Four benchmarks from the SPLASH-2 benchmark suite [142] (FFT, RADIX, FMM, LU) were used as application instances.

During normal operation, n IPs out of the 16 IPs in the 4×4 mesh, were chosen at random to run an instance of the benchmark (active IPs). To model the DoS attack scenario, an IP that did not run an instance of the benchmark injects memory request packets to the four memory controllers to cause performance degradation. Figure 11-2 shows one configuration of the random active, idle and malicious IP placement where $n = 2$. A complete set of training and testing configurations are listed in Table 11-2. Iteration ID (IID) 1 indicates that the model has been trained with two datasets: i) normal execution scenario with applications running on IPs 0 and 15 (N-0-15), and ii) attack scenario with an attacker at IP 1 launching a DoS attack while applications are running on IPs 0 and 15 (N-0-15-A-1). The trained model has been tested with three attack scenarios: i) N-0-15-A-7, ii) N-0-15-A-11, and iii) N-0-15-A-12. The IP numbers correspond to the node numbers given in Figure 11-1.

11.4.2 Machine Learning Model Comparison

To identify which ML model performs the best for our given architecture and threat models, we compared the performance of 12 ML models - Naive Bayes Classifer (NBC), Logistic Regression (LRN), 2-Layer Neural Network (2NN), 3-Layer Neural Network (3NN),

Table 11-2. Train and test configurations

Iteration ID (IID)	Train: Normal	Train: Attack	Test: Attack
1	N-0-15	N-0-15-A-1	N-0-15-A-7
			N-0-15-A-11
			N-0-15-A-12
2	N-0-15	N-0-15-A-1	N-0-15-A-7
	N-0-15	N-0-15-A-11	N-0-15-A-12
	N-0-9	N-0-9-A-1	N-0-9-A-7
	N-0-9	N-0-9-A-11	N-0-9-A-12
	N-0-6	N-0-6-A-1	N-0-6-A-7
	N-0-6	N-0-6-A-11	N-0-6-A-12
	N-0-4	N-0-4-A-1	N-0-4-A-7
	N-0-4	N-0-4-A-11	N-0-4-A-12
3	N-0-6-9-15	N-0-6-9-15-A-1-11	N-0-6-9-15-A-1-7
			N-0-6-9-15-A-7-11
			N-0-6-9-15-A-11-12
			N-0-6-9-15-A-7-12

4-Layer Neural Network (4NN), 5-Layer Neural Network (5NN), 6-Layer Neural Network (6NN), K-Neighbors Classifier (KNN), LightGBM Classifier (LGB), Decision Tree Classifier (DCT), Random Forest Classifier (RFC), and XGBoost Classifier (XGB). Each model was trained using the training dataset of IID 2. Figure 11-5 shows training accuracy and validation accuracy measured using an 80:20 training:validation split from the dataset at router 0 (r_0). The model comparison results at other routers manifested a similar trend (omitted from Figure 11-5 for clarity). As predicted in Section 11.2, non-linear ML models perform better than linear models with XGB showing the best results. XGBoost is an algorithm based on gradient boosting machines, that is optimized for parallel tree boosting with a better performance and faster execution speed on tabular data.

To evaluate the selected XGB model further, we use cross validation, which is a resampling process used to evaluate the performance of a trained ML model. "KFold" cross validation shuffles the dataset and splits it into k subsets, then trains on $k-1$ and evaluates on the other set iteratively (untill each subset is used as the test set). In contrast, "StratifiedKFold" cross validation shuffles the dataset and splits it into k subsets by class and uses a subset from each class in the test set emulating a representation of the entire dataset in

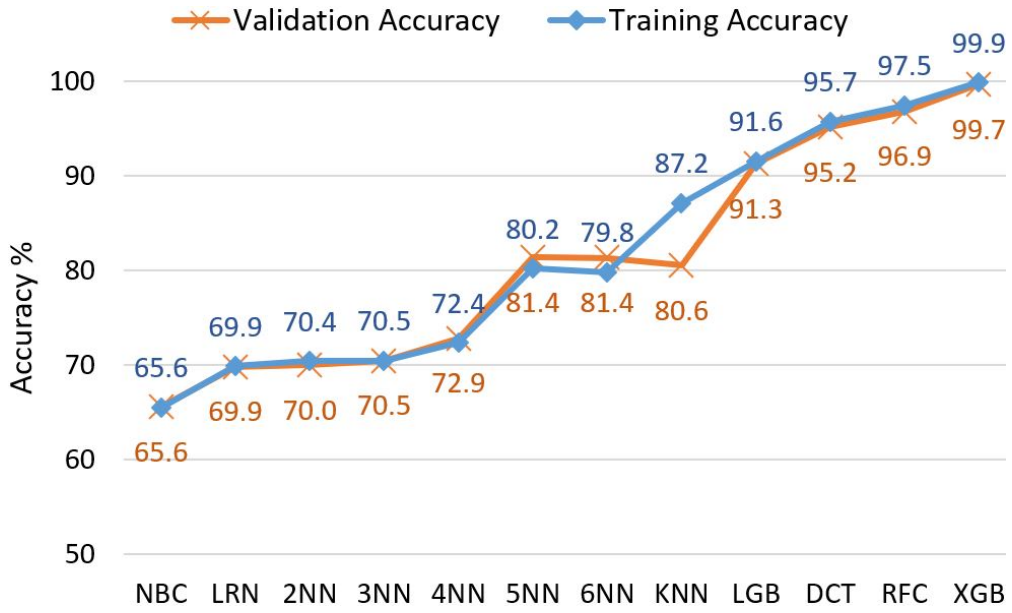


Figure 11-5. ML model performance comparison using IID 2 training dataset.

each fold for both training and validation. We use StratifiedKFold cross validation since it gives a better representation over the entire dataset. Results for 10 folds of StratifiedKFold cross validation is shown in Table 11-3. The results generated by cross validation confirms that the model is less biased, performing well in unseen data and not overfitting. Since our exploration indicated that XGB performs best in the given scenario, we use XGB as the ML model for our DoS attack detection method.

Table 11-3. Validation results of the trained XGBoost model using StratifiedKFold cross validation.

fold	1	2	3	4	5	6	7	8	9	10
accuracy %	96.84	96.86	96.84	96.88	96.80	96.92	96.84	96.54	96.71	96.90

11.4.3 Feature Importance

While using more features can certainly increase model accuracy, extracting redundant features from NoC traffic can lead to unnecessary performance and power overhead. Therefore, we eliminate features that show the least importance for the decision making process of the ML model-XGB. Table 11-4 shows the feature importance rank of each feature for the XGB model trained at each router for IID 2 dataset. Since each router runs a model trained from

the data extracted at that particular router, the feature importance rank slightly changes from router to router. However, the overall trend remains consistent where the highlighted features are the least used. Therefore, for the rest of the exploration, we eliminate the highlighted features when training and testing the accuracy of our DoS attack detection mechanism.

Table 11-4. Feature importance rank for each feature at each router with least important features highlighted.

	Router															
Feature	r_0	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}	r_{11}	r_{12}	r_{13}	r_{14}	r_{15}
outport	12	8	8	13	11	9	8	10	11	9	9	10	14	9	9	14
inport	14	12	11	12	10	11	10	14	13	12	11	12	15	11	10	11
cc type	13	15	15	14	15	13	12	16	15	13	13	15	13	14	13	13
flit id	18	18	18	18	20	20	20	19	19	20	20	20	18	18	18	19
flit type	17	17	16	16	19	18	18	18	18	19	18	19	17	17	17	16
vnet	16	19	20	20	18	19	19	20	20	18	19	18	20	20	20	20
vc	10	13	14	10	9	17	15	8	8	16	15	8	10	15	15	10
traversal id	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
hop count	6	7	7	7	7	7	7	7	7	7	7	7	7	7	7	6
current hop	9	16	17	17	14	16	16	12	10	14	16	11	12	16	16	12
hop percentage	15	9	10	11	16	10	11	13	14	10	12	17	11	10	12	15
enqueue time	8	10	9	9	8	8	9	9	9	8	8	9	8	8	8	8
packet count decr	3	4	4	5	4	5	4	4	4	4	4	4	5	5	5	5
packet count incr	5	5	6	6	5	6	6	6	5	6	6	5	6	6	6	7
max packet count	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
packet count index	4	3	3	4	3	3	3	3	3	3	3	3	4	4	4	4
port index	20	14	13	15	13	12	14	11	12	11	10	14	16	12	11	18
traversal index	7	6	5	3	6	4	5	5	6	5	5	6	3	3	3	3
cc vnet index	19	20	19	19	17	15	17	17	17	17	17	16	19	19	19	17
vnet vc cc index	11	11	12	8	12	14	13	15	16	15	14	13	9	13	14	9

