A Survey of Network-on-Chip Security Attacks and Countermeasures

SUBODHA CHARLES and PRABHAT MISHRA, University of Florida, USA

With the advances of chip manufacturing technologies, computer architects have been able to integrate an increasing number of processors and other heterogeneous components on the same chip. Network-on-Chip (NoC) is widely employed by multicore System-on-Chip (SoC) architectures to cater to their communication requirements. NoC has received significant attention from both attackers and defenders. The increased usage of NoC and its distributed nature across the chip has made it a focal point of potential security attacks. Due to its prime location in the SoC coupled with connectivity with various components, NoC can be effectively utilized to implement security countermeasures to protect the SoC from potential attacks. There is a wide variety of existing literature on NoC security attacks and countermeasures. In this article, we provide a comprehensive survey of security vulnerabilities in NoC-based SoC architectures and discuss relevant countermeasures.

CCS Concepts: • **Computer systems organization → System on a chip**; *Multicore architectures*; *Reconfigurable computing*; • **Hardware → Network on chip**;

Additional Key Words and Phrases: Hardware security, machine learning

ACM Reference format:
Subodha Charles and Prabhat Mishra. 2021. A Survey of Network-on-Chip Security Attacks and Countermeasures. *ACM Comput. Surv.* 54, 5, Article 101 (May 2021), 36 pages.
https://doi.org/10.1145/3450964

1 INTRODUCTION

We are living in the regime of **Internet-of-Things (IoT)**, a regime in which the number of connected smart computing devices exceeds the human population. Various reports suggest that we can expect over 50B devices to be deployed and mutually connected by 2025 [58], compared to about 500M in 2003 [43]. 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**

This work was partially supported by the National Science Foundation (NSF) grant SaTC-1936040.
Authors' addresses: S. Charles and P. Mishra, University of Florida, 432 Newell Drive, Gainesville, FL 32611-6120, USA; emails: {subodha96, prabhat}@ufl.edu.

0360-0300/2021/05-ART101 $15.00
https://doi.org/10.1145/3450964

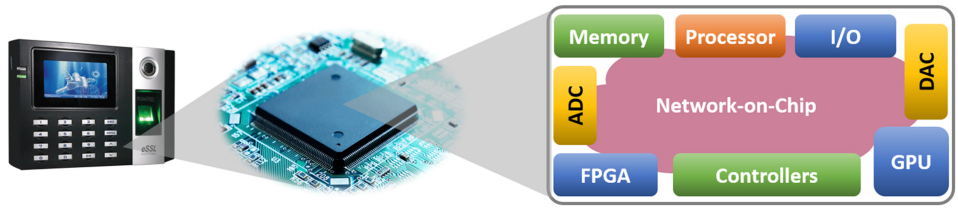


Fig. 1. An example System-on-Chip (SoC) with Network-on-Chip (NoC)-based communication fabric to interact with a wide variety of third-party Intellectual Property (IP) cores.

(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 [8]. Indeed, one of the most recent many-core processor architectures—Intel **"Knights Landing" (KNL)**—features 64–72 Atom cores and 144 vector processing units [109]. 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 [97], 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 **Network-on-Chip (NoC)**. As shown in Figure 1, a typical SoC utilizes NoC to communicate between multiple IP cores including processor, memory, controllers, converters, input/output devices, peripherals, and so on. 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 [59]. Therefore, it is evident that the NoC has become an increasingly important component in the SoC.

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 [83]. 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% [39, 80].

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 weapons control systems and nuclear power plants [108], which can allow these chips to be compromised remotely. The well-publicized *Spectre* [68] and *Meltdown* [75] attacks highlight how sensitive data can 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.

The ubiquity of devices using NoC-based SoCs has made NoC a focal point for security attacks as well as countermeasures. Therefore, 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. This article provides a comprehensive survey of NoC security on general purpose as well as application-specific systems. We outline potential security threats and effective countermeasures and discuss their strengths, weaknesses, and scope for improvement. Existing surveys related to NoC have focused on NoC architectures [2, 16] and energy optimization aspects [1]. Surveys on emerging NoC technologies such as Optical [119], Wireless [115], and 3D [125] NoC have been published as well. While there is an old review on NoC security (published in 2007 [48]), a vast majority of NoC security research has been performed in the past decade. To the best of our knowledge, there is no comprehensive survey of state-of-the-art NoC security attacks and countermeasures.

This article is organized as follows: Section 2 and Section 3 provides an overview of NoC architectures and its security landspace, respectively. Section 4 introduces the methodology of this survey. Specifically, it identifies five widely studied categories of NoC security attacks: (i) eavesdropping, (ii) spoofing and data integrity, (iii) denial-of-service, (iv) buffer overflow and memory extraction, and (v) side-channel attacks. Sections 5–9 describe these attacks and corresponding countermeasures in NoC-based SoCs. Section 10 summarizes the survey and outlines future research directions.

2 OVERVIEW OF NETWORK-ON-CHIP 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, and so on. 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 mail and other inventions might come into picture in the future as well. Analogous to this, the designer of a 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 signal. When the number of cores in a 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 [5] and IBM's CoreConnect [82] are two popular examples. 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.

The inspiration came from traditional networking solutions, more specifically, the Internet. The Network-on-chip, a miniature version of the wide area network with routers, packets and links was proposed as the solution [11, 38]. The new paradigm described a way of communicating between IPs, including features such as routing protocols, flow control, switching, arbitration, and

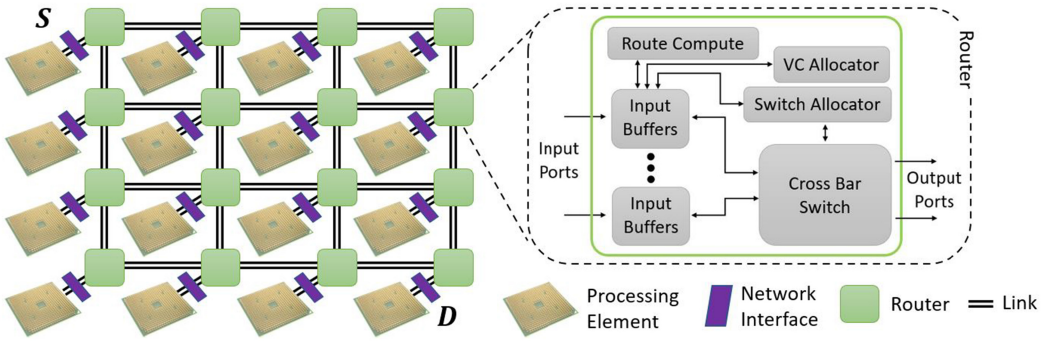


Fig. 2. Example of a NoC connecting 16 IPs.

buffering. With increased scalability, resource reuse, performance, and reduced costs, design risk, time-to-market, and so on, 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.

2.1 Network-on-chip Architecture and Communication Protocol

Figure 2 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. Without loss of generality, in this survey, 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 [109, 118].

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 on the network. A flit is the smallest unit of flow control in a NoC. Packets can 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 3 shows an overview of this process. Figure 4 shows the format of a memory request packet and a response data packet used in the gem5 architectural simulator [14].

Previous works have proposed several NoC architectures, such as Nostrum [71], SOCBUS [120], Proteo [107], Xpipes [37], Æthereal [52], and so on, based on different requirements. The choice of the parameters in the architecture depends on the design requirements, such as

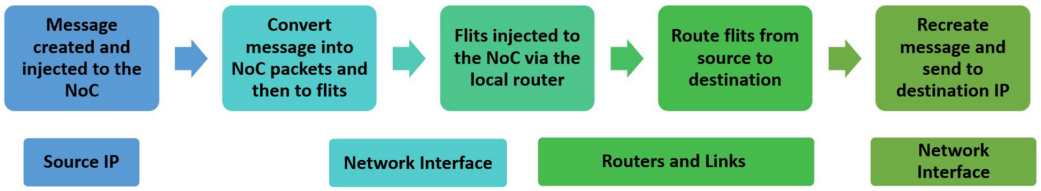
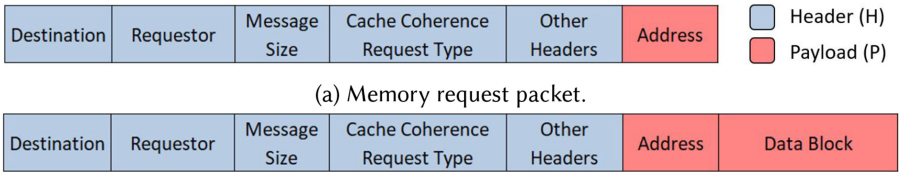


Fig. 3. Overview of a NoC traversal.



(a) Memory request packet.

(b) Response data packet from memory.

Fig. 4. NoC control (memory request packet) and data (response data packet) packet formats used in the gem5 simulator.

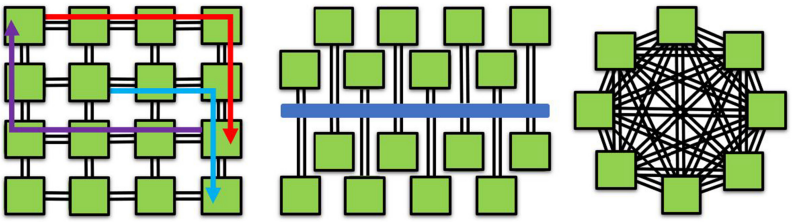


Fig. 5. NoC topologies and an example of X-Y routing in a Mesh NoC.

performance/power/area budgets, reliability, quality-of-service guarantees, scalability, and implementation cost. Some of the existing NoC architectures have been surveyed in literature [2, 16]. NoC architecture design needs to consider two important factors—network topology and routing protocol. The next two subsections describe these aspects in detail.

2.1.1 Topology. The topology defines the physical organization of IPs, routers and links of an interconnect. The organization in Figure 2 shows a Mesh topology. Crossbar, Point-to-point, Tree, 3-D Mesh are a few other commonly used topologies. Figure 5 shows some examples of them. The topology is chosen depending on the cost and performance requirements of a 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 tradeoff 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 overheads. The 2-D Mesh is the most common topology in NoC designs [109, 118]. 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.

2.1.2 Router and Routing Protocol. The routers comprise input buffers that accept packets from the local IP via the NI or from other routers connected to it (in the Mesh topology, except for the

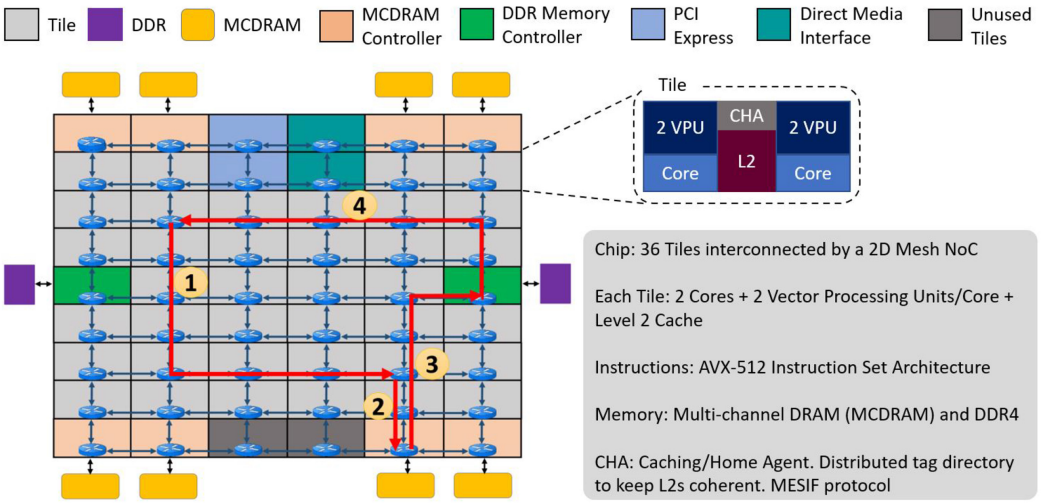


Fig. 6. Overview of the KNL architecture together with a NoC traversal for a memory request when Cache memory mode and All-to-all cluster mode are used: (1) L2 cache miss. Request sent to check the tag directory, (2) request forwarded to MCDRAM, which acts as the last-level cache after miss in tag directory, (3) request forwarded to memory after miss in MCDRAM, (4) data read from memory and sent to the requester [109].

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 [34].

The routing protocol defines the path a flit should take in a given topology. Routing protocols are broadly classified as (i) deterministic routing protocols and (ii) adaptive routing protocols. In deterministic routing, each packet traversing from S to D follows the same path. X-Y routing is one common example for deterministic routing. In X-Y routing, packets use X-directional links first, before using Y-directional links [40]. An example including three paths taken by X-Y routing in a Mesh NoC is shown in Figure 5. Adaptive routing takes network states such as congestion, security, reliability into account and takes the flits through different paths based on the current state of the network [127].

