Traffic Analysis Attacks on Wireless NoC-based SoCs

Hansika Weerasena and Prabhat Mishra
Department of Computer & Information Science & Engineering
University of Florida, Gainesville, Florida, USA

***Abstract*—Network-on-Chip (NoC) enables on-chip communication between diverse cores in modern System-on-Chip (SoC) designs. Despite the advantages of wireless NoCs (WiNoC) over its electrical counterpart, the use of a shared medium during wireless communication introduces a unique set of vulnerabilities. In WiNoCs, the medium access control (MAC) protocol orchestrates traffic flow across wireless hubs, shaping how traffic patterns emerge based on the protocol and the applications. In this paper, we propose a traffic analysis attack on wireless traffic governed by MAC to infer applications running in an SoC. Specifically, this paper the following major contributions. We outline a threat model involving a malicious wireless hub and a colluding application to leak traffic traces. We utilize deep learning to exploit spatiotemporal traffic features of the MAC protocol to infer applications running in the system, posing significant security and privacy risks. Extensive evaluation demonstrates that our proposed traffic analysis attack can infer applications with high accuracy (95%–99%) across diverse WiNoC configurations and applications. We also propose a lightweight countermeasure to defend against traffic analysis attack that uses stochastic scheduling of wireless packets over virtual wireless hubs. Experimental results show that our proposed countermeasure can defend against traffic analysis attacks with minimal overhead.**

***Index Terms*—system-on-chip, network-on-chip, on-chip communication, wireless NoC, deep neural networks, CNNs**

I. Introduction

Parallel workloads and specialized computing accelerators like neural network accelerators have become key in advancing computing capabilities, significantly influencing modern processor architecture designs. Heterogeneous Systems-on-Chips (SoCs) and Multi-Processor SoCs (MPSoCs) now integrate numerous Intellectual Property (IP) cores in a single chip, reflecting a major shift towards more complex and powerful systems. For example, commercial MPSoCs such as Altra® multicore server processors have 192 cores [1]. The heterogeneous integration and 3D stacking of IPs have gained attention as a promising path for extending Moore's Law [2].

Network on Chip (NoC) has become the de facto standard for communication, addressing the sophisticated needs of densely packed IP cores. It offers a robust and low latency framework for core-to-core communication in SoCs. For instance, companies like Intel employ Skylake Mesh NoC [3] in their server-grade processors. Wireless NoCs (WiNoCs) offer more efficient and scalable communication than traditional electrical NoCs. One-hop communication in WiNoC greatly reduces energy and packet latency, and its broadcast/multicast nature decreases the total packet count compared to electrical NoCs. WiNoCs with fully wireless topology relies entirely on wireless communication, while hybrid WiNoCs uses electrical connections for short distance communication and wireless for longer ones. WiNoCs use a Medium Access Control (MAC) protocol to arbitrate access to the shared wireless medium among wireless hubs. A Time Division Multiple Access (TDMA) based MAC protocol (T-MAC) is used in WiNoCs [4] due to its simplicity and less protocol overhead. Figure 1 illustrates an example scenario of T-MAC usage in a 4x4 fully wireless NoC. In T-MAC, each wireless hub will receive a time slot, where only that hub can send packets.

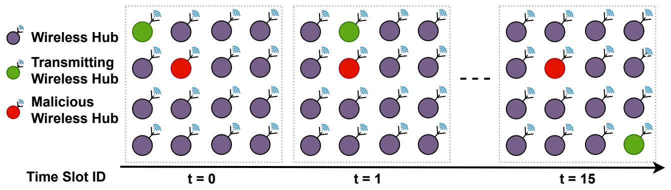

Fig. 1: T-MAC in 4x4 mesh WiNoC: Malicious wireless hub can eavesdrop on communication at every time slot.

SoC development with third-party IPs raises security concerns. These IPs can harbor hardware Trojans, hidden backdoors, and undocumented bugs [5]. Additionally, long supply chains and potentially untrusted vendors further heighten security risks in modern SoC designs. Moreover, the growing complexity of SoC designs makes comprehensive security verification increasingly difficult. Semiconductor Research Corporation's (SRC) Microelectronics and Advanced Packaging Technologies (MAPT) Roadmap [6] highlights Heterogeneous Integration (HI) as a key approach for achieving cost- and power-efficient designs in next-generation computing systems. HI increases security concerns by expanding the attack surface due to its complexity. The scalability of HI enables the integration of on-chip Machine Learning (ML) accelerators. Advances in ML and the availability of on-chip ML accelerators create opportunities for ML-based attacks in modern SoCs by leveraging their ability to analyze complex patterns.

The NoC has become a focal point of attack for adversaries since NoC has an extensive attack surface having access to all components in the SoC. Security issues across different NoC technologies (electrical, wireless, and optical) have been studied in the literature [7]. Despite WiNoCs' advantages, their shared and unguided medium heightens security vulnerabilities. Recent studies have also explored various attacks on WiNoCs and countermeasures [8], [9], [10], [11], [12], including snooping, denial-of-service, and spoofing. As shown in Figure 1, due to the shared medium and broadcast communication in WiNoCs, all hubs can observe packets transmitted

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

in a specific time slot, allowing malicious hubs to potentially steal sensitive information. Even if the packets are encrypted by lightweight encryption mechanisms [8], [9], [13], [10], the deterministic nature of TDMA allows malicious hubs to monitor time slot usage by each hub. In other words, regardless of whether messages are encrypted or unencrypted, distinct traffic patterns can still be observed for each application.