11.4.4 DoS Attack Detection Accuracy

With the selected model and features, in this section, we evaluate the accuracy of our DoS attack detection method. As outlined in Section 11.3, each model outputs the attack probability independently for a given time window τ_j. The overall attack probability during τ_j ($\mathcal{P}_j$) is calculated according to Equation 11-1. Figure 11-5 and Figure 11-6 show excerpts from results generated during an attack (IID 2 and test case N-0-15-A-12) and a normal (IID 2 and test case N-0-15) scenario, respectively. Each time window is fixed at 1000 cycles ($\tau_j = 1000$). The threshold for inferring attacks from $\mathcal{P}_j$ is set to 0.5 ($\lambda = 0.5$) since an attack scenario

should give probabilities close to 1 whereas in a normal scenario, the probabilities should be close to 0. Columns "r_0" through "r_{15}" in Figure 11-5 and Figure 11-6 show the probabilities outputted by models corresponding to each router. Column "$\mathcal{P}_j$" shows the overall probability for time window τ_j calculated using Equation 11-1 and the "Status" column indicates the final decision of the ML model for each τ_j. The two excerpts show 100% accuracy since all the time windows are classified accurately. However, each test case consists of more than 3000 such time windows (3280 in the complete table corresponding to Table 11-5), which is related to the application execution time. The DoS attack detection accuracy is calculated as the portion of accurately classified time windows.

Table 11-5. Results of attack scenario for IID 2 and test case N-0-15-A-12.

τ_j	Attack Probability																P_j	Status
	r_0	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}	r_{11}	r_{12}	r_{13}	r_{14}	r_{15}		
τ_1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	ATTACK
τ_2	1	1	1	0.3	1	1	1	1	1	1	1	1	1	1	1	1	1	ATTACK
τ_3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	ATTACK
τ_4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	ATTACK
τ_5	1	1	1	1	1	1	0.7	1	1	1	1	1	1	1	1	1	1	ATTACK
τ_6	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	ATTACK
τ_7	0.2	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0.9	ATTACK
τ_8	1	1	0.8	1	1	1	0.5	0	1	1	1	1	1	1	1	1	1	ATTACK
τ_9	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	ATTACK
τ_{10}	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	ATTACK

Table 11-6. Results of normal scenario IID 2 and test case N-0-15.

τ_j	Attack Probability																P_j	Status
	r_0	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}	r_{11}	r_{12}	r_{13}	r_{14}	r_{15}		
τ_1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.0	NORMAL
τ_2	0	0	0	0.7	0	0	0	0	0	0	0	0	0	0	0	0	0.0	NORMAL
τ_3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.0	NORMAL
τ_4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.0	NORMAL
τ_5	0	0	0	0	0	0	0	0	0	0	0	0	0	0.8	0	0	0.0	NORMAL
τ_6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.0	NORMAL
τ_7	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0.0	NORMAL
τ_8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.0	NORMAL
τ_9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.0	NORMAL
τ_{10}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.0	NORMAL

Figure 11-6 shows DoS attack detection accuracy for all test cases shown in Table 11-2. In IID 1, the model is trained with only two datasets (N-0-15 and N-0-15-A-1) and tested with varying MIP locations (7, 11 and 12). Even though the number of training datasets is low, the ML model still achieves an accuracy of ∼90%. As the number of training datasets is increased, the model achieves very high accuracy (∼99%), even when tested with MIP locations which the model was not trained on. Training and testing datasets in IID 3 corresponds to four applications (compared to two applications in IID 1 and IID 2) running on the SoC while two IPs are assumed to be malicious. Results show that our approach is capable of detecting DoS attacks with high accuracy irrespective of the number or the placement of MIPs and the number of applications running on the SoC. High attack detection accuracy is achieved not only if active and malicious IP placements match the training configurations, but also in new MIP placements, which the model has not been trained on.

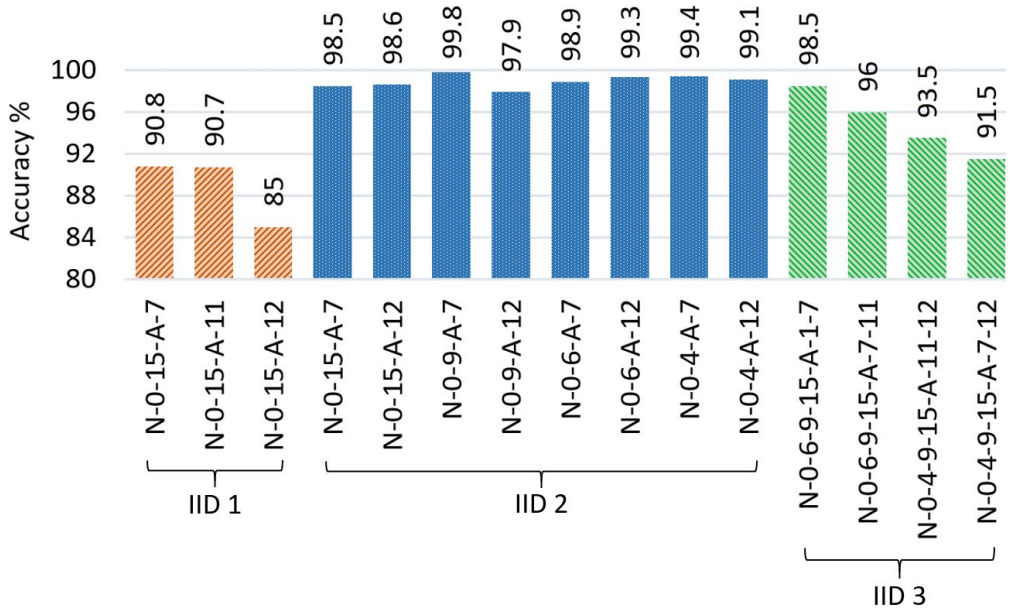


Figure 11-6. DoS attack detection accuracy for all test cases in Table 11-2.

To explore the behavior of our method across different applications, we trained the model on IID 1 with the FFT benchmark and tested on test case N-0-15-A-7 with LU, FMM and RADIX running as application instances. Results in Figure 11-7 show that even though the

model is not trained on a particular application (traffic pattern), it is capable of detecting attacks with high accuracy.

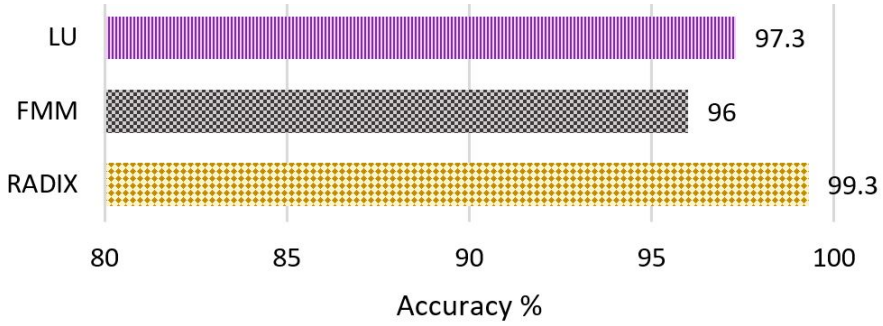


Figure 11-7. DoS attack detection accuracy across different applications for IID 2, test case N-0-15-A-7.

11.5 Summary

In this chapter, I introduced a machine learning based DoS attack detection mechanism for NoC-based SoCs. I considered a widely explored threat model where a malicious IP floods the NoC with a large number of packets causing deadline violations, performance degradation or reduced energy efficiency. Unlike existing DoS attack detection methods that rely on highly predictable NoC traffic patterns and specific use cases, my approach is capable of detecting DoS attacks with high accuracy in the presence of unpredictable NoC traffic patterns caused by diverse applications with input variations and application mappings. Experimental results demonstrated that non-linear models, such as gradient boosting, produce the best results for the given architecture and threat models. My observations from ML model performance and feature importance reveal that the key to achieving high accuracy is to carefully craft features out of the data extracted from NoC traffic. My approach is capable of detecting DoS attacks with high accuracy in a wide variety of scenarios.

CHAPTER 12
CONCLUSIONS AND FUTURE WORK

12.1 Conclusions

In this dissertation, I have presented a comprehensive investigation of security vulnerabilities and countermeasures in NoC-based SoCs. My work first introduced an accurate NoC model (Chapter 3) to enable exploration of optimization opportunities (Chapter 4) as well as realistic evaluation of lightweight security countermeasures. Next, I proposed several security countermeasures that fall into the broad categories of *design-for-security* and *runtime monitoring* solutions. Specifically, this dissertation made the following contributions. Chapter 5 presented a lightweight incremental encryption scheme that can provide confidentiality of NoC packets. Chapter 6 proposed a lightweight encryption and anonymous routing mechanism in NoC-based SoCs. Chapter 7 presented a framework for real-time detection and localization of DoS attacks. In Chapter 8, I proposed a trust-aware routing protocol in the presence of malicious IPs. Chapter 9 presented a reconfigurable security architecture that can be tuned based on use-case scenarios as well as changing circumstances. In Chapter 10, I proposed a digital watermarking based malicious IP detection method to address eavesdropping attacks. Finally, Chapter 11 presented a DoS attack detection method using machine learning.