2.2 An Example of a Commercial NoC-based SoC

There are many examples of commercial NoC-based SoCs. As an illustrative example, we describe how NoC is utilized in the Intel KNL architecture [109]. Figure 6 shows an overview of the KNL architecture together with a NoC traversal for a memory request. The architecture implements a directory-based cache coherence protocol and supports two types of memory: (i) high-bandwidth, low-capacity **multi-channel DRAM (MCDRAM)**, and (ii) low-bandwidth, high-capacity **double data rate (DDR)** memory. These two memory types can be configured in three configurations (Cache mode, Flat mode, and Hybrid mode) known as *memory modes*. Furthermore, the affinity between cores, directories, and memory controllers can be configured in three modes (All-to-all, Quadrant, and Sub-NUMA), which are known as *cluster modes*. Each combination of memory and cluster modes are intended at optimizing performance based on application requirements and they cause different traffic patterns in the NoC [24].

The communication between tiles is facilitated by the Intel mesh on-die interconnect architecture that is widely deployed in the Intel Xeon processor family. The interconnect consists of four

parallel networks, each delivering packets belonging to different types, and each network enforces the Y-X routing protocol. The example in Figure 6 shows the NoC traversal of a memory request when the Cache memory mode is used with the All-to-all cluster mode. After the L2 miss, a request is injected on the NoC in the form of packets to check the tag directory according to the cache coherence protocol. Upon the directory miss, the request is sent to the next element in the memory hierarchy—the MCDRAM. Since the Cache mode is used, the MCDRAM acts as a last-level cache. The cache miss at the MCDRAM causes a DDR memory access through the DDR memory controllers. When the requested data is retrieved from memory, it is sent back to the original requester. Therefore, it is evident that the NoC performance is critical to the overall performance of the SoC. The interconnect in KNL is capable of delivering more than 700 GBps of total aggregate bandwidth.

2.3 Emerging NoC Technologies

When NoC was first introduced, the discussion 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 has increased significantly. As a result, the communication between SoC components has become the bottleneck. Irrespective of the architectural optimizations, electrical NoC exhibits inherent limitations due to the physical characteristics of electrical wires [86].

- 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 [112, 113].
- Electrical NoC can contribute a significant portion of the on-chip capacitance. In some cases, about 70% of the total capacitance [84].
- The above two reasons combined make the Electrical NoC a major source of power dissipation.

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 [30, 111]. 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 [9].

Recent years have seen the introduction of emerging NoC technologies such as *wireless NoC* [41] and *optical NoC* [85]. While the focus of this article is to survey 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 NoCs always use one-hop routing, while optical and electrical NoCs utilize one-hop or muti-hop communication, depending on the source and destination. Figure 7 shows an overview of how different NoC technologies can be used to connect heterogeneous SoC components. In the rest of the survey, we use the word NoC to refer to electrical NoC unless otherwise specified as Wireless NoC or Optical NoC.

2.3.1 Wireless NoC. Wireless NoC was proposed as a solution to the latency experienced in electrical NoCs, which are based on metal interconnects and multi-hop communication. Wireless NoCs integrate on-chip antennas and suitable transceivers that enable communication between

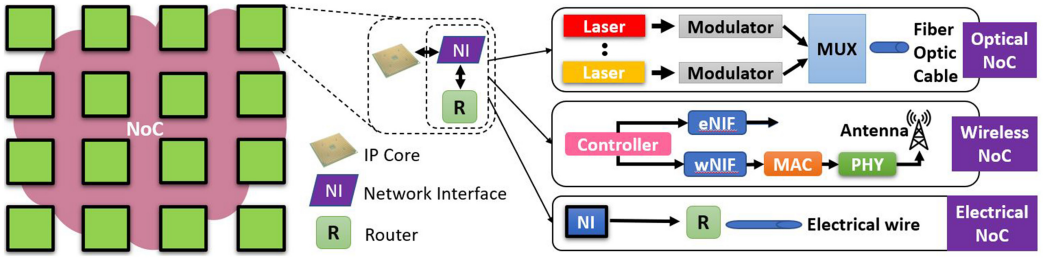


Fig. 7. NoC enables communication between IPs. The network interface (NI), router (R), and links can be implemented using optical, wireless, or electrical communication technologies.

two IPs without a wired medium. Silicon integrated antennas communicating using the millimeter wave range are shown to be a viable technology for on-chip communication [41].

2.3.2 Optical NoC. However, optical NoC, also known as photonic NoC, uses photo emitters, optical wave guides and transceivers for communication [126]. 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.

3 SECURITY LANDSCAPE IN NoC-BASED SYSTEM-ON-CHIP

There is a growing interest in the industry to use the NoC to secure the SoC as evident from NoC-Lock [110] and FlexNoC resilience package [7]. However, 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 two reasons: (i) NoC has access to all system data, and (ii) NoC spans across the entire SoC and 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 3.1) and why securing NoC-based SoCs has become a hard problem (Section 3.2).

3.1 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 [45]. An adversary can utilize the malicious implants (hardware Trojans) to cause malfunction or facilitate information leakage [83]. An adversary can also exploit legitimate test and debug interfaces as a backdoor for information leakage [108]. Many security vulnerabilities can be created unintentionally by design automation/**computer-aided design (CAD)** tools or by designers' mistakes [121]. These vulnerabilities can lead to untrusted (potentially malicious) IPs.

Attacks based on malicious implants, such as hardware Trojans, rely on Trojans being integrated to the SoC without being detected at the post-silicon verification stage or during runtime [83]. 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 [12]. Even if all the IPs are tested before integrating, hardware Trojans can still go undetected because of the complexity of designs with billions of transistors that make physical inspection or 100% coverage in design

verification/validation a costly or an impossible process [114]. 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 [13]. A smart attacker can carefully craft the Trojan activation method so it becomes difficult to detect. Previous work has discussed external/internal Trojan activation modes [114], software-hardware coalition [4], and triggers based on time, input sequence, traffic pattern, and even thermal conditions [13].

The usage of third-party NoC IPs has grown rapidly over the years. Due to the widespread use of NoC IPs as discussed in Section 1, outsourcing NoC IP fabrication has become a common practice. iSuppli—an independent market research firm, has concluded from their research that the FlexNoC on-chip interconnection architecture [7] is used by four out of the top five Chinese fabless semiconductor OEM (original equipment manufacturer) companies [105]. This has led to Arteris, the company that developed FlexNoC, achieving a sales growth of 1002% over a three-year time period through IP licensing [106]. 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 [91].

3.2 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 [23, 67, 124]. 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 poses additional unique challenges, as outlined below.

3.2.1 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.

3.2.2 Increased Complexity. The complexity of SoC designs has made exhaustive security validation an impossible task. Most IPs come as black boxes from vendors that do not reveal design details 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 get triggered only by a specific set of inputs, as discussed in Section 3.1. Therefore, using security validation tools during design time to capture all security vulnerabilities is not feasible.

3.2.3 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 7. 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

Fig. 8. Five classes of security attacks discussed in this survey.

wireless NoCs, optical and electrical NoCs need to consider a wide variety of network topologies as well.

4 SURVEY METHODOLOGY

There is a wide variety of security vulnerabilities in NoC-based SoCs. In this survey, we focus on some of the most commonly discussed vulnerabilities in the context of on-chip communication architecture, including information leakage, denial-of-service, as well as data corruption. Specifically, we analyze the following five types of security attacks and corresponding countermeasures in the existing literature:

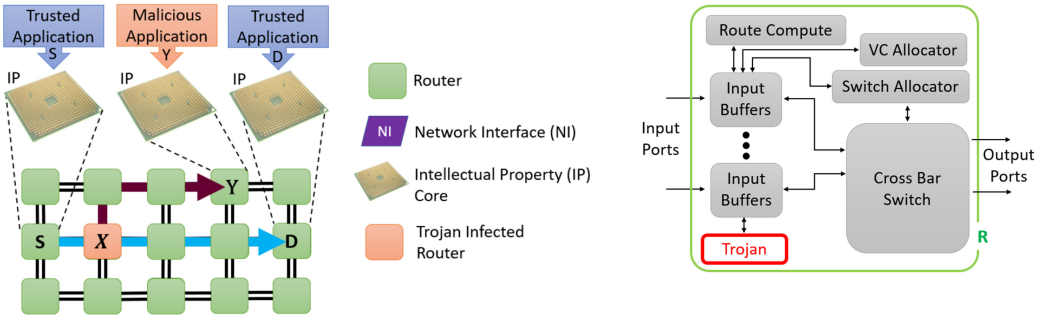
(1) Eavesdropping: Listening to confidential information (secure NoC packets) passed through the NoC when other IPs are communicating.
(2) Spoofing/Data integrity: Intentional data corruption to cause erroneous behavior and/or system failure, whereas Spoofing is an Impersonation type of attack that disguises communication from an untrusted IP as a trusted IP.
(3) Denial-of-service: Attacks that overwhelm the network resources in an attempt to cause performance degradation, real-time guarantee violations, and reduction of battery lifetime.
(4) Buffer overflow/Memory extraction: Gain access to unauthorized privilege levels and memory locations.
(5) Side Channel: Mount attacks by utilizing non-functional properties such as time, power, and electromagnetic radiation.

Figure 8 shows the five classes of attacks we consider in this article. The subsequent sections discuss each attack class in detail. Each section introduces the attack and outlines threat model(s) and defense(s) in NoC context. Related defenses are compared and contrasted in terms of pros and cons. This survey mainly includes manuscripts published in the past 20 years. Papers written related to NoC that do not have a security focus such as NoC architecture optimization and NoC verification/validation are not covered in this survey.

To provide defense against most of these attacks, an efficient way to generate and manage shared secrets (keys and cryptographic parameters) is required. Previous studies have addressed this challenge in several ways. One such example is the key management system proposed by Lebiednik et al. [73]. 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 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.

5 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



(a) A malicious router (X) copies packets passing through it and sends to a malicious application running on an IP (Y).

(b) Router infected with a hardware Trojan that can facilitate the attack.

Fig. 9. Illustrative example of an eavesdropping attack through colluding hardware and software.

of hardware security breaches where hard-to-detect hardware components, which were not a part of the original design, integrated into the original design leaking information have attracted more attention to eavesdropping attacks [17].

As discussed in Section 2, 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. In this section, we first discuss a few related threat models and introduce corresponding countermeasures.

5.1 Threat Models for Eavesdropping Attacks

In this section, we classify eavesdropping attacks on NoC-based SoCs into different categories based on the nature of the threat model.

5.1.1 Malicious Router and Colluding Application. 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. Figure 9(a) shows an illustrative example. Two trusted applications running in nodes S and D are communicating with each other on a path that goes through node X. A hardware Trojan is inserted into the NoC IP during design time, and as a result, the router at X is infected. The infected router copies packets passing through it and sends them to the IP running the malicious application at node Y. As a result, the malicious application at Y gets to eavesdrop on the communication between S and D.

Figure 9(b) shows a block diagram of a router infected with the hardware Trojan that can facilitate the attack. The Trojan can act based on the commands sent by the malicious application. Once the SoC is in operation, the malicious application sends commands to the hardware Trojan at a desired time. Then the Trojan initiates the attack. When packets are received at the input buffer, the Trojan makes a copy of it, modifies the header information so the destination is IP Y instead of D, and places it back in the input buffer. The NoC then routes the duplicated packets to Y and the malicious application receives it. The malicious application can also send commands to pause the attack to minimize the risk of being detected. This threat model has been extensively used to study eavesdropping attacks especially, since the attack is hard to detect [4, 21, 27, 31, 65].

The attack is hard to detect because the Trojan has a very small area and power footprint. The same threat model used by Ancajas et al. [4] reported 4.62% and 0.28% area and power overheads, respectively, when compared with the router design without the Trojan. The copied packets can introduce performance overhead due to NoC congestion, but it is shown to be less than 1% [4]. Therefore, the likelihood of the Trojan being detected is very small unless additional security mechanisms are implemented.

Sepúlveda et al. [103] introduced a similar threat model for eavesdropping. Compared to the accompanying malicious application sending commands to initiate or pause the attack, their work discussed malicious control logic integrated within the router controlling the attack.

5.1.2 Malicious NI and Colluding Application. 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 [91]. In the threat model presented in Reference [91], 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 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% [91].

The authors argue that the 4.62% area overhead [4] in the router-application combination can be noticed by testers during verification. Furthermore, the use of buffers and virtual channels in the router when duplicating packets increases the likelihood of being detected using secure model checkers as well. Therefore, the NI-application collusion is considered to be a stronger threat compared to router-application collusion.

5.1.3 Eavesdropping through External I/O Pins. 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 [50]. However, NoCs are generally more resistant against bus-probing attacks compared to the traditional bus-based architectures.

5.2 Defenses against Eavesdropping Attacks

In this section, we classify defenses for eavesdropping attacks on NoC-based SoCs into different categories based on the nature of the defense.