12.2 Future Research Directions

This dissertation addressed security challenges in NoC-based SoC architectures. The future giga and tera-scale architectures can impose new challenges and opportunities. The introduction of emerging NoC technologies such as wireless and optical have already shown promising results. However, it is a major challenge to develop low-cost and flexible security solutions with minimal impact on area, performance and energy. The work proposed in this dissertation can be extended to the following directions.

- **Security and privacy analytics using machine learning**: The intersection of machine learning and security has not been given adequate attention in an NoC context. Apart from a few runtime monitoring techniques that used machine learning concepts, this area is still in its infancy. Tools such as *Cisco Encrypted Traffic Analytics* [223] utilize machine learning to detect threats by observing traffic behavior and unencrypted packet

header information. It has shown promising results in the computer networks domain. Models that can be trained offline and detect threats during runtime has the potential to provide security guarantees, especially for real-time and safety-critical applications.

- **Assertion based security**: The two main techniques used to validate NoC packets and components so far are formal verification and simulation based techniques. While formal methods can provide security guarantees, the complexity of NoC designs make the exploration space grow exponentially. On the other hand, simulation based techniques cannot provide 100% security guarantees. Assertion based security validation is considered to be a middle ground that uses best of both worlds. Assertions have been extensively used for functional validation. The applicability of assertions to verify non-functional requirements (e.g., to monitor NoC security vulnerabilities) is still an open problem for future investigation.

- **Security of emerging NoC architectures:** The increased usage of emerging NoC technologies have motivated researchers to explore security in optical, wireless and 3D NoC architectures. The applicability of the proposed ideas to emerging NoC technologies is an interesting future research direction. The inherent characteristics of emerging NoC technologies can create unique security vulnerabilities as well. For example, wireless NoCs inherently use broadcast message to communicate between nodes. In such a scenario, eavesdropping attacks can become more prominent. Therefore, an evaluation of what modifications (if any) are needed to the proposed approaches to fit the characteristics of each domain is worth exploring.

- **Seamless integration of security mechanisms**: While existing literature has discussed different threat models, it is naive to think that mitigating one particular type of threat will secure the SoC. For example, defending against eavesdropping attacks does not guarantee that eavesdropping is the only possible attack in that particular architecture. Developing security mechanisms for different threat models is a promising starting point. However, seamless integration of a suite of security mechanisms is required to secure the hardware root of trust. For example, Intel SGX (Software Guard Extensions) [224] provides hardware based software protection techniques. Future research needs to explore how to integrate several NoC security mechanisms and ensure their inter-operability in hardware, firmware and software layers in order to enable a truly secure cuberspace.

APPENDIX
LIST OF PUBLICATIONS

Book Chapters:

1. **Subodha Charles** and Prabhat Mishra, Network-on-Chip Validation and Debug, *Post-Silicon Validation and Debug*, P. Mishra and F. Farahmandi (editors), Springer, 2018.

Peer-Reviewed Journal Articles:

1. **Subodha Charles**, Yangdi Lyu and Prabhat 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 (TCAD)*, 2020.
2. **Subodha Charles** and Prabhat Mishra, Reconfigurable Network-on-Chip Security Architecture, *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 2020.
3. **Subodha Charles**, Alif Ahmed, Umit Ogras and Prabhat Mishra, Efficient Cache Reconfiguration using Machine Learning in NoC-based Many-Core CMPs, *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 2019.

Peer-Reviewed Conference Papers:

1. **Subodha Charles** and Prabhat Mishra, Securing Network-on-Chip Using Incremental Cryptography, *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2020.
2. **Subodha Charles** and Prabhat Mishra, Lightweight and Trust-aware Routing in NoC-based SoCs, *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2020.
3. **Subodha Charles**, Megan Logan and Prabhat Mishra, Lightweight Anonymous Routing in NoC based SoCs, *Design, Automation and Test in Europe (DATE)*, 2020.
4. **Subodha Charles**, Yangdi Lyu and Prabhat Mishra, Real-time Detection and Localization of DoS Attacks in NoC based SoCs, *Design Automation and Test in Europe (DATE)*, 2019.
5. **Subodha Charles**, Hadi Hajimiri and Prabhat Mishra, Proactive Thermal Management using Memory-based Computing in Multicore Architectures, *International Green and Sustainable Computing Conference (IGSC)*, 2018.
6. **Subodha Charles**, Chetan A. Patil, Umit Ogras and Prabhat Mishra, Exploration of Memory Cluster Modes in Directory-Based Many-Core CMPs, *IEEE/ACM International Symposium on Networks-on-chip (NOCS)*, 2018.

Patents and Copyrights:

1. Prabhat Mishra and **Subodha Charles**, Reconfigurable Network-on-Chip Security Architecture, U.S. Provisional Patent Application Serial No. 62/937,858, filed November 20, 2019.

2. **Subodha Charles** and Prabhat Mishra, Lightweight Encryption and Anonymous Routing in NoC based SoCs, U.S. Provisional Patent Application Serial No. 62/879,657, filed July 29, 2019.

3. **Subodha Charles** and Prabhat Mishra, Lightweight and Trust-aware Routing in NoC based SoC Architectures, U.S. Provisional Patent Application Serial No. 62/878,147, filed July 24, 2019.

4. Prabhat Mishra, **Subodha Charles** and Yangdi Lyu, Securing System-on-Chip using Incremental Cryptography, U.S. Provisional Patent Application Serial No. 62/874,187, filed July 15, 2019.

5. Prabhat Mishra, **Subodha Charles** and Yangdi Lyu, Real-Time Detection and Localization of DoS Attacks in NoC based SoC Architectures, U.S. Provisional Patent Application Serial No. 62/868,258, filed June 28, 2019.

REFERENCES

[1] IDC, "The growth in connected iot devices is expected to generate 79.4 zb of data in 2025, according to a new idc forecast," 2019.

[2] D. Evans, "The internet of things: How the next evolution of the internet is changing everything," *CISCO white paper*, vol. 1, no. 2011, pp. 1–11, 2011.

[3] "2015 International Technology Roadmap for Semiconductors (ITRS)," www.semiconductors.org/main/2015_international_technology_roadmap_for_semiconductors_itrs/, [Online].

[4] A. Sodani, R. Gramunt, J. Corbal, H. Kim, K. Vinod, S. Chinthamani, S. Hutsell, R. Agarwal, and Y. Liu, "Knights landing: Second-generation intel xeon phi product," *IEEE Micro*, vol. 36, no. 2, pp. 34–46, 2016.

[5] S. Saidi, R. Ernst, S. Uhrig, H. Theiling, and B. D. de Dinechin, "The shift to multicores in real-time and safety-critical systems," in *2015 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*. IEEE, 2015, pp. 220–229.

[6] G. Inc., "Semiconductor design ip revenue, chip infrastructure, worldwide, 2012 and 2013," *Gartner Research*, pp. 70–78, March 2014.

[7] P. Mishra, S. Bhunia, and M. Tehranipoor, *Hardware IP security and Trust*. Springer, 2017.

[8] R. A. Martin, "Common weakness enumeration," *Mitre Corporation*, 2007.

[9] "DARPA System Security Integrated Through Hardware and Firmware (SSITH) 2017," https://www.fbo.gov, [Online].

[10] S. Skorobogatov and C. Woods, "Breakthrough silicon scanning discovers backdoor in military chip," in *Cryptographic Hardware and Embedded Systems – CHES 2012*, E. Prouff and P. Schaumont, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 23–40.

[11] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 1–19.

[12] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown: Reading kernel memory from user space," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018.

[13] "ARM: 'Amba specification', Technical report, ARM, Revision 2.0, 1999," developer.arm.com/products/architecture/amba-protocol, [Online].

[14] "IBM Core Connect," http://www.chips.ibm.com/product/coreconnect/docs/crconwp.pdf, [Online].

[15] L. Benini and G. De Micheli, "Networks on chips: A new soc paradigm," *Computer*, vol. 35, no. 1, p. 70–78, Jan. 2002. [Online]. Available: https://doi.org/10.1109/2.976921

[16] W. J. Dally and B. Towles, "Route packets, not wires: on-chip interconnection networks," in *Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232)*, 2001, pp. 684–689.

[17] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C. Miao, J. F. Brown III, and A. Agarwal, "On-chip interconnection architecture of the tile processor," *IEEE Micro*, vol. 27, no. 5, pp. 15–31, 2007.

[18] 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," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, p. 1–7, Aug. 2011. [Online]. Available: https://doi.org/10.1145/2024716.2024718

[19] S. Kumar, A. Jantsch, J.-P. Soininen, M. Forsell, M. Millberg, J. Oberg, K. Tiensyrja, and A. Hemani, "A network on chip architecture and design methodology," in *Proceedings IEEE Computer Society Annual Symposium on VLSI. New Paradigms for VLSI Systems Design. ISVLSI 2002*. IEEE, 2002, pp. 117–124.

[20] P. T. Wolkotte, G. J. Smit, G. K. Rauwerda, and L. T. Smit, "An energy-efficient reconfigurable circuit-switched network-on-chip," in *19th IEEE International Parallel and Distributed Processing Symposium*. IEEE, 2005, pp. 8–pp.

[21] D. Sigüenza-Tortosa, T. Ahonen, and J. Nurmi, "Issues in the development of a practical noc: the proteo concept," *Integration*, vol. 38, no. 1, pp. 95–105, 2004.

[22] M. Dall'Osso, G. Biccari, L. Giovannini, D. Bertozzi, and L. Benini, "Xpipes: a latency insensitive parameterized network-on-chip architecture for multi-processor socs," in *2012 IEEE 30th International Conference on Computer Design (ICCD)*. IEEE, 2012, pp. 45–48.

[23] K. Goossens, J. Dielissen, and A. Radulescu, "Æthereal network on chip: concepts, architectures, and implementations," *IEEE Design & Test of Computers*, vol. 22, no. 5, pp. 414–421, 2005.

[24] T. Bjerregaard and S. Mahadevan, "A survey of research and practices of network-on-chip," *ACM Computing Surveys (CSUR)*, vol. 38, no. 1, pp. 1–es, 2006.

[25] A. Agarwal, C. Iskander, and R. Shankar, "Survey of network on chip (noc) architectures & contributions," *Journal of engineering, Computing and Architecture*, vol. 3, no. 1, pp. 21–27, 2009.

[26] É. Cota, A. de Morais Amory, and M. S. Lubaszewski, "Noc basics," in *Reliability, Availability and Serviceability of Networks-on-Chip*. Springer, 2012, pp. 11–24.

[27] A. V. de Mello, L. C. Ost, F. G. Moraes, and N. L. V. Calazans, "Evaluation of routing algorithms on mesh based nocs," *PUCRS, Av. Ipiranga*, p. 22, 2004.

[28] H. Zhu, P. P. Pande, and C. Grecu, "Performance evaluation of adaptive routing algorithms for achieving fault tolerance in noc fabrics," in *2007 IEEE International Conf. on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 2007, pp. 42–47.

[29] S. Pasricha and N. Dutt, *On-chip communication architectures: system on chip interconnect*. Morgan Kaufmann, 2010.

[30] W. Steinhögl, G. Schindler, G. Steinlesberger, and M. Engelhardt, "Size-dependent resistivity of metallic wires in the mesoscopic range," *Physical Review B*, vol. 66, no. 7, p. 075414, 2002.

[31] N. Srivastava and K. Banerjee, "Performance analysis of carbon nanotube interconnects for vlsi applications," in *ICCAD-2005. IEEE/ACM International Conference on Computer-Aided Design, 2005.* IEEE, 2005, pp. 383–390.

[32] A. Naeemi, R. Sarvari, and J. D. Meindl, "On-chip interconnect networks at the end of the roadmap: Limits and nanotechnology opportunities," in *2006 International Interconnect Technology Conference*. IEEE, 2006, pp. 201–203.

[33] G. Chen, H. Chen, M. Haurylau, N. A. Nelson, D. H. Albonesi, P. M. Fauchet, and E. G. Friedman, "On-chip copper-based vs. optical interconnects: Delay uncertainty, latency, power, and bandwidth density comparative predictions," in *2006 International Interconnect Technology Conference*. IEEE, 2006, pp. 39–41.

[34] N. Srivastava and K. Banerjee, "A comparative scaling analysis of metallic and carbon nanotube interconnections for nanometer scale vlsi technologies," in *Proc. 21st Intl. VLSI Multilevel Interconnect Conf*, 2004, pp. 393–398.

[35] S. I. Association *et al.*, "International technology roadmap for semiconductors (itrs), 2003 edition," *Incheon, Korea, Dec*, 2011.

[36] S. Deb, A. Ganguly, P. P. Pande, B. Belzer, and D. Heo, "Wireless noc as interconnection backbone for multicore chips: Promises and challenges," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 2, no. 2, pp. 228–239, 2012.

[37] S. Pasricha and N. Dutt, "Orb: An on-chip optical ring bus communication architecture for multi-processor systems-on-chip," in *2008 Asia and South Pacific Design Automation Conference*. IEEE, 2008, pp. 789–794.

[38] L. Zhang, "Optical network-on-chip architectures and designs," 2011.

[39] "SONICS NoCk-Lock Security," www.sonicsinc.com/wp-content/uploads/NoC-Lock.pdf, [Online].

[40] "Alteris FlexNoC Resilience Package," www.arteris.com/flexnoc-resilience-package-functional-safety, [Online].

[41] F. Farahmandi, Y. Huang, and P. Mishra, *System-on-Chip Security: Validation and Verification*. Springer Nature, 2019.

[42] K. Xiao, A. Nahiyan, and M. Tehranipoor, "Security rule checking in ic design," *Computer*, vol. 49, no. 08, pp. 54–61, aug 2016.

[43] S. Bhunia, M. S. Hsiao, M. Banga, and S. Narasimhan, "Hardware trojan attacks: threat analysis and countermeasures," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1229–1247, 2014.

[44] M. Tehranipoor and F. Koushanfar, "A survey of hardware trojan taxonomy and detection," *IEEE design & test of computers*, vol. 27, no. 1, 2010.

[45] S. Bhunia and M. Tehranipoor, *The Hardware Trojan War*. Springer, 2018.

[46] D. M. Ancajas, K. Chakraborty, and S. Roy, "Fort-nocs: Mitigating the threat of a compromised noc," in *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2014, pp. 1–6.

[47] K. Shuler, "Majority of leading china semiconductor companies rely on arteris network-on-chip interconnect ip," 2013.

[48] "Arteris makes big gains on inc. 500 list of america's fastest-growing private companies," www.arteris.com/Inc-500-Arteris-pr-2013-august-20, August 2013, [Online].

[49] V. Y. Raparti and S. Pasricha, "Lightweight mitigation of hardware trojan attacks in noc-based manycore computing," in *Proceedings of the 56th Annual Design Automation Conference 2019*. ACM, 2019, p. 48.

[50] S. T. King, J. Tucek, A. Cozzie, C. Grier, W. Jiang, and Y. Zhou, "Designing and implementing malicious hardware." *Leet*, vol. 8, pp. 1–8, 2008.

[51] T. Bui, S. P. Rao, M. Antikainen, V. M. Bojan, and T. Aura, "Man-in-the-machine: exploiting ill-secured communication inside the computer," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, 2018, pp. 1511–1525.

[52] S. Zander, G. Armitage, and P. Branch, "A survey of covert channels and countermeasures in computer network protocols," *IEEE Communications Surveys & Tutorials*, vol. 9, no. 3, pp. 44–57, 2007.

[53] Bloomberg, *The Big Hack: How China Used a Tiny Chip to Infiltrate U.S. Companies*, https://www.bloomberg.com/news/features/2018-10-04/the-big-hack-how-china-used-a-tiny-chip-to-infiltrate-america-s-top-companies.

[54] C. Cowan, P. Wagle, C. Pu, S. Beattie, and J. Walpole, "Buffer overflows: Attacks and defenses for the vulnerability of the decade," in

DARPA Information Survivability Conference and Exposition,, vol. 3. Los Alamitos, CA, USA: IEEE Computer Society, jan 2000, p. 1119. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/DISCEX.2000.821514

[55] S. Lukovic and N. Christianos, "Enhancing network-on-chip components to support security of processing elements," in *Proceedings of the 5th Workshop on Embedded Systems Security*, ser. WESS '10. New York, NY, USA: Association for Computing Machinery, 2010. [Online]. Available: https://doi.org/10.1145/1873548.1873560

[56] W. Yu, O. A. Uzun, and S. Köse, "Leveraging on-chip voltage regulators as a countermeasure against side-channel attacks," in *Proceedings of the 52nd Annual Design Automation Conference*, 2015, pp. 1–6.

[57] Z. H. Jiang, Y. Fei, and D. Kaeli, "A novel side-channel timing attack on gpus," in *Proceedings of the on Great Lakes Symposium on VLSI 2017*, 2017, pp. 167–172.