5.2.1 Authentication/Encryption. Previous work that addressed eavesdropping attacks in NoC proposed authenticated encryption as a solution [29, 50, 103]. An overview of how an authenticated encryption scheme can be integrated in the NoC is shown in Figure 10. The block diagram closely resembles the **Galois Counter Mode (GCM)**-based encryption and authentication [81]. Packets originating from each IP are encrypted (ciphertext denoted by C) and an authentication tag (T) is appended to each packet at the NI before injecting it to the NoC. The entire packet, which consists of $H \parallel C \parallel T$ traverses the NoC and arrives at the destination. The header H is sent as plaintext so intermediate routers can use the header information, such as source and destination addresses, for routing. At the destination NI, the inverse process takes place. The tag T is validated, and if valid, the ciphertext C is decrypted to send the plaintext to the desired IP. Encryption ensures that the plaintext of the secure information is not leaked, and authentication detects any tampering with the packet including header information.

Sepúlveda et al. [103] proposed a variation of authenticated encryption where only the destination is sent as plaintext and the source and data is encrypted using AES Counter mode [36]. The hash of the entire packet is calculated using SipHash [10] to ensure data integrity. The

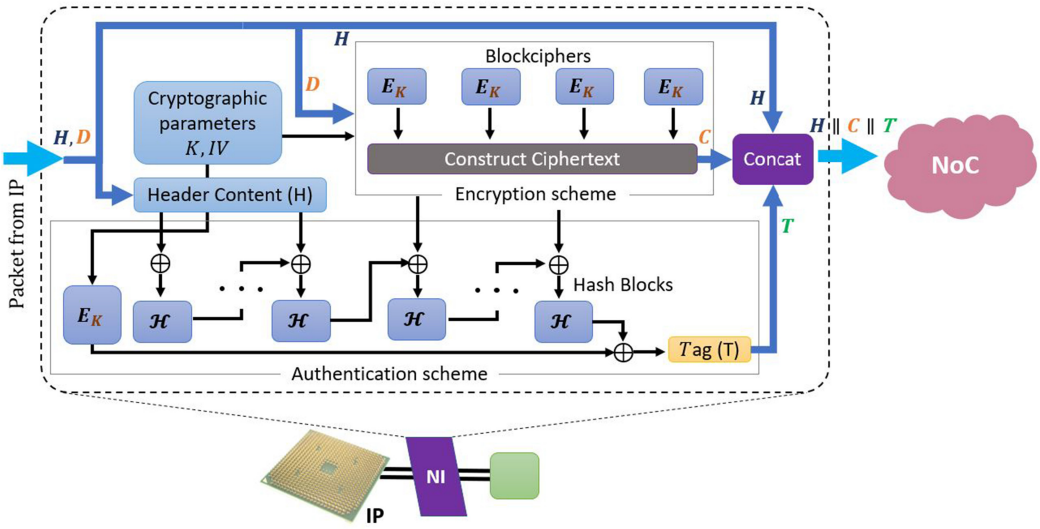


Fig. 10. Overview of an authenticated encryption scheme implemented to provide security to NoC.

authors introduce their solution as a tunnel-based communication mechanism to isolate sensitive information transferred between IPs. Tunnels are created by encapsulating packets in a way such that the communication is isolated by the compromised NoC. AES Counter mode provides high parallelizability at the expense of power and area. SipHash is chosen as the hash function, since it provides a fast and lightweight message authentication code. SipHash is specially suited to provide a secure and fast MAC function for short inputs, which is an ideal fit for NoC packets. AES Counter mode and SipHash also provide reconfigurability, in terms of number of AES blocks and SipHash rounds, depending on the performance requirement of the SoC.

Several prior studies have tried to develop lightweight encryption and authentication schemes for on-chip data communication. Ancajas et al. [4] 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 Reference [21]. Intel's TinyCrypt—a cryptographic library with a small footprint—is built for constrained IoT devices [62]. It provides basic functionality to build a secure system with very little overhead. It gives SHA-256 hash functions, message authentication, a pseudo-random number generator that 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.

5.2.2 Additional Validation Checks. The duplicated packets in router-application combination as well as NI-application combination can be detected by additional validation checks. In Reference [91], the authors implemented a **snooping invalidator module (SIM)** at the NI output queue to discard duplicate packets. To discard duplicates, the uniqueness of the flits was validated using an additional key sent by the IP to NI. SIM is shown to have low power and area overhead. The same methodology can be applied to the packets duplicated at the router with small modifications. In addition to the SIM, an eavesdropping attack detector is implemented at each IP. The attack detector works on the principle that during an attack, the ratio of incoming and outgoing packets at an IP is significantly greater than one. The detector observes the ratio over a long period of

time (few hours) and identifies ongoing eavesdropping attacks. As a result of the combination of SIM and detector, not only the packet duplication is prevented at the NI, but also the malicious application is detected so future attacks can be prevented.

5.2.3 Information Obfuscation. 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. Even if the data in a packet are encrypted, typically, source and destination addresses are sent as plaintext for faster routing. Therefore, eavesdroppers can easily extract this information. Onion routing, a well-known mechanism in the computer networks domain, can hide the origin and target of a network packet [51]. 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 [4, 27]. Ancajas et al. [4] proposed to migrate applications periodically to another node in the SoC. The SoC firmware maintains the relevant information and initiates a seamless migration periodically. As a result, the source and destination addresses are decoupled, making it harder for a malicious agent to launch the attack.

An anonymous routing scheme that implements virtual-channel-based communication was proposed in Reference [27]. When two IPs want to communicate with each other, a three-way handshake is initiated, during which the routers along the routing path share virtual channel numbers anonymously. The anonymity during the handshake is maintained by encrypting packets using a combination of symmetric and asymmetric encryption. During data transfer, packets are routed using the virtual channel numbers of the preceding and following routers. The source and destination addresses are not revealed at any point. The performance overhead incurred during the three-way handshake gets amortized during operation, since once the routing path is established, packets are routed using a simple table look-up with virtual channel numbers.

5.3 Overhead and Effectiveness of Defenses against Eavesdropping Attacks

As outlined in Section 5.2, securing NoC packets against eavesdropping attacks mainly involve a "design-for-security" approach. The extra hardware required to secure the packets introduce additional power, performance, and area overhead. The goal is to develop an effective solution while minimizing the overhead. The effectiveness of a defense against eavesdropping attacks can be evaluated by the security guarantees it provides. For example, an encryption scheme implemented to secure packets must be hard to break. We define a three-step scale (low, average, high) based on the security guarantees provided by each countermeasure. A similar three-step scale (low, average, high) is defined to indicate the hardware overhead introduced by the added security compared to the default (without security) architecture. It is important to note that the overhead of each countermeasure is evaluated relative to the overhead of other countermeasures against eavesdropping attacks in NoC-based SoCs. Table 1 shows a summary of the papers we have surveyed related to eavesdropping attacks. The second and third columns outline the threat models and corresponding defenses, respectively. The last two columns provide the overhead as well as effectiveness of each countermeasure against eavesdropping attacks, respectively. For example, the AES counter mode encryption proposed in Reference [103] is highly effective but introduces high overhead. An ideal countermeasure should be high in effectiveness and low in hardware overhead.

6 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, then it can lead to erroneous execution of programs as well as system failures [27]. However, spoofing is the act of disguising a communication

Table 1. Threat Model, Defense, Overhead, and Effectiveness of Work in Existing Literature Related to Eavesdropping Attacks in NoC-based SoCs

	Threat Model	Defense	Overhead	Effectiveness
Gebotys, 2003 [50]	ETEP	ANEN	Average	Average
Sajeesh, 2011 [98]	MRCA*	ANEN	High	High
Kapoor, 2013 [66]	MRCA*	ANEN	High	High
Ancajas, 2014 [4]	MRCA	ANEN, INON	Low	Low
Boraten, 2016 [21]	MRCA	ANEN	Average	Average
Sepúlveda, 2017 [103]	MRCA	ANEN	High	High
Kumar, 2018 [65]	MRCA	INON	Low	Low
Hussain, 2018 [57]	MRCA	ANEN	Average	Average
Raparti, 2019 [91]	MNCA	AVCS	Low	Average
Charles, 2020 [27]	MRCA	ANEN, INON	Average	High

Threat model: Malicious Router and Colluding Application (MRCA), Malicious NI and Colluding Application (MNCA), and Eavesdropping Through External I/O Pins (ETEP). **Defense**: Authentication/Encryption (ANEN), Additional Validation Checks (AVCS), and Information Obfuscation (INON). **Overhead**: A three-step scale (low, average, high) compared to the overhead of the default (without security) architecture. **Effectiveness:** A three-step scale (low, average, high) based on the security guarantees against eavesdropping attacks. "*" indicates that this is our best estimate, since the authors did not provide the exact information.

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 into 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. While data corruption and spoofing might appear disjoint, the threat models and defenses for both types of attacks are similar, and therefore, we classify them in the same attack class. Existing literature on NoC security has explored several variations of the data integrity/spoofing attacks. In this section, we first discuss a few related threat models and introduce corresponding countermeasures.

6.1 Threat Models for Spoofing and Data Integrity Attacks

In this section, we classify spoofing and data integrity attacks on NoC-based SoCs into different categories based on the nature of the threat model.

6.1.1 Packet Corruption at Routers. Sepúlveda et al. presented *MalNoC*, a Trojan-infected NoC that can perform multiple attacks on NoC packets [103]. Out of the capabilities of MalNoC, the attack model for eavesdropping was discussed in Section 5.1. MalNoC launches data integrity attacks in a similar manner where the infected 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. Depending on the control instructions, the router can either eavesdrop or corrupt packets. A similar threat model that discussed eavesdropping, DoS, and illegal packet forwarding, all of which utilized packet corruption at a router, was presented in Reference [57]. Kumar et al. [65] discussed a Trojan that corrupts flits arriving at the input buffers of a router. However, their main focus was to cause DoS attacks, and therefore, we have discussed their threat model and countermeasures in Section 7.

6.1.2 Packet Corruption at Links. 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 [122]. 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 [122].

6.2 Defenses against Spoofing and Data Integrity Attacks

In this section, we classify defenses for spoofing and data integrity attacks on NoC-based SoCs into different categories based on the nature of the defense.

6.2.1 Authentication/Encryption. As discussed in Section 5.2.1, authenticated encryption schemes provide data confidentiality through encryption and data integrity through authentication [66, 98, 103]. While the focus of Section 5.2.1 was on preventing eavesdropping attacks using the encryption portion, this section is focused on checking data integrity using the authentication scheme. The underlying principal remains the same, as shown in Figure 10. 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. [57] 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 [57].

6.2.2 Error Correcting Codes. **Error correcting codes (ECC)** are widely used in the telecommunications domain [56]. 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 Reference [122]. 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, then 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.

6.3 Overhead and Effectiveness of Defenses against Spoofing/Data Integrity Attacks

Similar to eavesdropping attacks, securing NoC packets against spoofing and data integrity attacks also involves a "design-for-security" approach. Additional validation and error correction is used to defend against the attacks, and the extra hardware required to implement the defenses introduces additional power, performance, and area overhead. The effectiveness of a defense against eavesdropping and data integrity attacks can be evaluated by its ability to detect corrupted/spoofed packets and/or how difficult it is to launch an attack when the defense is present. Table 2 shows a summary of the papers we have surveyed related to spoofing and data integrity attacks. The second and third columns outline the threat models and corresponding defenses, respectively. The last two columns provide the overhead as well as effectiveness of each countermeasure against spoofing and data integrity attacks. We use a three-step scale (low, average, high) to quantify both overhead and effectiveness of these countermeasures.

Table 2. Threat Model, Defense, Overhead, and Effectiveness of Work in Existing Literature Related to Spoofing and Data Integrity Attacks in NoC-based SoCs

	Threat Model	Defense	Overhead	Effectiveness
Sajeesh, 2011 [98]	PCAR*	ANEN	High	High
Kapoor, 2013 [66]	PCAR*	ANEN	High	High
Yu, 2013 [122]	PCAL	ECCS	Low	Average
Sepúlveda, 2017 [103]	PCAR	ANEN	Average	High
Hussain, 2018 [57]	PCAR	ANEN	Average	High

Threat model: Packet Corruption at Routers (PCAR) and Packet Corruption at Links (PCAL). **Defense**: Authentication/Encryption (ANEN) and Error Correcting Codes (ECCS). **Overhead**: A three-step scale (low, average, high) compared to the overhead of the default (without security) architecture. **Effectiveness:** A three-step scale (low, average, high) based on the security guarantees against spoofing and data integrity attacks. "*" indicates that this is our best estimate, since the authors did not provide the exact information.

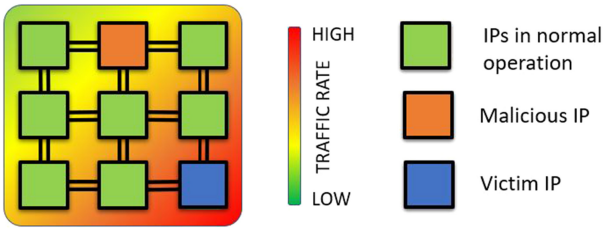


Fig. 11. Example DoS attack from a malicious IP to a victim IP in a Mesh NoC setup. The thermal map shows high traffic near the victim IP.