[58] M. Arora, "How secure is aes against brute force attacks?. freescale semiconductor, usa," 2012.

[59] J. W. Lee, M. C. Ng, and K. Asanovic, "Globally-synchronized frames for guaranteed quality-of-service in on-chip networks," in *2008 International Symposium on Computer Architecture*, 2008, pp. 89–100.

[60] D. Abts, N. D. Enright Jerger, J. Kim, D. Gibson, and M. H. Lipasti, "Achieving predictable performance through better memory controller placement in many-core cmps," in *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ser. ISCA '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 451–461. [Online]. Available: https://doi.org/10.1145/1555754.1555810

[61] T. C. Xu, P. Liljeberg, and H. Tenhunen, "Optimal memory controller placement for chip multiprocessor," in *2011 Proceedings of the Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2011, pp. 217–226.

[62] J. Howard, S. Dighe, Y. Hoskote, S. Vangal, D. Finan, G. Ruhl, D. Jenkins, H. Wilson, N. Borkar, G. Schrom, F. Pailet, S. Jain, T. Jacob, S. Yada, S. Marella, P. Salihundam, V. Erraguntla, M. Konow, M. Riepen, G. Droege, J. Lindemann, M. Gries, T. Apel, K. Henriss, T. Lund-Larsen, S. Steibl, S. Borkar, V. De, R. V. D. Wijngaart, and T. Mattson, "A 48-core ia-32 message-passing processor with dvfs in 45nm cmos," in *2010 IEEE International Solid-State Circuits Conference - (ISSCC)*, 2010, pp. 108–109.

[63] P. Eitschberger, J. Keller, F. Thiele, and C. Kessler, "Exploring the placement of memory controllers on manycore processors: A case study for intel scc," in *Sixth Swedish Workshop on Multi-Core Computing MCC*, 2013, pp. 71–74.

[64] M. Awasthi, D. Nellans, K. Sudan, R. Balasubramonian, and A. Davis, "Handling the problems and opportunities posed by multiple on-chip memory controllers," in *2010 19th*

International Conference on Parallel Architectures and Compilation Techniques (PACT). IEEE, 2010, pp. 319–330.

[65] K. Duraisamy, Y. Xue, P. Bogdan, and P. P. Pande, "Multicast-aware high-performance wireless network-on-chip architectures," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 3, pp. 1126–1139, 2017.

[66] P. Conway and B. Hughes, "The amd opteron northbridge architecture," *IEEE Micro*, vol. 27, no. 2, pp. 10–21, 2007.

[67] A. Ros, M. E. Acacio, and J. M. García, "Dealing with traffic-area trade-off in direct coherence protocols for many-core cmps," in *Advanced Parallel Processing Technologies*, Y. Dou, R. Gruber, and J. M. Joller, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 11–27.

[68] M. Schuchhardt, A. Das, N. Hardavellas, G. Memik, and A. Choudhary, "The impact of dynamic directories on multicore interconnects," *Computer*, vol. 46, no. 10, pp. 32–39, 2013.

[69] B. Cuesta, A. Ros, M. E. Gómez, A. Robles, and J. Duato, "Increasing the effectiveness of directory caches by deactivating coherence for private memory blocks," in *2011 38th Annual International Symposium on Computer Architecture (ISCA)*, 2011, pp. 93–103.

[70] J. Zebchuk, V. Srinivasan, M. K. Qureshi, and A. Moshovos, "A tagless coherence directory," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2009, pp. 423–434.

[71] S. V. R. Chittamuru, S. Desai, and S. Pasricha, "Swiftnoc: A reconfigurable silicon-photonic network with multicast-enabled channel sharing for multicore architectures," *J. Emerg. Technol. Comput. Syst.*, vol. 13, no. 4, Jun. 2017. [Online]. Available: https://doi.org/10.1145/3060517

[72] R. Morris, A. K. Kodi, and A. Louri, "3d-noc: Reconfigurable 3d photonic on-chip interconnect for multicores," in *2012 IEEE 30th International Conference on Computer Design (ICCD)*, 2012, pp. 413–418.

[73] C. Zhang, F. Vahid, and W. Najjar, "A highly configurable cache for low energy embedded systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 4, no. 2, pp. 363–387, 2005.

[74] A. Malik, B. Moyer, and D. Cermak, "A low power unified cache architecture providing power and performance flexibility (poster session)," in *Proceedings of the 2000 international symposium on Low power electronics and design (ISLPED)*. ACM, 2000, pp. 241–243.

[75] M. Modarressi, S. Hessabi, and M. Goudarzi, "A reconfigurable cache architecture for object-oriented embedded systems," in *Canadian Conference on Electrical and Computer Engineering (CCECE) 2006*. IEEE, 2006, pp. 959–962.

[76] A. Ahmed, Y. Huang, and P. Mishra, "Cache reconfiguration using machine learning for vulnerability-aware energy optimization," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 18, no. 2, p. 15, 2019.

[77] A. Settle, D. Connors, E. Gibert, and A. González, "A dynamically reconfigurable cache for multithreaded processors," *Journal of Embedded Computing*, vol. 2, no. 2, pp. 221–233, 2006.

[78] W. Wang and P. Mishra, "Dynamic reconfiguration of two-level caches in soft real-time embedded systems," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI) 2009*. IEEE, 2009, pp. 145–150.

[79] P.-Y. Hsu and T. Hwang, "Thread-criticality aware dynamic cache reconfiguration in multi-core system," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2013*. IEEE, 2013, pp. 413–420.

[80] Y. Huang and P. Mishra, "Vulnerability-aware energy optimization using reconfigurable caches in multicore systems," in *IEEE International Conference on Computer Design (ICCD) 2017*. IEEE, 2017, pp. 241–248.

[81] W. Wang, P. Mishra, and A. Gordon-Ross, "Dynamic cache reconfiguration for soft real-time systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 11, no. 2, p. 28, 2012.

[82] W. Wang, P. Mishra, and S. Ranka, "Dynamic cache reconfiguration and partitioning for energy optimization in real-time multi-core systems," in *48th ACM/EDAC/IEEE Design Automation Conference (DAC), 2011*. IEEE, 2011, pp. 948–953.

[83] R. Reddy and P. Petrov, "Cache partitioning for energy-efficient and interference-free embedded multitasking," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 9, no. 3, p. 16, 2010.

[84] D. Kaseridis, J. Stuecheli, and L. K. John, "Bank-aware dynamic cache partitioning for multicore architectures," in *International Conference on Parallel Processing (ICPP) 2009*. IEEE, 2009, pp. 18–25.

[85] S. Charles, Y. Lyu, and P. Mishra, "Real-time detection and localization of dos attacks in noc based socs," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 1160–1165.

[86] C. Chen, J. L. Abellán, and A. Joshi, "Managing laser power in silicon-photonic noc through cache and noc reconfiguration," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 34, no. 6, pp. 972–985, 2015.

[87] H. Hajimiri, K. Rahmani, and P. Mishra, "Compression-aware dynamic cache reconfiguration for embedded systems," *Sustainable Computing: Informatics and Systems*, vol. 2, no. 2, pp. 71–80, 2012.

[88] 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 *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2018, pp. 345–350.

[89] M. K. J.Y.V., A. K. Swain, S. Kumar, S. R. Sahoo, and K. Mahapatra, "Run time mitigation of performance degradation hardware trojan attacks in network on chip," in *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2018, pp. 738–743.

[90] J. Sepúlveda, A. Zankl, D. Flórez, and G. Sigl, "Towards protected mpsoc communication for information protection against a malicious noc," *Procedia computer science*, vol. 108, pp. 1103–1112, 2017.

[91] T. Boraten and A. K. Kodi, "Packet security with path sensitization for nocs," in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2016, pp. 1136–1139.

[92] C. H. Gebotys and R. J. Gebotys, "A framework for security on noc technologies," in *Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI'03)*, ser. ISVLSI '03. USA: IEEE Computer Society, 2003, p. 113.

[93] "Using TinyCrypt Library, Intel Developer Zone, Intel, 2016." https://software.intel.com/en-us/node/734330, [Online].

[94] D. Goldschlag, M. Reed, and P. Syverson, "Onion routing," *Communications of the ACM*, vol. 42, no. 2, pp. 39–41, 1999.

[95] S. Charles, M. Logan, and P. Mishra, "Lightweight Anonymous Routing in NoC based SoCs," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2020.

[96] Q. Yu and J. Frey, "Exploiting error control approaches for hardware trojans on network-on-chip links," in *2013 IEEE international symposium on defect and fault tolerance in VLSI and nanotechnology systems (DFTS)*. IEEE, 2013, pp. 266–271.