7 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 a NoC, several threat models have been explored. The DoS attack can have different effects based on the threat model such as performance degradation, quality of service degradation, real-time guarantee violations, and reduction of battery lifetime. In this section, we first discuss a few related threat models and introduce corresponding countermeasures.

7.1 Threat Models for Denial-of-service Attacks

In this section, we classify DoS attacks on NoC-based SoCs into different categories based on the nature of the threat model.

7.1.1 Flooding. DoS attacks can happen from malicious IPs manipulating the availability of on-chip resources by flooding the NoC with packets. The performance of a 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, then the SoC performance will suffer significant degradation [25, 26]. This is known as a flooding-type DoS attack. An illustrative example is shown in Figure 11 to demonstrate the flooding-type DoS attack. The additional packets injected from the

malicious IP to the victim IP (this can be a critical NoC component such as a memory controller) cause congestion in the routers, and responses experience severe delays.

Several other works discussed performance degradation by flooding the network with additional packets [47, 89]. In Reference [89], Trojans embedded in the router inject additional packets to the network to cause congestion. Additional packets are generated by sending illegal packet requests to the switch allocator when the cores are idling (core idle times during application execution).

Fang et al. explored the effects of a DoS attack in a NoC architecture with mesh topology. They showed that with changes to the attack traffic rate (i.e, severity of attack), different routing protocols will get affected differently [44].

7.1.2 Packet Corruption. Continuous corruption of packets can also lead to a DoS attack [49, 65]. In Reference [65], 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. In their work, four Trojans were introduced that differ based on the field in the flit they tamper with.

(1) Quan Trojan: modifies the flit quantity indication field. The destination detects that more or less flits are there than indicated by the flit quantity in head flit and drops the flits.
(2) Address Trojan: modifies the destination address field. This can lead to both information leakage, as explained in Section 5.1.1, and performance degradation, since legitimate packets sent to the destination can be blocked.
(3) Head hardware Trojan: modifies the head bit of the head flit. As a result, the route computation module will not access the address field, and the entire packet will be dropped, requesting a re-transmission.
(4) Tail hardware Trojan: modifies the tail indication bit of the tail flit. The destination waits for the tail to arrive, causing packet mixing and packet loss.

Boraten et al. [19] 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 Reference [20]. Compared to router-based packet corruption, they discussed a Trojan that performs deep packet inspection on links and injects 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. The Trojan tries to prevent infected links from being detected by altering the locations of faults to disguise them as transient faults.

7.1.3 Traffic Flow Manipulation. Rajesh et al. [64] 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. The authors show that once the malicious NoC IP is synthesized with TSMC 45nm library using the Synposys Design Compiler, the area and power overheads are 4.32% and 0.014%, respectively. Therefore, detecting the attack is difficult without additional security mechanisms.

Another method to manipulate the traffic flow is to attack the routing table and routing logic of a router. Biswas et al. [15] discussed several performance degradation attacks that can be launched by manipulating the routing protocol.

(1) Sub-optimal routing: deliberately route packets in sub-optimal paths to cause delays.
(2) Link overload: route packets through overlapping paths to cause congestion.
(3) Deletion of routing table entries: causing some packets never to reach the intended destination.
(4) Overwhelming IPs: re-route all packets to a victim IP to cause congestion.
(5) Deadlock/Livelock: force some packets to loop around and force deadlocks and livelocks.

7.2 Defenses against Denial-of-service Attacks

In this section, we classify defenses for DoS attacks on NoC-based SoCs into different categories based on the nature of the defense.

7.2.1 Fuzzing and Partitioning. The goal of fuzzing methods is to make the task of the Trojan harder. 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 [65]. In their work, a hardware Trojan in the router corrupts critical fields of packets such as flit quantity, address, head/tail flits, causing performance degradation. The attacker can target the critical fields, since the attacker is aware of the packet structure. The authors proposed to shuffle the critical bit fields of the flits among themselves and others so the Trojan is attacking on randomly shuffled data and not on the critical fields within the packets. The shuffling patterns are changed frequently. Therefore, the Trojan, which is unaware of the shuffling patterns, cannot target particular fields to launch a meaningful attack. Since the flit indication fields are typically one bit long, an error-correcting code based on a 1-bit Hamming code was also proposed that can complement the bit-shuffling mechanism.

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 [20]. 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. The SurfNoC architecture proposed in Reference [117] addressed both security and performance by dividing data carried by the NoC into different domains and sending data from different domains in different "waves" through the NoC. Waves are a form of temporal partitioning. This allows packets of one type to travel along the wave without interfering with other domains. As a result, the packets experience reduced latencies and minimize the risk of DoS and bandwidth depletion attacks from different domains.

7.2.2 Traffic Flow Monitoring. Monitoring the traffic flow to detect abnormalities is one common defense against DoS attacks. Rajesh et al. [64] 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, then they are expected to exhibit comparable latencies. If not, it will be flagged as a potential threat.

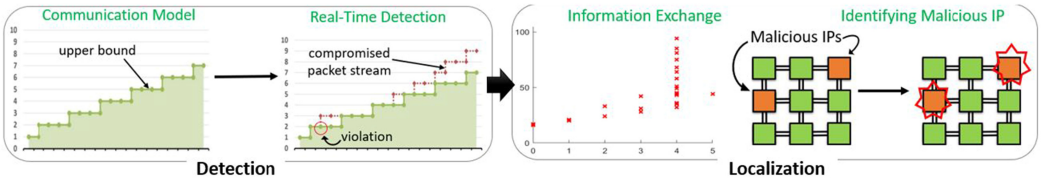


Fig. 12. Overview of the DoS attack detection and localization framework: The system specification is analyzed to obtain the necessary packet counts and detection parameters. These are used to design the real-time attack detection and localization framework [26].

Charles et al. presented a solution that statically profiles normal behavior of the SoC and detects both DoS [25] and distributed DoS [26] attacks during runtime. An overview of their approach is shown in Figure 12 [26]. Their solution has two major components. (i) DoS attack detection: Packet arrivals at all the routers are statically profiled during runtime to build upper bounds for the maximum number of packet arrivals within a given time window. During runtime, an approach based on the *Leaky Bucket Algorithm* [72] detects violations of the bound to flag potential DoS attacks. (ii) DoS attack localization: Once an attack has been detected, the malicious IP that launched the flooding type DoS attack is localized by communicating with routers along the congested path.

A similar method that profiled normal behavior of traffic during design time and monitored NoC traffic to detect deviations from normal behavior was proposed in Reference [47]. In their work, probes implemented inside the NI collect information about the NoC traffic and send to a central unit for decision-making.

Biswas et al. [15] proposed a solution that counts packets at each router input port to mitigate the threat of misrouted packets (discussed in Section 7.1.3). Their method uses the maximum inter-packet latency as the threshold to receive the next packet, and if not, a potential threat is flagged. The counter method is complemented by a packet header monitor that tries to identify misrouted packets based on header information. For example, if X-Y routing is used, a packet is not allowed to be routed in Y direction unless X dimensional routing is complete.

7.2.3 Additional Validation Checks. Validation techniques have been commonly used in functional validation. Prodromou et al. [90] proposed a runtime technique called NoCAlert to detect failures in the control logic of NoC components. The same analogy can be used to detect security violations as well. Enhancing the work done in Reference [90], Boraten et al. [19] proposed to use secure model checkers to alert the SoC if the control logic in NoC routers detect invariance violations caused by hardware Trojans placed in the control path. Their method is capable of detecting instances where the buffers, virtual channels, and switch allocators are illegally utilized, causing DoS attacks.

Sepúlveda et al. [99] defined properties that should hold in a router implementation to avoid DoS, packet corruption, packet duplication, and timing side-channel attacks. The properties were verified using unbounded model checking. They explored six different router implementations and discussed the validity of the properties in each implementation. Similar validation-based methods were proposed in References [49, 89]. Frey et al. [49] relied on security mechanisms implemented inside the NoC router to mitigate attacks. Their work combined flit scrambling to make the critical flit corruption hard and flit integrity checks to prevent bandwidth depletion attacks. In Reference [89], checks integrated at routers detected illegal packet injections.

7.3 Overhead and Effectiveness of Defenses against DoS Attacks

Defenses against DoS attacks try to prevent the attack from happening or detect and localize quickly if the attack is initiated. All these defense mechanisms include additional hardware and

Table 3. Threat Model, Defense, Overhead, and Effectiveness of Work in Existing Literature Related to DoS Attacks in NoC-based SoCs

	Threat Model	Defense	Overhead	Effectiveness
Fiorin, 2008 [47]	FLNG	TFMG	Average	High
Wassel, 2013 [117]	FLNG, TFMN	Low	Low	FGPG
Rajesh, 2015 [64]	TFMN	TFMG	Average	High
Biswas, 2015 [15]	TFMN	TFMG	Average	High
Boraten, 2016 [19]	PTCN	AVCS	Low	Low
Boraten, 2016 [20]	PTCN	FGPG, AVCS	Low	High
Prasad, 2017 [89]	FLNG	AVCS	Low	Average
Frey, 2017 [49]	PTCN	FGPG, AVCS	High	High
Kumar, 2018 [65]	PTCN	FGPG	Average	Low
Sepúlveda, 2018 [99]	PTCN, TFMN	AVCS	Low	Average
Charles, 2019 [25]	FLNG	TFMG	Low	High
Charles, 2020 [26]	FLNG	TFMG	Low	High

Threat model: Flooding (FLNG), Packet Corruption (PTCN), and Traffic Flow Manipulation (TFMN). **Defense**: Fuzzing and Partitioning (FGPG), Traffic Flow Monitoring (TFMG), and Additional Validation Checks (AVCS). **Overhead**: A three-step scale (low, average, high) compared to the overhead of the default (without security) architecture. **Effectiveness:** A three-step scale (low, average, high) based on the security guarantees against DoS attacks.

software, which adds to the overhead. The effectiveness of countermeasures against DoS attacks is evaluated by its ability to prevent the attack and the time taken to detect/localize after the attack is initiated. The ability to handle diverse use cases and attack scenarios also contributes to the effectiveness. Table 3 shows a summary of the papers we have surveyed related to denial-of-service attacks. The second and third columns outline the threat models and corresponding defenses, respectively. The last two columns provide the overhead as well as effectiveness of each countermeasure against denial-of-service attacks. We quantify the overhead and effectiveness of countermeasures using a three-step scale (low, average, high). For example, the countermeasure in Reference [20] introduced low overhead (2% area and 6% power consumption overhead) while being very (high) effective in defending against DoS attacks.

8 BUFFER OVERFLOW AND MEMORY EXTRACTION ATTACKS

The goal of a buffer overflow attack is to alter the function of a privileged program so the attacker can gain access and execute his own code. A program with high privileges (root programs) typically becomes 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 that will be stored in the program buffer. The string will contain some root-level instructions that the adversary wants the program to execute [35]. 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 the program will jump to that location and start executing the malicious code [76]. 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 a NoC context, the threat gets aggravated due to memory spaces being shared between multiple cores.

Defenses for the buffer overflow attacks discussed in the computer networks domain and in general can be classified into four main categories.

(1) Access authorization: when a read or write for the buffer comes in, the address is checked to see if it is within the allowed range and whether the address is allowed to access the requested memory location.
(2) Writing correct code: buffer overflow vulnerabilities can be caused from vulnerable features of the programming language itself. For example, library calls such as *strcpy* and *sprintf* in **C** language do not check the length of their arguments, and therefore can be exploited. Debugging tools are employed to identify and correct these codes.
(3) Non-executable data buffers: make the data area of the address space non-executable so the injected code cannot be executed.
(4) Integrity checking: detects code pointers that are corrupted by the attacker before they get referenced.

In the NoC setup, majority of the focus is on securing the SoC from buffer overflow attacks through access authorization. In this section, we first discuss a few related threat models and introduce corresponding countermeasures.

8.1 Threat Models for Buffer Overflow and Memory Extraction Attacks

In this section, we classify buffer overflow and memory extraction attacks on NoC-based SoCs into different categories based on the nature of the threat model.

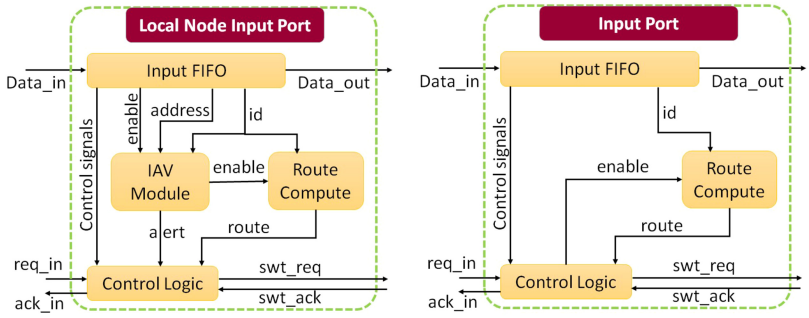
8.1.1 Smashing the Stack. Similar to the buffer overflow attacks in the computer networks domain described above, injecting the malicious code and getting it to execute can launch a buffer overflow attack. If a malicious IP writes on the stack and modifies the return address of a function to point at the malicious code, then 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" [74]. 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 Reference [76] explored this threat model. Buffer overflow attacks pose a significant threat in NoC-based SoCs, where the memory is shared among multiple cores.

8.1.2 Untrusted IPs Accessing Secure IPs. Kapoor et al. in their work considered some IPs on the SoC to contain confidential information (secure/trusted IP cores) and some untrusted IPs that can potentially carry hardware Trojans (non-secure/untrusted IP cores) [66]. 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 [66]. This threat model of non-secure IP cores trying to access secure-IP cores has been used in several other works as well [42, 46, 47, 96, 98].

8.2 Defenses against Buffer Overflow and Memory Extraction Attacks

In this section, we classify defenses for buffer overflow and memory extraction attacks on NoC-based SoCs into different categories based on the nature of the defense.

8.2.1 Access Authorization. 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)** [76]. 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



(a) With IAV. (b) Without IAV.

Fig. 13. Additional hardware complexity comparison of the router input port with and without the address verification unit (IAV) proposed in Reference [96]. The IAV protects the memory from attacks by verifying the ID and address of incoming requests.

unauthorized memory access requests. A **stack protection unit (SPU)** is developed that protects the stack from attacks that target 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 Reference [76] to a hierarchical security architecture [77]. 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 Reference [76], many existing works provide access control by monitoring the incoming request. Each countermeasure has slight variations, but the underlying core concept remains the same [46, 47, 96]. Saeed et al. introduced a method to mitigate buffer overflow attacks in a NoC-based shared memory architecture by deploying an **ID and address verification unit (IAV)** [96]. This minimizes the threats caused by malicious IPs in the NoC, because the IAV verifies each incoming packet by its ID and address. The additional hardware required to implement the IAV in the router is shown in Figure 13. The proposed architecture is capable of delivering security guarantees while causing less overhead compared to other similar methods [42, 46].

Fiorin et al. [46, 47] developed a similar security monitoring system for NoC-based architectures. In their work, a data protection unit examines the memory access requests and authorizes them after considering the memory address, type of operation requested (data load/store), and the status (role) of the initiator (user or superuser mode). This acts as a firewall that stops illegal access requests.

8.2.2 Partitioning into Secure and Non-secure Zones. Adding an extra layer of security to access authorization, commercial products such as Sonic SMART Interconnect [110] and ARM TrustZone [3] 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, then requests are validated by access authorization techniques [66, 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. [100] achieved this by using a partitioning method that used a lightweight Diffie-Hellman key-exchange

Table 4. Threat Model, Defense, Overhead, and Effectiveness of Work in Existing Literature Related to Buffer Overflow and Memory Extraction Attacks in NoC-based SoCs

	Threat Model	Defense	Overhead	Effectiveness
Diguet, 2007 [42]	UIAS, STSK	ASAN	Average	High
Fiorin, 2008 [46]	UIAS	ASAN	Average	High
Fiorin, 2008 [47]	UIAS, STSK	ASAN	Average	High
Lukovic, 2010 [76]	STSK	ASAN	Average*	High
Lukovic, 2010 [77]	STSK	ASAN	High*	High
Porquet, 2011 [88]	UIAS*	PSNZ	Low	Average
Sajeesh, 2011 [98]	UIAS	PSNZ	High	High
Kapoor, 2013 [66]	UIAS	PSNZ	Average	Average
Saeed, 2014 [96]	UIAS	ASAN	Low	High
Sepúlveda, 2014 [102]	UIAS	PSNZ	Average*	High
Sepúlveda, 2015 [100]	UIAS	PSNZ	Low	High

Threat model: Smashing the Stack (STSK) and Untrusted IPs Accessing Secure IPs (UIAS). **Defense**: Access Authorization (ASAN) and Partitioning into Secure and Non-secure Zones (PSNZ). **Overhead**: A three-step scale (low, average, high) compared to the overhead of the default (without security) architecture. **Effectiveness:** A three-step scale (low, average, high) based on the security guarantees against buffer overflow and memory extraction attacks. "*" indicates that this is our best estimate, since the authors did not provide the exact information.

protocol. Their architecture provided authentication, access control, and confidentiality services for applications running on secure cores. The same authors proposed a method to create dynamic firewalls at the network interface to monitor and filter the NoC traffic [102]. 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. [88] 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, and so on, when multiple virtual machines are running on the same host machine [22].

8.3 Overhead and Effectiveness of Defenses against Buffer Overflow and Memory Extraction Attacks

The most common defense designed against buffer overflow and memory extraction attacks is to validate the legitimacy of the access request based on the content as well as the requester. The validation can be done either at the network interfaces, memory controllers, or the routers. Additional hardware and software is required to implement the validation techniques, which contributes to the overhead. For example, the address verification unit illustrated in Figure 13 [96] accounts for an area increase of 1.5% per router compared to the default router implementation. The effectiveness of countermeasures is evaluated based on its ability to prevent illegal access requests and detect such attempts. Table 4 shows a summary of the papers we have surveyed related to buffer overflow and memory extraction attacks. The second and third columns outline the threat models and corresponding defenses, respectively. The last two columns provide the overhead as well as effectiveness of each countermeasure. We quantify the overhead and effectiveness of countermeasures using a three-step scale (low, average, high).

9 SIDE-CHANNEL ATTACKS

In the early days, attacks on encrypted devices were based on observing ciphertext (ciphertext-only attacks), knowing both plaintext and ciphertext (plaintext attacks), or selecting a set of plaintexts to encrypt and observing the results of encryption (chosen plaintext attacks). In contrast to that, side-channel attacks exploit non-functional behavior such as time, power, electromagnetic radiation, heat, and acoustic waveforms to attack a secure system [123]. 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 [63]. In contrast, a paper published in 2012 showed that a brute-force attack on AES using a super computer can take 149 trillion years [6]. Even though computing resources have significantly improved since then, a brute-force attack on AES-128 is still not possible.

The possibility of side-channel attacks escalated, since in a realistic scenario, more constraints are imposed on the system, such as performance and power other than security. Even for systems with theoretically proven security bounds, revealing the secrets through these non-functional physical properties is a likely scenario.

9.1 Threat Models for Side-channel Attacks

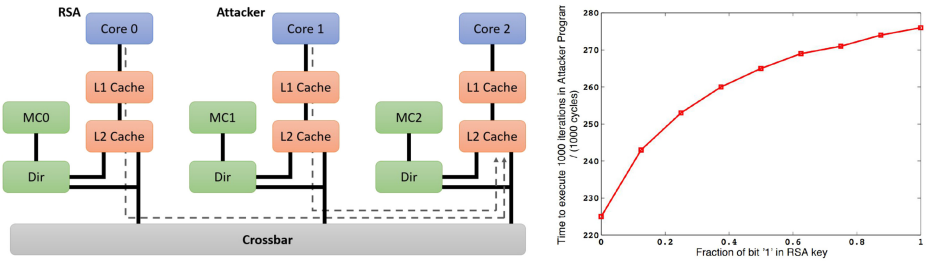
In this section, we classify side-channel attacks on NoC-based SoCs into different categories based on the nature of the threat model.

9.1.1 Timing Side Channels. 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 a NoC-based SoC [94]. The target of their attack was the communication between an ARM Cortex-A9 core and a shared cache memory. This is carried out in four steps;

(1) Infection: attacker stores malware in the SoC.
(2) Prime: attacker overwrites some locations of the shared cache. This is done to make sure there are no **Advanced Encryption Standard (AES)** lookup tables in the cache before the attack starts.
(3) Probe: monitor the cache access locations during execution.
(4) Analysis: analyze the access patterns to reveal information about the secret key.

Other studies carried out on timing attacks also used similar concepts on timing analysis of network traffic for attacks [60, 61, 93, 116]. The threat model in Reference [61] included four cores: two of which are carrying out a secure communication, and the other two, which lie 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. [116] in their work showed that the number of ones in the RSA [95] 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 (1,024 or 2,048-bit) numbers. The modulo multiplication is shown to be vulnerable to timing side-channel attacks [70], 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



(a) Architectural setup.

(b) Attack program's execution time.

Fig. 14. Illustrative example of a side-channel attack on RSA [116].

multiplications [116]. An example attack scenario is shown in Figure 14. Multiplication operation causes additional NoC traffic to the **memory controller 2 (MC2)** due to cache conflicts. The attack program running on Core 1 also sends requests to MC2. Since packets from Core 0 and Core 1 both share the same output port of the crossbar, the packets experience network interference, causing the correlation between the attack program's runtime and the number of ones in the RSA key to be enhanced. Sepúlveda et al. [101, 104] used a similar threat model in their NoC timing side-channel analysis.

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. [18] proposed a differential cache collision attack on embedded systems. While their work did not consider a NoC-based setup, in 2018, Reinbrecht et al. [92] showed that combining their previous work on NoC timing attacks [93] with Bogdanov et al's cache collision attack [18] can significantly enhance the AES key recovery effort.

9.1.2 Power/Thermal Side Channels. 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), then 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 that can be utilized to break the security scheme [32]. 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 [69].

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. [53] 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.

Figure 15 shows a graphical representation of the two characteristics.

9.2 Defenses against Side-channel Attacks

The most common method to mitigate side-channel attacks is to "cloud" the information an attacker can obtain. For example, consider the case of an if-else statement. Assume that if the

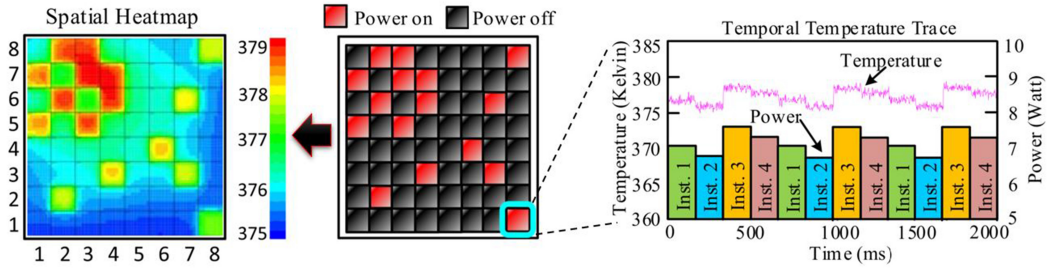


Fig. 15. Examples to show how SoC operation impacts spatial and temporal thermal characteristics [53].

condition is true, then the computation takes a long time, whereas if it is false, then the else statement is executed and it takes a much shorter time. An adversary mounting a timing attack can observe the execution time and identify whether the code branched or not. However, if the computationally intensive block was always executed irrespective of whether it is used or not, then the attacker can gain no useful information by a timing analysis. However, if the condition is false, then the results of the computation will no longer be used, and therefore, will introduce performance overhead. This is the tradeoff between security and performance.

The defenses in a NoC context follow the same underlying concepts where the major focus is to randomize application execution in terms of data, packet routes, and application placement. In this section, we classify defenses for side-channel attacks on NoC-based SoCs into different categories based on the nature of the defense.

9.2.1 Route Randomization. The Prime+Probe attack model that mounts a timing attack on a NoC was discussed in Section 9.1.1 [94]. As a countermeasure, the authors proposed *Gossip NoC* [93, 94]—a two-stage security mechanism that 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 beach. The protection mechanism gets triggered by this alert message that 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 References [60, 61]. The authors propose several design alternatives for route randomization. Even though randomization can increase NoC resilience, the performance impact might cause some tasks to be non-schedulable. The tradeoff between security and schedulability is extensively studied in their work. 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 [104].

9.2.2 Application and Data Randomization. As a solution to the thermal side-channel attacks discussed in Reference [53], 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. Even though their work considers a NoC-based SoC, neither the threat model nor the solution utilize the unique capabilities offered by the NoC. Most of the side-channel threats that utilize NoC characteristics fall in the category of timing side-channel attacks. However, we introduce the thermal side-channel attack in a NoC-based SoC in this survey as a motivation for future research efforts. Even though an explicit threat model was not discussed in Reference [4], the data scrambling and node obfuscation methods, proposed as a part of the suite of security countermeasures, has the "clouding" effect that provides resilience to side-channel attacks.

9.2.3 Traffic Partitioning. To avoid timing side-channel attacks similar to the one introduced in Reference [116], 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 [116]. Their method assigns a high priority to non-secure application traffic so the traffic behavior is not affected by traffic from secure applications. At buffers, virtual channels are statically allocated to avoid interference. To avoid DoS attacks that can happen due to prioritizing non-secure application traffic over secure application traffic, a limit is set for the number of low-security flits that can traverse through a given router port within a certain interval of time. This method eliminates correlation between secure and non-secure application traffic and allows dynamic allocation of resources to minimize performance degradation.