[97] H. K. Kapoor, G. B. Rao, S. Arshi, and G. Trivedi, "A security framework for noc using authenticated encryption and session keys," *Circuits, Systems, and Signal Processing*, vol. 32, no. 6, pp. 2605–2622, 2013.

[98] K. Sajeesh and H. K. Kapoor, "An authenticated encryption based security framework for noc architectures," in *2011 International Symposium on Electronic System Design*. IEEE, 2011, pp. 134–139.

[99] W. C. Huffman and V. Pless, *Fundamentals of error-correcting codes*. Cambridge university press, 2010.

[100] 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*, 2020.

[101] L. Fiorin, G. Palermo, and C. Silvano, *A Security Monitoring Service for NoCs*. New York, NY, USA: Association for Computing Machinery, 2008, p. 197–202. [Online]. Available: https://doi.org/10.1145/1450135.1450180

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

[103] J. Frey and Q. Yu, "A hardened network-on-chip design using runtime hardware trojan mitigation methods," *Integr. VLSI J.*, vol. 56, no. C, p. 15–31, Jan. 2017. [Online]. Available: https://doi.org/10.1016/j.vlsi.2016.06.008

[104] T. Boraten, D. DiTomaso, and A. K. Kodi, "Secure model checkers for network-on-chip (noc) architectures," in *2016 International Great Lakes Symposium on VLSI (GLSVLSI)*, 2016, pp. 45–50.

[105] T. Boraten and A. K. Kodi, "Mitigation of denial of service attack with hardware Trojans in NoC architectures," in *IPDPS*, 2016, pp. 1091–1100.

[106] R. JS, 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 9th International Symposium on Networks-on-Chip*, ser. NOCS '15. New York, NY, USA: Association for Computing Machinery, 2015. [Online]. Available: https://doi.org/10.1145/2786572.2786580

[107] H. M. G. Wassel, Y. Gao, J. K. Oberg, T. Huffmire, R. Kastner, F. T. Chong, and T. Sherwood, "Surfnoc: A low latency and provably non-interfering approach to secure networks-on-chip," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ser. ISCA '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 583–594. [Online]. Available: https://doi.org/10.1145/2485922.2485972

[108] 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, no. 10, pp. 3241–3290, 2015.

[109] J. Sepúlveda, D. Aboul-Hassan, G. Sigl, B. Becker, and M. Sauer, "Towards the formal verification of security properties of a network-on-chip router," in *2018 IEEE 23rd European Test Symposium (ETS)*. IEEE, 2018, pp. 1–6.

[110] E. Levi, "Smashing the stack for fun and profit," *Phrack Magazine*, vol. 7, no. 49, 1996.

[111] L. Fiorin, G. Palermo, S. Lukovic, V. Catalano, and C. Silvano, "Secure memory accesses on networks-on-chip," *IEEE Transactions on Computers*, vol. 57, no. 9, pp. 1216–1229, 2008.

[112] A. Saeed, A. Ahmadinia, M. Just, and C. Bobda, "An id and address protection unit for noc based communication architectures," in *Proceedings of the 7th International Conference on Security of Information and Networks*. ACM, 2014, p. 288.

[113] J. Diguet, S. Evain, R. Vaslin, G. Gogniat, and E. Juin, "Noc-centric security of reconfigurable soc," in *First International Symposium on Networks-on-Chip (NOCS'07)*, 2007, pp. 223–232.

[114] S. Lukovic and N. Christianos, "Hierarchical multi-agent protection system for noc based mpsocs," in *Proceedings of the International Workshop on Security and Dependability for Resource Constrained Embedded Systems*. New York, NY, USA: Association for Computing Machinery, 2010. [Online]. Available: https://doi.org/10.1145/1868433.1868441

[115] T. Alves, "Trustzone: Integrated hardware and software security," *White paper*, 2004.

[116] J. Sepúlveda, D. Flórez, and G. Gogniat, "Reconfigurable security architecture for disrupted protection zones in NoC-based MPSoCs," in *2015 10th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*. IEEE, 2015, pp. 1–8.

[117] J. Sepúlveda, G. Gogniat, D. Florez, J.-P. Diguet, C. Zeferino, and M. Strum, "Elastic security zones for noc-based 3d-mpsocs," in *2014 21st IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. IEEE, 2014, pp. 506–509.

[118] J. Porquet, A. Greiner, and C. Schwarz, "Noc-mpu: A secure architecture for flexible co-hosting on shared memory mpsocs," in *2011 Design, Automation Test in Europe*, 2011, pp. 1–4.

[119] T. C. Bressoud and F. B. Schneider, "Hypervisor-based fault tolerance," *ACM Transactions on Computer Systems (TOCS)*, vol. 14, no. 1, pp. 80–107, 1996.

[120] 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 *2016 29th Symposium on Integrated Circuits and Systems Design (SBCCI)*, 2016, pp. 1–6.

[121] L. S. Indrusiak, J. Harbin, and M. J. Sepulveda, "Side-channel attack resilience through route randomisation in secure real-time networks-on-chip," in *2017 12th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, 2017, pp. 1–8.

[122] L. S. Indrusiak, J. Harbin, C. Reinbrecht, and J. Sepúlveda, "Side-channel protected mpsoc through secure real-time networks-on-chip," *Microprocessors and Microsystems*, vol. 68, pp. 34–46, 2019.

[123] C. Reinbrecht, A. Susin, L. Bossuet, and J. Sepúlveda, "Gossip noc – avoiding timing side-channel attacks through traffic management," in *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2016, pp. 601–606.

[124] Y. Wang and G. E. Suh, "Efficient timing channel protection for on-chip networks," in *2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip*, 2012, pp. 142–151.

[125] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 26, no. 1, pp. 96–99, 1983.

[126] P. C. Kocher, "Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems," in *Annual International Cryptology Conference*. Springer, 1996, pp. 104–113.

[127] A. Bogdanov, T. Eisenbarth, C. Paar, and M. Wienecke, "Differential cache-collision timing attacks on aes with applications to embedded cpus," in *Cryptographers' Track at the RSA Conference*. Springer, 2010, pp. 235–251.

[128] C. Reinbrecht, B. Forlin, A. Zankl, and J. Sepúlveda, "Earthquake—a noc-based optimized differential cache-collision attack for mpsocs," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 648–653.

[129] O. Choudary *et al.*, "Breaking smartcards using power analysis," *University of Cambridge*, 2005.

[130] P. Kocher, J. Jaffe, B. Jun, and P. Rohatgi, "Introduction to differential power analysis," *Journal of Cryptographic Engineering*, vol. 1, no. 1, pp. 5–27, 2011.

[131] S. Guo, J. Wang, Z. Chen, Z. Lu, J. Guo, and L. Yang, "Security-aware task mapping reducing thermal side channel leakage in cmps," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 10, pp. 5435–5443, 2019.

[132] M. J. Sepulveda, J.-P. Diguet, M. Strum, and G. Gogniat, "Noc-based protection for soc time-driven attacks," *IEEE Embedded Systems Letters*, vol. 7, no. 1, pp. 7–10, 2014.

[133] J. Sepúlveda, D. Flórez, M. Soeken, J.-P. Diguet, and G. Gogniat, "Dynamic noc buffer allocation for mpsoc timing side channel attack protection," in *2016 IEEE 7th Latin American Symposium on Circuits & Systems (LASCAS)*. IEEE, 2016, pp. 91–94.

[134] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar, "A 5-ghz mesh interconnect for a teraflops processor," *IEEE Micro*, vol. 27, no. 5, pp. 51–61, 2007.

[135] U. Y. Ogras and R. Marculescu, *Modeling, Analysis and Optimization of Network-on-Chip Communication Architectures*. Springer Publishing Company, Incorporated, 2013.

[136] "Intel Core i7-900 Processor," http://download.intel.com/design/processor/datashts/320834.pdf, [Online].

[137] "Intel Xeon Phi Processor 7210," http://ark.intel.com/products/94033/Intel-Xeon-Phi-Processor-7210-16GB-1_30-GHz-64-core, [Online].

[138] Y. Kim, D. Han, O. Mutlu, and M. Harchol-Balter, "Atlas: A scalable and high-performance scheduling algorithm for multiple memory controllers," in *HPCA-16 2010 The Sixteenth*

International Symposium on High-Performance Computer Architecture. IEEE, 2010, pp. 1–12.

[139] N. Agarwal, T. Krishna, L.-S. Peh, and N. K. Jha, "Garnet: A detailed on-chip network model inside a full-system simulator," in *2009 IEEE international symposium on performance analysis of systems and software*. IEEE, 2009, pp. 33–42.

[140] U. Y. Ogras, R. Marculescu, D. Marculescu, and E. G. Jung, "Design and management of voltage-frequency island partitioned networks-on-chip," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 3, pp. 330–341, 2009.