A similar principal was used in the *Secure Enhanced Router* architecture proposed by Sepúlveda et al. [101]. 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 Reference [116], the goal was to make the non-secure traffic flow oblivious of the secure traffic flow.

9.3 Overhead and Effectiveness of Defenses against Side-channel Attacks

Most NoC-specific side-channel attacks focus on exploiting the timing side channel and the most common countermeasure is to "cloud" the information an attacker can obtain. For example, if an attacker can observe whether a complex instruction is being executed or a memory fetch is initiated by observing the time, then one approach is to randomly delay even the simple instructions that typically finish execution quickly. Such approaches introduce performance overhead. Furthermore, the hardware and software required to handle the countermeasure introduces power and area overhead. The gossip router proposed in Reference [93] exhibited 21% area and 16% power overhead when compared to the typical router (without countermeasure). The effectiveness of countermeasures against side-channel attacks is measured by its ability to mitigate the attack as well as their adaptability and scalability to different use case scenarios. Table 5 shows a summary of the papers we have surveyed related to side-channel attacks. The second and third columns outline the threat models and corresponding defenses, respectively. The last two columns provide the overhead as well as effectiveness of each countermeasure using a three-step scale (low, average, high).

10 CONCLUSION AND FUTURE DIRECTIONS

In this article, we have surveyed security vulnerabilities and defenses in NoC-based SoCs. We have considered existing literature published by both industry and academia outlining state-of-the-art attacks and defense mechanisms. The literature contains a significant amount of work related to on-chip network security. In particular, we have discussed the papers under five classes of attacks, highlighting their threat models and respective countermeasures. Security countermeasures were compared in terms of pros and cons. Table 6 provides a summary of papers we have considered in this survey.

The NoC security countermeasures developed so far have, for the most part, shown promising results. The challenge has been addressing the security concerns while staying within the

Table 5. Threat Model, Defense, Overhead, and Effectiveness of Work in Existing Literature Related to Side-channel Attacks in NoC-based SoCs

	Threat Model	Defense	Overhead	Effectiveness
Wang, 2012 [116]	TSCS	TCPG	Low	Average
Ancajas, 2014 [4]	TSCS*	AADR	Low	Low
Sepúlveda, 2014 [104]	TSCS	RERN	Average	Average
Reinbrecht, 2016 [94]	TSCS	RERN	Low	Average
Reinbrecht, 2016 [93]	TSCS	RERN, TCPG	Low	Average
Sepúlveda, 2016 [101]	TSCS	TCPG	Average	High
Indrusiak, 2017 [61]	TSCS	RERN	Average	Average
Indrusiak, 2019 [60]	TSCS	RERN	Average	Average

Threat model: Timing Side Channels (TSCS). **Defense**: Route Randomization (RERN), Application and Data Randomization (AADR), and Traffic Partitioning (TCPG). **Overhead**: A three-step scale (low, average, high) compared to the overhead of the default (without security) architecture. **Effectiveness:** A three-step scale (low, average, high) based on the security guarantees against side-channel attacks. "*" indicates that this is our best estimate, since the authors did not provide the exact information.

non-functional requirements such as power, performance, and area. Experience tells us that there is still room for improvement by utilizing the unique characteristics of NoC traffic. In this section, we outline a few research directions that we believe are interesting and open problems in the domain.

- **Machine learning and security:** The intersection of machine learning and security has not been given much attention in a NoC context. Apart from a few runtime monitoring techniques that used machine learning concepts, not a lot of work has been done in that regard. Tools such as *Cisco Encrypted Traffic Analytics* [87] utilize machine 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 have the potential to provide security guarantees, especially for real-time and safety-critical applications.
- **Reconfigurable security:** Increased usage of NoC-based SoCs in IoT devices has introduced new security challenges [28]. When embedded devices first came into consumer use, they were intended for one or few use cases. In comparison, IoT and embedded devices today are intended to serve a variety of use cases. Furthermore, the device life expectancy has considerably increased. IoT nodes are expected to work over long periods of time. Therefore, any security mechanisms developed for these devices are expected to survive over extended periods of time over several use-case scenarios. Reconfiguration techniques such as cache reconfiguration has been explored to optimize SoC power and performance [24]. Similarly, reconfigurable security is an interesting avenue that we believe should be explored further to address the requirements of "future-proof" devices.
- **Assertion-based security:** Validating NoC packets as well as components with additional validation checks was discussed in Section 7.2.3. The two main techniques used so far are formal verification and simulation-based techniques. While formal methods can provide security guarantees, the complexity of NoC designs makes the exploration space grow exponentially. However, 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 not adequately investigated.

Table 6. Summary of NoC Security Papers Found in Literature Categorized by Attack Class, Attack Location, and Defense Type

	Attack Class	Attack Location	Defense Type
Gebotys, 2003 [50]	EAVE, SDIY, SCAS	Router, MIP*	OBFU
Diguet, 2007 [42]	DOSE, BOME	MIP	DETE
Fiorin, 2008 [46]	BOME	MIP	DETE
Fiorin, 2008 [47]	DOSE, BOME	MIP	DETE, LOCA
Lukovic, 2010 [77]	BOME	MIP	DETE
Lukovic, 2010 [76]	BOME	MIP	DETE
Sajeesh, 2011 [98]	EAVE, SDIY, BOME	MIP	OBFU, DETE
Porquet, 2011 [88]	BOME	MIP*	OBFU
Wang, 2012 [116]	DOSE, SCAS	MIP	OBFU
Kapoor, 2013 [66]	EAVE, SDIY, DOSE, BOME	MIP	OBFU, DETE
Yu, 2013 [122]	SDIY	Link	OBFU
Wassel, 2013 [117]	DOSE	MIP*	OBFU
Sepúlveda, 2014 [102]	BOME	MIP	OBFU, DETE
Sepúlveda, 2014 [104]	SCAS	MIP	OBFU
Ancajas, 2014 [4]	EAVE, SCAS	Router	OBFU
Saeed, 2014 [96]	BOME	MIP	DETE
Sepúlveda, 2015 [100]	BOME	MIP	OBFU, DETE
Rajesh, 2015 [64]	DOSE	Router	DETE
Biswas, 2015 [15]	DOSE	Router	DETE
Reinbrecht, 2016 [93]	SCAS	MIP	OBFU, DETE
Reinbrecht, 2016 [94]	SCAS	MIP	OBFU, DETE
Sepúlveda, 2016 [101]	SCAS	MIP	OBFU
Boraten, 2016 [19]	DOSE	Router	DETE, LOCA
Boraten, 2016 [20]	DOSE	Link	OBFU, DETE, LOCA
Boraten, 2016 [21]	EAVE, SDIY, SCAS	Router, Link	OBFU
Prasad, 2017 [89]	DOSE	Router	DETE
Sepúlveda, 2017 [103]	EAVE, SDIY	Router, MIP	OBFU
Frey, 2017 [49]	DOSE	Router	OBFU, DETE
Indrusiak, 2017 [61]	SCAS	MIP	OBFU
Sepúlveda, 2018 [99]	SDIY, DOSE, SCAS	Router	DETE
Reinbrecht, 2018 [92]	SCAS	MIP	N/A
Hussain, 2018 [57]	EAVE, SDIY, DOSE	Router	DETE, LOCA
Kumar, 2018 [65]	EAVE, SDIY, DOSE	Router	OBFU
Indrusiak, 2019 [60]	SCAS	MIP	OBFU
Charles, 2019 [25]	DOSE	MIP	DETE, LOCA
Raparti, 2019 [91]	EAVE	NI	DETE, LOCA
Charles, 2020 [27]	EAVE	Router	OBFU
Charles, 2020 [26]	DOSE	MIP	DETE, LOCA

Attack Class: Eavesdropping (EAVE), Spoofing/Data Integrity (SDIY), Denial-of-service (DOSE), Buffer Overflow and Memory Extraction (BOME), and Side-channel Attacks (SCAS). **Attack Location**: Router, Network Interface (NI), Link and Other Malicious Intellectual Property Core (MIP). **Defense Type**: Obfuscation (OBFU), Detection (DETE), and Localization (LOCA). "*" indicates that this is our best estimate, since the authors did not provide the exact information. N/A indicates that a defense is beyond the scope of this article, since the work introduced only an attack scenario.

- **Seamless integration of security mechanisms:** While existing literature has discussed several 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)** [33] provides hardware-based software protection techniques. Similarly, how to integrate several NoC security mechanisms and ensuring their inter-operability in hardware, firmware, and software layers is worth exploring further.
- **Emerging side-channel attacks:** Spectre [68] and Meltdown [75] have shown that still there is plenty of room for side-channel attacks in the hardware domain. With the ubiquity of devices, side-channel attacks are going to get cheaper and easier. The widespread usage of SoCs in IoT and embedded systems gives increasing importance to protecting the NoC from side-channel attacks. In addition to the attacks and defenses described here, designers also utilize side-channel analysis to determine potential side-channel vulnerabilities and develop mitigation techniques. In many cases, it is hard to identify side-channel vulnerabilities. For example, it is difficult to figure out the presence of a few-gate hardware Trojan in a multi-million-gate design, since the side-channel footprint will hide in typical process variations and environmental noise margins. Recent efforts try to improve the side-channel sensitivity through effective combination of logic testing and side-channel analysis. For example, in dynamic current (power)-based side-channel analysis, the researchers try to generate test patterns such that it maximizes the switching in suspicious regions while it minimizes the switching in the rest of the design [54, 55, 78]. Similarly, in delay-based side-channel analysis, the focus is to alter the critical paths when a Trojan is activated [79]. Clearly, architectural side-channel defense mechanisms coupled with effective side-channel analysis would be ideal for protecting NoC-based SoCs against emerging side-channel attacks.

The future of NoC-based SoC architectures with the emergence of giga- and tera-scale architectures can impose more challenges. With challenges come unique opportunities as well. The introduction of emerging NoC technologies such as wireless and optical have already shown promising results. However, developing efficient and flexible solutions at lower costs and minimal impact on performance as well as how to certify these solutions remain as challenges to the industry. Therefore, there is still a lot of work to do to achieve the desired results.

REFERENCES

[1] Assad Abbas, Mazhar Ali, Ahmad Fayyaz, Ankan Ghosh, Anshul Kalra, Samee U. Khan, Muhammad Usman Shahid Khan, Thiago De Menezes, Sayantica Pattanayak, Alarka Sanyal et al. 2014. A survey on energy-efficient methodologies and architectures of network-on-chip. *Computers. Electric. Eng.* 40, 8 (2014), 333–347.

[2] Ankur Agarwal, Cyril Iskander, and Ravi Shankar. 2009. Survey of network on chip (NoC) architectures & contributions. *J. Eng. Comput. Archit.* 3, 1 (2009), 21–27.

[3] Tiago Alves. 2004. Trustzone: Integrated hardware and software security. White Paper (2004). Retrieved from https://www.techonline.com/tech-papers/trustzone-integrated-hardware-and-software-security/.

[4] Dean Michael Ancajas, Koushik Chakraborty, and Sanghamitra Roy. 2014. Fort-NoCs: Mitigating the threat of a compromised NoC. In *Proceedings of the 51st Design Automation Conference*. ACM, 1–6.

[5] ARM. 1999. *Amba specification.* Technical Report. ARM, Revision 2.0. Retrieved from https://developer.arm.com/products/architecture/amba-protocol.

[6] M. Arora. 2012. How secure is AES against brute force attacks. *EE Times* 5, 7 (2012).

[7] Arteris. 2009. FlexNoC Resilience Package. Retrieved from www.arteris.com/flexnoc-resilience-package-functional-safety.

[8] Semiconductor Industry Association. 2015. International Technology Roadmap for Semiconductors (ITRS). Retrieved from www.semiconductors.org/main/2015_international_technology_roadmap_for_semiconductors_itrs/.
[9] Semiconductor Industry Association et al. 2011. *International Technology Roadmap for Semiconductors (ITRS).* 2003 edition. *Incheon, Korea.*.
[10] Jean-Philippe Aumasson and Daniel J. Bernstein. 2012. SipHash: A fast short-input PRF. In *Proceedings of the International Conference on Cryptology in India.* Springer, 489–508.
[11] Luca Benini and Giovanni De Micheli. 2002. Networks on chips: A new SoC paradigm. *Computer* 35, 1 (2002), 70–78.
[12] Swarup Bhunia, Michael S. Hsiao, Mainak Banga, and Seetharam Narasimhan. 2014. Hardware Trojan attacks: Threat analysis and countermeasures. *Proc. IEEE* 102, 8 (2014), 1229–1247.
[13] Swarup Bhunia and M. Tehranipoor. 2018. *The Hardware Trojan War.* Springer.
[14] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti et al. 2011. The gem5 simulator. *ACM SIGARCH Comput. Archit. News* 39, 2 (2011), 1–7.
[15] Arnab Kumar Biswas, S. K. Nandy, and Ranjani Narayan. 2015. Router attack toward NoC-enabled MPSoC and monitoring countermeasures against such threat. *Circ., Syst., Sig. Proc.* 34, 10 (2015), 3241–3290.
[16] Tobias Bjerregaard and Shankar Mahadevan. 2006. A survey of research and practices of network-on-chip. *ACM Comput. Surv.* 38, 1 (2006).
[17] Bloomberg. 2018. *The Big Hack: How China Used a Tiny Chip to Infiltrate U.S. Companies.* Retrieved from https://www.bloomberg.com/news/features/2018-10-04/the-big-hack-how-china-used-a-tiny-chip-to-infiltrate-america-s-top-companies.
[18] Andrey Bogdanov, Thomas Eisenbarth, Christof Paar, and Malte Wienecke. 2010. Differential cache-collision timing attacks on AES with applications to embedded CPUs. In *Cryptographers' Track at the RSA Conference.* 235–251.
[19] Travis Boraten, Dominic DiTomaso, and Avinash Karanth Kodi. 2016. Secure model checkers for Network-on-Chip (NoC) architectures. In *Proceedings of the International Great Lakes Symposium on VLSI (GLSVLSI'16).* IEEE, 45–50.
[20] Travis Boraten and Avinash Karanth Kodi. 2016. Mitigation of denial of service attack with hardware trojans in NoC architectures. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS'16).* IEEE, 1091–1100.
[21] Travis Boraten and Avinash Karanth Kodi. 2016. Packet security with path sensitization for NoCs. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE'16).* IEEE, 1136–1139.
[22] Thomas C. Bressoud and Fred B. Schneider. 1996. Hypervisor-based fault tolerance. *ACM Trans. Comput. Syst.* 14, 1 (1996), 80–107.
[23] Thanh Bui, Siddharth Prakash Rao, Markku Antikainen, Viswanathan Manihatty Bojan, and Tuomas Aura. 2018. Man-in-the-machine: Exploiting ill-secured communication inside the computer. In *Proceedings of the 27th USENIX Security Symposium (USENIX Security'18).* 1511–1525.
[24] Subodha Charles, Alif Ahmed, Umit Y. Ogras, and Prabhat Mishra. 2019. Efficient cache reconfiguration using machine learning in NoC-based many-core CMPs. *ACM Trans. Des. Automat. Electron. Syst.* 24, 6 (2019), 1–23.
[25] Subodha Charles, Yangdi Lyu, and Prabhat Mishra. 2019. Real-time detection and localization of DoS attacks in NoC based SoCs. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE'19).* IEEE, 1160–1165.
[26] Subodha Charles, Yangdi Lyu, and Prabhat Mishra. 2020. Real-time detection and localization of distributed DoS attacks in NoC based SoCs. *IEEE Trans. Comput.-aided Des. Integ. Circ. Syst.* 25, 6 (2020).
[27] Subodha Charles and Prabhat Mishra. 2020. Lightweight and trust-aware routing in NoC-based SoCs. In *Proceedings of the IEEE Computer Society Symposium on VLSI (ISVLSI'20).* IEEE, 160–167.
[28] Subodha Charles and Prabhat Mishra. 2020. Reconfigurable network-on-chip security architecture. *ACM Trans. Des. Automat. Electron. Syst.* 25, 6 (2020), 1–25.
[29] Subodha Charles and Prabhat Mishra. 2020. Securing network-on-chip using incremental cryptography. In *Proceedings of the IEEE Computer Society Symposium on VLSI (ISVLSI'20).* IEEE, 168–175.
[30] Guoqing Chen, Hui Chen, Mikhail Haurylau, Nicholas A. Nelson, David H. Albonesi, Philippe M. Fauchet, and Eby G. Friedman. 2006. On-chip copper-based vs. optical interconnects: Delay uncertainty, latency, power, and bandwidth density comparative predictions. In *Proceedings of the International Interconnect Technology Conference.* IEEE, 39–41.
[31] Sai Vineel Reddy Chittamuru, Ishan G. Thakkar, Varun Bhat, and Sudeep Pasricha. 2018. SOTERIA: Exploiting process variations to enhance hardware security with photonic NoC architectures. In *Proceedings of the 55th ACM/ESDA/IEEE Design Automation Conference (DAC'18).* IEEE, 1–6.
[32] Omar Choudary et al. 2005. *Breaking Smartcards Using Power Analysis.* University of Cambridge.
[33] Victor Costan and Srinivas Devadas. 2016. Intel SGX explained.*IACR Cryptology ePrint Archive* (2016), 1–118.

[34] Érika Cota, Alexandre de Morais Amory, and Marcelo Soares Lubaszewski. 2012. NoC basics. In *Reliability, Availability and Serviceability of Networks-on-Chip*. Springer, 11–24.
[35] Crispin Cowan, F. Wagle, Calton Pu, Steve Beattie, and Jonathan Walpole. 2000. Buffer overflows: Attacks and defenses for the vulnerability of the decade. In *Proceedings of the DARPA Information Survivability Conference and Exposition (DISCEX'00)*. IEEE, 119–129.
[36] Joan Daemen and Vincent Rijmen. 2013. *The Design of Rijndael: AES-the Advanced Encryption Standard*. Springer Science & Business Media.
[37] Matteo Dall'Osso, Gianluca Biccari, Luca Giovannini, Davide Bertozzi, and Luca Benini. 2012. Xpipes: A latency insensitive parameterized network-on-chip architecture for multi-processor SoCs. In *Proceedings of the IEEE 30th International Conference on Computer Design (ICCD'12)*. IEEE, 45–48.
[38] William J. Dally and Brian Towles. 2001. Route packets, not wires: On-chip interconnection networks. In *Proceedings of the Design Automation Conference*. IEEE, 684–689.
[39] DARPA. 2017. System Security Integrated Through Hardware and Firmware (SSITH). Retrieved from www.darpa.mil.
[40] A. Vieira de Mello, Luciano Copello Ost, Fernando Gehm Moraes, and Ney Laert Vilar Calazans. 2004. Evaluation of routing algorithms on mesh based NoCs. *PUCRS, Av. Ipiranga* (2004), 22.
[41] Sujay Deb, Amlan Ganguly, Partha Pratim Pande, Benjamin Belzer, and Deukhyoun Heo. 2012. Wireless NoC as interconnection backbone for multicore chips: Promises and challenges. *IEEE J. Emerg. Select. Topics Circ. Syst.* 2, 2 (2012), 228–239.
[42] Jean-Philippe Diguet, Samuel Evain, Romain Vaslin, Guy Gogniat, and Emmanuel Juin. 2007. NOC-centric security of reconfigurable SoC. In *Proceedings of the 1st International Symposium on Networks-on-Chip (NOCS'07)*. IEEE, 223–232.
[43] Dave Evans. 2011. The internet of things: How the next evolution of the internet is changing everything. *CISCO White Paper* 1, 2011 (2011), 1–11.
[44] Dabin Fang, Huikai Li, Jun Han, and Xiaoyang Zeng. 2013. Robustness analysis of mesh-based network-on-chip architecture under flooding-based denial of service attacks. In *Proceedings of the IEEE8th International Conference on Networking, Architecture and Storage*. IEEE, 178–186.
[45] Farimah Farahmandi, Yuanwen Huang, and Prabhat Mishra. 2019. *System-on-Chip Security: Validation and Verification*. Springer Nature.
[46] Leandro Fiorin, Gianluca Palermo, Slobodan Lukovic, Valerio Catalano, and Cristina Silvano. 2008. Secure memory accesses on networks-on-chip. *IEEE Trans. Comput.* 57, 9 (2008), 1216–1229.
[47] Leandro Fiorin, Gianluca Palermo, and Cristina Silvano. 2008. A security monitoring service for NoCs. In *Proceedings of the 6th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*. ACM, 197–202.
[48] Leandro Fiorin, Cristina Silvano, and Mariagiovanna Sami. 2007. Security aspects in networks-on-chips: Overview and proposals for secure implementations. In *Proceedings of the 10th Euromicro Conference on Digital System Design Architectures, Methods and Tools (DSD'07)*. IEEE, 539–542.
[49] Jonathan Frey and Qiaoyan Yu. 2017. A hardened network-on-chip design using runtime hardware Trojan mitigation methods. *Integration* 56 (2017), 15–31.
[50] Catherine H. Gebotys and Robert J. Gebotys. 2003. A framework for security on NoC technologies. In *Proceedings of the IEEE Computer Society Symposium on VLSI*. IEEE, 113–117.
[51] David Goldschlag, Michael Reed, and Paul Syverson. 1999. Onion routing. *Commun. ACM* 42, 2 (1999), 39–41.
[52] Kees Goossens, John Dielissen, and Andrei Radulescu. 2005. Æthereal network on chip: Concepts, architectures, and implementations. *IEEE Des. Test Comput.* 22, 5 (2005), 414–421.
[53] Shize Guo, Jian Wang, Zhe Chen, Zhonghai Lu, Jinhong Guo, and Lian Yang. 2019. Security-aware task mapping reducing thermal side channel leakage in CMPs. *IEEE Trans. Industr. Inform.* 15, 10 (2019), 5435–5443.
[54] Yuanwen Huang, Swarup Bhunia, and Prabhat Mishra. 2016. MERS: Statistical test generation for side-channel analysis based Trojan detection. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. 130–141.
[55] Yuanwen Huang, Swarup Bhunia, and Prabhat Mishra. 2018. Scalable test generation for Trojan detection using side channel analysis. *IEEE Trans. Inform. Forens. Secur.* 13, 11 (2018), 2746–2760.
[56] W. Cary Huffman and Vera Pless. 2010. *Fundamentals of Error-correcting Codes*. Cambridge University Press.
[57] Mubashir Hussain, Amin Malekpour, Hui Guo, and Sri Parameswaran. 2018. EETD: An energy efficient design for runtime hardware trojan detection in untrusted network-on-chip. In *Proceedings of the IEEE Computer Society Symposium on VLSI (ISVLSI'18)*. IEEE, 345–350.
[58] IDC. 2019. The growth in connected IoT devices is expected to generate 79.4 ZB of data in 2025, according to a new IDC forecast. (2019). Retrieved from https://www.businesswire.com/news/home/20190618005012/en/The-Growth-in-Connected-IoT-Devices-is-Expected-to-Generate-79.4ZB-of-Data-in-2025-According-to-a-New-IDC-Forecast.