[141] 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 *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2009, pp. 469–480.

[142] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The splash-2 programs: Characterization and methodological considerations," *ACM SIGARCH computer architecture news*, vol. 23, no. 2, pp. 24–36, 1995.

[143] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization. WWC-4 (Cat. No. 01EX538)*. IEEE, 2001, pp. 3–14.

[144] D. Molka, D. Hackenberg, R. Schöne, and W. E. Nagel, "Cache coherence protocol and memory performance of the intel haswell-ep architecture," in *2015 44th International Conference on Parallel Processing*. IEEE, 2015, pp. 739–748.

[145] C. Zhang, F. Vahid, and R. Lysecky, "A self-tuning cache architecture for embedded systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 3, no. 2, pp. 407–425, 2004.

[146] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The splash-2 programs: Characterization and methodological considerations," in *ACM SIGARCH computer architecture news*, vol. 23, no. 2. ACM, 1995, pp. 24–36.

[147] N. Agarwal, T. Krishna, L.-S. Peh, and N. K. Jha, "Garnet: A detailed on-chip network model inside a full-system simulator," in *IEEE International Symposium on Performance Analysis of Systems and Software, 2009. ISPASS 2009.* IEEE, 2009, pp. 33–42.

[148] S. Bhat, "Energy models for network-on-chip components," *Master of Science, Department of Mathematics and Computer Science, Technische Universiteit Eindhoven, Eindhoven*, 2005.

[149] T. T. Ye, G. D. Micheli, and L. Benini, "Analysis of power consumption on switch fabrics in network routers," in *Proceedings of the 39th annual Design Automation Conference (DAC)*. ACM, 2002, pp. 524–529.

[150] C.-L. Su and A. M. Despain, "Cache designs for energy efficiency," in *Proceedings of the Twenty-Eighth Hawaii International Conference on System Sciences, 1995*, vol. 1. IEEE, 1995, pp. 306–315.

[151] 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 *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2009, pp. 469–480.

[152] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti *et al.*, "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, 2011.

[153] S. Charles, C. A. Patil, U. Y. Ogras, and P. Mishra, "Exploration of memory and cluster modes in directory-based many-core cmps," in *2018 Twelfth IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*. IEEE, 2018, pp. 1–8.

[154] S. Bell, B. Edwards, J. Amann, R. Conlin, K. Joyce, V. Leung, J. MacKay, M. Reif, L. Bao, J. Brown *et al.*, "Tile64-processor: A 64-core soc with mesh interconnect," in *2008 IEEE International Solid-State Circuits Conference-Digest of Technical Papers*. IEEE, 2008, pp. 88–598.

[155] B. D. de Dinechin, P. G. de Massas, G. Lager, C. Léger, B. Orgogozo, J. Reybert, and T. Strudel, "A distributed run-time environment for the kalray mppa®-256 integrated manycore processor," *Procedia Computer Science*, vol. 18, pp. 1654–1663, 2013.

[156] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *IEEE International Workshop on Workload Characterization, 2001. WWC-4. 2001*. IEEE, 2001, pp. 3–14.

[157] S. Manolache, P. Eles, and Z. Peng, "Task mapping and priority assignment for soft real-time applications under deadline miss ratio constraints," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 7, no. 2, p. 19, 2008.

[158] M. Bellare, O. Goldreich, and S. Goldwasser, "Incremental cryptography and application to virus protection," in *STOC*, vol. 95, 1995, pp. 45–56.

[159] S. Garg and O. Pandey, "Incremental program obfuscation," in *Annual International Cryptology Conference*. Springer, 2017, pp. 193–223.

[160] M. Bellare, O. Goldreich, and S. Goldwasser, "Incremental cryptography: The case of hashing and signing," in *Annual International Cryptology Conference*. Springer, 1994, pp. 216–233.

[161] D. Engels, M.-J. O. Saarinen, P. Schweitzer, and E. M. Smith, "The hummingbird-2 lightweight authenticated encryption algorithm," in *International Workshop on Radio Frequency Identification: Security and Privacy Issues*. Springer, 2011, pp. 19–31.

[162] D. A. McGrew, "Counter mode security: Analysis and recommendations," *Cisco Systems, November*, vol. 2, no. 4, 2002.

[163] B. Lebiednik, S. Abadal, H. Kwon, and T. Krishna, "Architecting a secure wireless network-on-chip," in *2018 Twelfth IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*. IEEE, 2018, pp. 1–8.

[164] J. Sepulveda, D. Flórez, V. Immler, G. Gogniat, and G. Sigl, "Efficient security zones implementation through hierarchical group key management at noc-based mpsocs," *Microprocessors and Microsystems*, vol. 50, pp. 164–174, 2017.

[165] I. Mironov, O. Pandey, O. Reingold, and G. Segev, "Incremental deterministic public-key encryption," in *Proceedings of the 31st Annual international conference on Theory and Applications of Cryptographic Techniques*. Springer-Verlag, 2012, pp. 628–644.

[166] M.-J. O. Saarinen, "Cryptanalysis of hummingbird-1," in *International Workshop on Fast Software Encryption*. Springer, 2011, pp. 328–341.

[167] K. Zhang, L. Ding, and J. Guan, "Cryptanalysis of hummingbird-2," Cryptology ePrint Archive, Report 2012/207, Tech. Rep., 2012.

[168] S. V. R. Chittamuru, I. G. Thakkar, V. Bhat, and S. Pasricha, "Soteria: Exploiting process variations to enhance hardware security with photonic noc architectures," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. IEEE, 2018, pp. 1–6.

[169] Y. Qin, D. Huang, and V. Kandiah, "Olar: On-demand lightweight anonymous routing in manets," in *Proc. Fourth Int'l Conf. Mobile Computing and Ubiquitous Networking (ICMU'08)*. Citeseer, 2008, pp. 72–79.

[170] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[171] J. Katz, A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, *Handbook of applied cryptography*. CRC press, 1996.

[172] Y. J. Yoon, N. Concer, M. Petracca, and L. P. Carloni, "Virtual channels and multiple physical networks: Two alternatives to improve noc performance," *IEEE Transactions on computer-aided design of integrated circuits and systems*, vol. 32, no. 12, pp. 1906–1919, 2013.

[173] P. Waszecki, P. Mundhenk, S. Steinhorst, M. Lukasiewycz, R. Karri, and S. Chakraborty, "Automotive electrical and electronic architecture security via distributed in-vehicle traffic monitoring," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 11, pp. 1790–1803, 2017.

[174] A. Sodani, R. Gramunt, J. Corbal, H.-S. Kim, K. Vinod, S. Chinthamani, S. Hutsell, R. Agarwal, and Y.-C. Liu, "Knights landing: Second-generation intel xeon phi product," *Ieee micro*, vol. 36, no. 2, pp. 34–46, 2016.

[175] L. Fiorin, C. Silvano, and M. Sami, "Security aspects in networks-on-chips: Overview and proposals for secure implementations," in *10th Euromicro Conference on Digital System Design Architectures, Methods and Tools (DSD 2007)*. IEEE, 2007, pp. 539–542.

[176] J. Niemela, "F-Secure Virus Descriptions," *F-Secure*, December 2007.

[177] S. Chakraborty, S. Künzli, and L. Thiele, "A General Framework for Analysing System Properties in Platform-Based Embedded System Designs," in *Date*, vol. 3. Citeseer, 2003, p. 10190.

[178] U. Suppiger, S. Perathoner, K. Lampka, and L. Thiele, "A simple approximation method for reducing the complexity of modular performance analysis," *Tech. Rep. 329*, 2010.

[179] E. Wandeler and L. Thiele, "Real-Time Calculus (RTC) Toolbox," http://www.mpa.ethz.ch/Rtctoolbox, 2006, [Online].

[180] J.-Y. Le Boudec and P. Thiran, *Network calculus: a theory of deterministic queuing systems for the internet*. Springer Science & Business Media, 2001, vol. 2050.

[181] K. Lampka, S. Perathoner, and L. Thiele, "Analytic real-time analysis and timed automata: a hybrid method for analyzing embedded real-time systems," in *Proceedings of the seventh ACM international conference on Embedded software*. ACM, 2009, pp. 107–116.

[182] K. Huang, G. Chen, C. Buckl, and A. Knoll, "Conforming the runtime inputs for hard real-time embedded systems," in *Proceedings of the 49th Annual Design Automation Conference*. ACM, 2012, pp. 430–436.

[183] M. Lukasiewycz, S. Steinhorst, and S. Chakraborty, "Priority assignment for event-triggered systems using mathematical programming," in *2013 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2013, pp. 982–987.

[184] S. Baruah *et al.*, "Scheduling periodic task systems to minimize output jitter," *RTCSA*, 1999.