[59] Gartner Inc.2014. Semiconductor design IP revenue, chip infrastructure, worldwide, 2012 and 2013. *Gartner Res.* (Mar. 2014), 70–78.
[60] Leandro Soares Indrusiak, James Harbin, Cezar Reinbrecht, and Johanna Sepúlveda. 2019. Side-channel protected MPSoC through secure real-time networks-on-chip. *Microproc. Microsyst.* 68 (2019), 34–46.
[61] Leandro Soares Indrusiak, James Harbin, and Martha Johanna Sepulveda. 2017. Side-channel attack resilience through route randomisation in secure real-time networks-on-chip. In *Proceedings of the International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC'17)*. 1–8.
[62] Intel. 2016. Using TinyCrypt Library, Intel Developer Zone. Retrieved from software.intel.com/en-us/node/734330.
[63] Zhen Hang Jiang, Yunsi Fei, and David Kaeli. 2017. A novel side-channel timing attack on GPUs. In *Proceedings of the Great Lakes Symposium on VLSI*. 167–172.
[64] Rajesh J. S., Dean Michael Ancajas, Koushik Chakraborty, and Sanghamitra Roy. 2015. Runtime detection of a bandwidth denial attack from a rogue network-on-chip. In *Proceedings of the 9th International Symposium on Networks-on-Chip*. ACM, 8.
[65] Manoj Kumar J. Y. V., Ayas Kanta Swain, Sudeendra Kumar, Sauvagya Ranjan Sahoo, and Kamalakanta Mahapatra. 2018. Run time mitigation of performance degradation hardware trojan attacks in network on chip. In *Proceedings of the IEEE Computer Society Symposium on VLSI (ISVLSI'18)*. IEEE, 738–743.
[66] Hemangee K. Kapoor, G. Bhoopal Rao, Sharique Arshi, and Gaurav Trivedi. 2013. A security framework for NoC using authenticated encryption and session keys. *Circ., Syst., Sig. Proc.* 32, 6 (2013), 2605–2622.
[67] Samuel T. King, Joseph Tucek, Anthony Cozzie, Chris Grier, Weihang Jiang, and Yuanyuan Zhou. 2008. Designing and implementing malicious hardware. In *Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*. 1–8.
[68] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. 2019. Spectre attacks: Exploiting speculative execution. In *Proceedings of the 40th IEEE Symposium on Security and Privacy (S&P'19)*.
[69] Paul Kocher, Joshua Jaffe, Benjamin Jun, and Pankaj Rohatgi. 2011. Introduction to differential power analysis. *J. Cryptog. Eng.* 1, 1 (2011), 5–27.
[70] Paul C. Kocher. 1996. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Proceedings of the International Cryptology Conference*. Springer, 104–113.
[71] Shashi Kumar, Axel Jantsch, J.-P. Soininen, Martti Forsell, Mikael Millberg, Johny Oberg, Kari Tiensyrja, and Ahmed Hemani. 2002. A network on chip architecture and design methodology. In *Proceedings of the IEEE Computer Society Symposium on VLSI: New Paradigms for VLSI Systems Design (ISVLSI'02)*. IEEE, 117–124.
[72] Jean-Yves Le Boudec and Patrick Thiran. 2001. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Springer.
[73] Brian Lebiednik, Sergi Abadal, Hyoukjun Kwon, and Tushar Krishna. 2018. Architecting a secure wireless network-on-chip. In *Proceedings of the 12th IEEE/ACM International Symposium on Networks-on-Chip (NOCS'18)*. IEEE, 1–8.
[74] Elias Levi. 1996. Smashing the stack for fun and profit. *Phrack Mag.* 7, 49 (1996).
[75] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. 2018. Meltdown: Reading kernel memory from user space. In *Proceedings of the 27th USENIX Security Symposium (USENIX Security'18)*.
[76] Slobodan Lukovic and Nikolaos Christianos. 2010. Enhancing network-on-chip components to support security of processing elements. In *Proceedings of the Workshop on Embedded Systems Security*. 12.
[77] Slobodan Lukovic and Nikolaos Christianos. 2010. Hierarchical multi-agent protection system for NoC based MPSoCs. In *Proceedings of the International Workshop on Security and Dependability for Resource Constrained Embedded Systems*.
[78] Yangdi Lyu and Prabhat Mishra. 2019. Efficient test generation for Trojan detection using side channel analysis. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE'19)*. IEEE, 408–413.
[79] Yangdi Lyu and Prabhat Mishra. 2020. Automated test generation for Trojan detection using delay-based side channel analysis. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE'20)*. IEEE.
[80] Robert A. Martin. 2007. Common weakness enumeration. Mitre Corp. (2007). Retrieved from https://cwe.mitre.org/.
[81] David McGrew and John Viega. 2004. The Galois/counter mode of operation (GCM). *Submiss. NIST Modes Oper. Proc.* 20 (2004).
[82] IBM Microelectronics. 1999. CoreConnect bus architecture. IBM White Paper (1999). Retrieved from http://www.scarpaz.com/2100-papers/SystemOnChip/ibm_core_connect_whitepaper.pdf.
[83] Prabhat Mishra, Swarup Bhunia, and Mark Tehranipoor. 2017. *Hardware IP Security and Trust*. Springer.
[84] Azad Naeemi, Reza Sarvari, and James D. Meindl. 2006. On-chip interconnect networks at the end of the roadmap: Limits and nanotechnology opportunities. In *Proceedings of the International Interconnect Technology Conference*. IEEE, 201–203.

[85] Sudeep Pasricha and Nikil Dutt. 2008. ORB: An on-chip optical ring bus communication architecture for multi-processor systems-on-chip. In *Proceedings of the Asia and South Pacific Design Automation Conference*. IEEE, 789–794.

[86] Sudeep Pasricha and Nikil Dutt. 2010. *On-chip Communication Architectures: System on Chip Interconnect*. Morgan Kaufmann.

[87] Santosh Ramrao Patil, Gangadharan Byju Pularikkal, David McGrew, Blake Harrell Anderson, and Madhusudan Nanjanagud. 2019. Encrypted traffic analytics over a multi-path TCP connection. US Patent App. 15/891,708.

[88] Joël Porquet, Alain Greiner, and Christian Schwarz. 2011. NoC-MPU: A secure architecture for flexible co-hosting on shared memory MPSoCs. In *Proceedings of the Design, Automation & Test in Europe Conference*. IEEE, 1–4.

[89] N. Prasad, Rajit Karmakar, Santanu Chattopadhyay, and Indrajit Chakrabarti. 2017. Runtime mitigation of illegal packet request attacks in Networks-on-Chip. In *Proceedings of the International Symposium on Circuits and Systems (ISCAS'17)*. 1–4.

[90] Andreas Prodromou, Andreas Panteli, Chrysostomos Nicopoulos, and Yiannakis Sazeides. 2012. NoCalert: An on-line and real-time fault detection mechanism for network-on-chip architectures. In *Proceedings of the 45th IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 60–71.

[91] Venkata Yaswanth Raparti and Sudeep Pasricha. 2019. Lightweight mitigation of hardware Trojan attacks in NoC-based manycore computing. In *Proceedings of the 56th Design Automation Conference*. ACM, 48.

[92] Cezar Reinbrecht, Bruno Forlin, Andreas Zankl, and Johanna Sepúlveda. 2018. Earthquake—A NoC-based optimized differential cache-collision attack for MPSoCs. In *Proceedings of the Design, Automation & Test in Europe (DATE'18)*. 648–653.

[93] Cezar Reinbrecht, Altamiro Susin, Lilian Bossuet, and Johanna Sepúlveda. 2016. Gossip NoC–avoiding timing side-channel attacks through traffic management. In *Proceedings of the Symposium on VLSI (ISVLSI'16)*. 601–606.

[94] Cezar Reinbrecht, Altamiro Susin, Lilian Bossuet, Georg Sigl, and Johanna Sepúlveda. 2016. Side channel attack on NoC-based MPSoCs are practical: NoC Prime+ Probe attack. In *Proceedings of the Symposium on Integrated Circuits and Systems Design (SBCCI'16)*. 1–6.

[95] Ronald L. Rivest, Adi Shamir, and Leonard Adleman. 1983. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM* 26, 1 (1983), 96–99.

[96] Ahmed Saeed, Ali Ahmadinia, Mike Just, and Christophe Bobda. 2014. 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, 288.

[97] Selma Saidi, Rolf Ernst, Sascha Uhrig, Henrik Theiling, and Benoît Dupont de Dinechin. 2015. The shift to multicores in real-time and safety-critical systems. In *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS'15)*. IEEE, 220–229.

[98] K. Sajeesh and Hemangee K. Kapoor. 2011. An authenticated encryption based security framework for NoC architectures. In *Proceedings of the International Symposium on Electronic System Design*. IEEE, 134–139.

[99] Johanna Sepúlveda, Damian Aboul-Hassan, Georg Sigl, Bernd Becker, and Matthias Sauer. 2018. Towards the formal verification of security properties of a Network-on-Chip router. In *Proceedings of the European Test Symposium (ETS'18)*. 1–6.

[100] Johanna Sepúlveda, Daniel Flórez, and Guy Gogniat. 2015. Reconfigurable security architecture for disrupted protection zones in NoC-based MPSoCs. In *Proceedings of the International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC'15)*. 1–8.

[101] Johanna Sepúlveda, Daniel Flórez, Mathias Soeken, Jean-Philippe Diguet, and Guy Gogniat. 2016. Dynamic NoC buffer allocation for MPSoC timing side channel attack protection. In *Proceedings of the Latin American Symposium on Circuits & Systems (LASCAS'16)*. 91–94.

[102] Johanna Sepúlveda, Guy Gogniat, Daniel Florez, Jean-Philippe Diguet, Cesar Zeferino, and Marius Strum. 2014. Elastic security zones for NoC-based 3D-MPSoCs. In *Proceedings of the IEEE International Conference on Electronics, Circuits and Systems (ICECS'14)*. 506–509.

[103] Johanna Sepúlveda, Andreas Zankl, Daniel Flórez, and Georg Sigl. 2017. Towards protected MPSoC communication for information protection against a malicious NoC. *Procedia Comput. Sci.* 108 (2017), 1103–1112.

[104] Martha Johanna Sepulveda, Jean-Philippe Diguet, Marius Strum, and Guy Gogniat. 2014. NoC-based protection for SoC time-driven attacks. *IEEE Embed. Syst. Lett.* 7, 1 (2014), 7–10.

[105] K. Shuler. 2013. Majority of leading China semiconductor companies rely on arteris network-on-chip interconnect IP.

[106] K. Shuler. 2013. Arteris Makes Big Gains on Inc. 500 List of America's Fastest-Growing Private Companies. Retrieved from www.arteris.com/Inc-500-Arteris-pr-2013-august-20.

[107] David Sigüenza-Tortosa, Tapani Ahonen, and Jari Nurmi. 2004. Issues in the development of a practical NoC: The Proteo concept. *Integration* 38, 1 (2004), 95–105.

[108] Sergei Skorobogatov and Christopher Woods. 2012. Breakthrough silicon scanning discovers backdoor in military chip. In *Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 23–40.

[109] Avinash Sodani, Roger Gramunt, Jesus Corbal, Ho-Seop Kim, Krishna Vinod, Sundaram Chinthamani, Steven Hutsell, Rajat Agarwal, and Yen-Chen Liu. 2016. Knights landing: Second-generation Intel Xeon phi product. *IEEE MICRO* 36, 2 (2016), 34–46.

[110] SONICS. 2011. SONICS NoCk-Lock Security. Retrieved from www.sonicsinc.com.

[111] Navin Srivastava and Kaustav Banerjee. 2004. A comparative scaling analysis of metallic and carbon nanotube interconnections for nanometer scale VLSI technologies. In *Proceedings of the 21st International VLSI Multilevel Interconnect Conference*. 393–398.

[112] Navin Srivastava and Kaustav Banerjee. 2005. Performance analysis of carbon nanotube interconnects for VLSI applications. In *Proceedings of the IEEE/ACM International Conference on Computer-aided Design (ICCAD'05)*. IEEE, 383–390.

[113] Werner Steinhögl, Günther Schindler, Gernot Steinlesberger, and Manfred Engelhardt. 2002. Size-dependent resistivity of metallic wires in the mesoscopic range. *Phys. Rev. B* 66, 7 (2002), 075414.

[114] Mohammad Tehranipoor and Farinaz Koushanfar. 2010. A survey of hardware trojan taxonomy and detection. *IEEE Des. Test Comput.* 27, 1 (2010), 10–25.

[115] Shuai Wang and Tao Jin. 2014. Wireless network-on-chip: A survey. *J. Eng.* 2014, 3 (2014), 98–104.

[116] Yao Wang and G. Edward Suh. 2012. Efficient timing channel protection for on-chip networks. In *Proceedings of the IEEE/ACM 6th International Symposium on Networks-on-Chip*. IEEE, 142–151.

[117] Hassan M. G. Wassel, Ying Gao, Jason K. Oberg, Ted Huffmire, Ryan Kastner, Frederic T. Chong, and Timothy Sherwood. 2013. SurfNoC: A low latency and provably non-interfering approach to secure networks-on-chip. *ACM SIGARCH Comput. Archit. News* 41, 3 (2013), 583–594.

[118] David Wentzlaff, Patrick Griffin, Henry Hoffmann, Liewei Bao, Bruce Edwards, Carl Ramey, Matthew Mattina, Chyi-Chang Miao, John F. Brown III, and Anant Agarwal. 2007. On-chip interconnection architecture of the tile processor. *IEEE Micro*5 (2007), 15–31.

[119] Sebastian Werner, Javier Navaridas, and Mikel Luján. 2017. A survey on optical network-on-chip architectures. *ACM Comput. Surv.* 50, 6 (2017), 1–37.

[120] Pascal T. Wolkotte, Gerard J. M. Smit, Gerard K. Rauwerda, and Lodewijk T. Smit. 2005. An energy-efficient reconfigurable circuit-switched network-on-chip. In *Proceedings of the International Parallel and Distributed Processing Symposium*.

[121] Yang Xiao. 2016. *Security in Sensor Networks*. CRC Press.

[122] Qiaoyan Yu and Jonathan Frey. 2013. Exploiting error control approaches for hardware trojans on network-on-chip links. In *Proceedings of the International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS'13)*. 266–271.

[123] Weize Yu, Orhun Aras Uzun, and Selçuk Köse. 2015. Leveraging on-chip voltage regulators as a countermeasure against side-channel attacks. In *Proceedings of the 52nd Design Automation Conference*. 1–6.

[124] Sebastian Zander, Grenville Armitage, and Philip Branch. 2007. A survey of covert channels and countermeasures in computer network protocols. *IEEE Commun. Surv. Tutor.* 9, 3 (2007), 44–57.

[125] Da-Kun Zhang, Cui Huang, and Guo-Zhi Song. 2016. Survey on three-dimensional network-on-chip. *J. Softw.* 27, 1 (2016), 155–187.

[126] Lei Zhang. 2011. Optical network-on-chip architectures and designs. (2011). Retrieved from https://digitalscholarship.unlv.edu/thesesdissertations/1037/.

[127] Haibo Zhu, Partha Pratim Pande, and Cristian Grecu. 2007. Performance evaluation of adaptive routing algorithms for achieving fault tolerance in NoC fabrics. In *Proceedings of the IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP'07)*. IEEE, 42–47.

Received April 2020; revised December 2020; accepted February 2021