[185] A. Monemi, J. W. Tang, M. Palesi, and M. N. Marsono, "ProNoC: A low latency network-on-chip based many-core system-on-chip prototyping platform," *Microprocessors and Microsystems*, vol. 54, pp. 60–74, 2017.

[186] M. Ramakrishna, V. K. Kodati, P. V. Gratz, and A. Sprintson, "GCA: Global congestion awareness for load balance in networks-on-chip," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 7, pp. 2022–2035, 2015.

[187] S. Murali, T. Theocharides, N. Vijaykrishnan, M. J. Irwin, L. Benini, and G. De Micheli, "Analysis of error recovery schemes for networks on chips," *IEEE Design & Test of Computers*, vol. 22, no. 5, pp. 434–442, 2005.

[188] T. Dierks, "The transport layer security (tls) protocol version 1.2," 2008.

[189] D. Engels, X. Fan, G. Gong, H. Hu, and E. M. Smith, "Ultra-lightweight cryptography for low-cost rfid tags: Hummingbird algorithm and protocol," *Centre for Applied Cryptographic Research (CACR) Technical Reports*, vol. 29, 2009.

[190] M. Li, Q.-A. Zeng, and W.-B. Jone, "Dyxy: a proximity congestion-aware deadlock-free dynamic routing method for network on chip," in *Proceedings of the 43rd annual Design Automation Conference*. ACM, 2006, pp. 849–852.

[191] (2008) Security on arm trustzone. [Online]. Available: https://www.arm.com/products/security-on-arm/trustzone

[192] I. F. Elashry, O. S. Faragallah, A. M. Abbas, S. El-Rabaie, and F. E. Abd El-Samie, "A new method for encrypting images with few details using Rijndael and RC6 block ciphers in the Electronic Code Book mode," *Information Security Journal: A Global Perspective*, vol. 21, no. 4, pp. 193–205, 2012.

[193] M. Bellare, J. Kilian, and P. Rogaway, "The security of cipher block chaining," in *Annual International Cryptology Conference*. Springer, 1994, pp. 341–358.

[194] R. S. Winternitz, "A secure one-way hash function built from DES," in *1984 IEEE Symposium on Security and Privacy*. IEEE, 1984, pp. 88–88.

[195] L. Thulasimani and M. Madheswaran, "Implementation of an energy efficient reconfigurable authentication unit for software radio," *International Journal on Computer Science and Engineering*, vol. 2, no. 04, pp. 1375–1380, 2010.

[196] J. Hesse, D. Hofheinz, and A. Rupp, "Reconfigurable cryptography: A flexible approach to long-term security," in *Theory of Cryptography Conference*. Springer, 2016, pp. 416–445.

[197] Z. Wang, Y. Yao, X. Tong, Q. Luo, and X. Chen, "Dynamically reconfigurable encryption and decryption system design for the internet of things information security," *Sensors*, vol. 19, no. 1, p. 143, 2019.

[198] D. McGrew and J. Viega, "The galois/counter mode of operation (gcm)," *Submission to NIST Modes of Operation Process*, vol. 20, 2004.

[199] H. Gilbert, M. J. Robshaw, and Y. Seurin, "How to encrypt with the LPN problem," in *International Colloquium on Automata, Languages, and Programming*. Springer, 2008, pp. 679–690.

[200] H. Gilbert and H. Handschuh, "Security analysis of sha-256 and sisters," in *International workshop on selected areas in cryptography*. Springer, 2003, pp. 175–193.

[201] R. Rivest, "The md5 message-digest algorithm," 1992.

[202] S. Chaudhury, K. T. Sistla, and S. Chattopadhyay, "Genetic algorithm-based fsm synthesis with area-power trade-offs," *Integration*, vol. 42, no. 3, pp. 376–384, 2009.

[203] C. Grecu, A. Ivanov, R. Saleh, E. S. Sogomonyan, and P. P. Pande, "On-line fault detection and location for noc interconnects," in *12th IEEE International On-Line Testing Symposium (IOLTS'06)*. IEEE, 2006, pp. 6–pp.

[204] W. Stallings, L. Brown, M. D. Bauer, and A. K. Bhattacharjee, *Computer security: principles and practice*. Pearson Education Upper Saddle River (NJ, 2012.

[205] H. Deng, X. Sun, B. Wang, and Y. Cao, "Selective forwarding attack detection using watermark in wsns," in *2009 ISECS International Colloquium on Computing, Communication, Control, and Management*, vol. 3. IEEE, 2009, pp. 109–113.

[206] X. Wang, D. S. Reeves, P. Ning, and F. Feng, "Robust network-based attack attribution through probabilistic watermarking of packet flows," North Carolina State University. Dept. of Computer Science, Tech. Rep., 2005.

[207] Z. Ling, X. Fu, W. Jia, W. Yu, D. Xuan, and J. Luo, "Novel packet size-based covert channel attacks against anonymizer," *IEEE Transactions on Computers*, vol. 62, no. 12, pp. 2411–2426, 2012.

[208] A. Houmansadr and N. Borisov, "Botmosaic: Collaborative network watermark for the detection of irc-based botnets," *Journal of Systems and Software*, vol. 86, no. 3, pp. 707–715, 2013.

[209] A. Houmansadr, N. Kiyavash, and N. Borisov, "Rainbow: A robust and invisible non-blind watermark for network flows." in *NDSS*, 2009.

[210] A. Zand, G. Vigna, R. Kemmerer, and C. Kruegel, "Rippler: Delay injection for service dependency detection," in *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*. IEEE, 2014, pp. 2157–2165.

[211] W. Hoeffding, "Probability inequalities for sums of bounded random variables," in *The Collected Works of Wassily Hoeffding*. Springer, 1994, pp. 409–426.

[212] R. M. Roth and G. Seroussi, "Bounds for binary codes with narrow distance distributions," *IEEE transactions on information theory*, vol. 53, no. 8, pp. 2760–2768, 2007.

[213] I. Dumer, D. Micciancio, and M. Sudan, "Hardness of approximating the minimum distance of a linear code," *IEEE Transactions on Information Theory*, vol. 49, no. 1, pp. 22–37, 2003.

[214] M. Best, A. Brouwer, F. MacWilliams, A. Odlyzko, and N. Sloane, "Bounds for binary codes of length less than 25," *IEEE Transactions on Information theory*, vol. 24, no. 1, pp. 81–93, 1978.

[215] T.-A. University, *Table of Nonlinear Binary Codes*, www.eng.tau.ac.il/~litsyn/tableand/index.html.

[216] A. Iacovazzi and Y. Elovici, "Network flow watermarking: A survey," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 1, pp. 512–530, 2016.

[217] A. Van Herrewege and I. Verbauwhede, "Software only, extremely compact, keccak-based secure prng on arm cortex-m," in *Proceedings of the 51st Annual Design Automation Conference (DAC)*. IEEE, 2014, pp. 1–6.

[218] A. May and I. Ozerov, "On computing nearest neighbors with applications to decoding of binary linear codes," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2015, pp. 203–228.

[219] K. Wang, A. Louri, A. Karanth, and R. Bunescu, "High-performance, energy-efficient, fault-tolerant network-on-chip design using reinforcement learning," in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2019, pp. 1166–1171.

[220] A. Garg and K. Tai, "Comparison of regression analysis, artificial neural network and genetic programming in handling the multicollinearity problem," in *2012 Proceedings of International Conference on Modelling, Identification and Control*, 2012, pp. 353–358.

[221] F. Dietrich and C. List, "Probabilistic opinion pooling," 2016.

[222] C. Ciordas, T. Basten, A. Radulescu, K. Goossens, and J. Meerbergen, "An event-based network-on-chip monitoring service," in *Proceedings. Ninth IEEE International High-Level Design Validation and Test Workshop (IEEE Cat. No.04EX940)*, 2004, pp. 149–154.

[223] S. R. Patil, G. B. Pularikkal, D. McGrew, B. H. Anderson, and M. Nanjanagud, "Encrypted traffic analytics over a multi-path tcp connection," Aug. 8 2019, uS Patent App. 15/891,708.

[224] V. Costan and S. Devadas, "Intel sgx explained." *IACR Cryptology ePrint Archive*, vol. 2016, no. 086, pp. 1–118, 2016.

BIOGRAPHICAL SKETCH

Subodha Charles received his Ph.D. degree in the Department of Computer and Information Science and Engineering, University of Florida, Gainesville, USA in 2020. He received his B.Sc. degree from the Department of Electronics and Telecommunications Engineering, University of Moratuwa, Sri Lanka in 2015. His research interests include secure SoC design, energy-aware computing and NoC architectures. He has published one book, ten book chapters, three journal articles and six conference papers. He has served as a reviewer of several premier international conferences and journals.