In this paper, we analyze this vulnerability and show that an adversary can use it for ML-based traffic analysis attacks to deduce applications running in WiNoC-based SoCs, thereby violating the privacy of the system and its stakeholders. Specifically, this paper makes the following major contributions:

- We propose a threat model with a malicious wireless hub and a colluding application to exploit the broadcast nature of wireless communication to leak T-MAC traffic traces.
- We model T-MAC traffic in a WiNoC as a spatiotemporal matrix to train convolutional neural network at design time and use it to predict applications in run time.
- Our attack demonstrates high accuracy (95%–99%) across diverse WiNoC configurations and benchmarks.
- We propose a lightweight countermeasure by traffic reshaping via virtual MAC modules.
- Experimental results show that our proposed countermeasure can defend against the traffic analysis attack.

This paper is organized as follows. Section II provides background and surveys related efforts. Section III provides motivation, while Section IV outlines the threat model. Section V describes our traffic analysis attack followed by a lightweight countermeasure in Section VI. Section VII presents the experimental results. Finally, Section VIII concludes the paper.

II. Background and Related Work

This section first introduces the TDMA-based MAC protocol for WiNoCs. Next, it surveys prior efforts on NoC security.

A. *Media Access Control (MAC) in Wireless NoCs (WiNoCs)*

WiNoCs feature CMOS-compatible wireless hubs operating at millimeter-wave frequencies. Due to complex transceiver designs and limited bandwidth, most WiNoCs use a single wireless channel for data communication [14], [15]. The MAC protocol governs the division of the shared medium among wireless hubs. WiNoC requires a simple, lightweight MAC protocol. A notable implementation is the T-MAC protocol, which employs a token-based TDMA mechanism [4], [15], [16], [17]. Figure 2 shows a hybrid mesh WiNoC using the T-MAC protocol, where a token as a control packet assigns time slots to each wireless hub (WH). In NoC communication, a packet is divided into smaller flow units called flits [7]. The T-MAC protocol permits only the token-holding hub to broadcast flits into the wireless medium. All hubs receive the flit because they are tuned to the same frequency, but only the hub with a matching destination address processes it further. Once all flits in a packet are sent, the token passes in a round-robin to the next hub. If a hub has no flits to transfer, it immediately passes the token to the next hub. T-MAC is ideal for WiNoC environments because of its low overhead and simplicity.

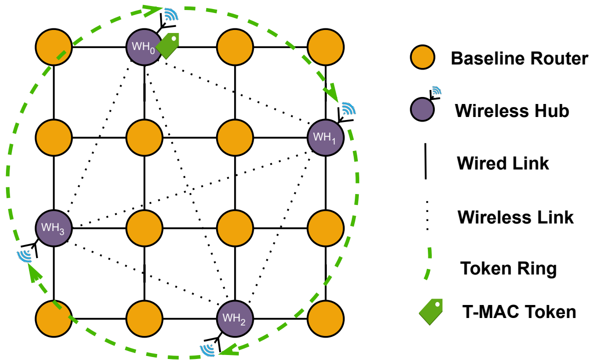

Fig. 2: Time division multiplexing in 4x4 hybrid WiNoC: the T-MAC token is circulated in round-robin manner.

Example 1: Consider the WiNoC in Figure 2 with 4 wireless hubs (WHs) ($n = 4$) and a packet sequence of 6 packets ($P_0 - P_5$). Each packet is described by three parameters: the hub ID where it arrives, the arrival cycle, and its size in flits. For example, P_0 (WH_2, 0, 1) indicates that packet P_0 arrives at the hub WH_2 in cycle 0 and consists of 1 flit. The packet sequence is as follows: P_0 (WH_2, 0, 1), P_1 (WH_0, 4, 1), P_2 (WH_3, 7, 1), P_3 (WH_1, 12, 5), P_4 (WH_0, 21, 5), and P_5 (WH_2, 32, 5). Assume communication takes one cycle per flit and token packet is one flit. Table I lists the time slot occupancy for each hub over 6 slots. Each entry shows the slot occupancy (A ✓ indicates if a packet uses the time slot, 'X' otherwise), the token arrival cycle (the cycle when the token arrived at the hub), the hub's active cycles (even without transmission, the hub takes one cycle for token handling), and the packet ID, if applicable. The token circulates in a round-robin manner, first passing from WH_0 to WH_4 during slot 0, and then repeating the same order for subsequent slots. In other words, the chronological order of entries in Table I is as follows: **slot 0** [WH_0 (X,0,1) $\rightarrow \ldots \rightarrow$ WH_3 (X,4,1)] $\rightarrow \ldots \rightarrow$ **slot 5** [WH_0 (X,33,1) $\rightarrow \ldots \rightarrow$ WH_2 (✓,35,6, P_5)].

TABLE I: Time slot occupancy for each hub using T-MAC for packet sequence given in Example 1.

	slot 0	slot 1	slot 2	slot 3	slot 4	slot 5
WH_0	X, 0, 1	✓, 5, 2, P_1	X, 11, 1	X, 20, 1	✓, 24, 6, P_4	X, 33, 1
WH_1	X, 1, 1	X, 7, 1	✓, 12, 6, P_3	X, 21, 1	X, 30 1	X, 34, 1
WH_2	✓, 2, 2, P_0	X, 8, 1	X, 18, 1	X, 22, 1	X, 31, 1	✓, 35, 6, P_5
WH_3	X, 4, 1	✓, 9, 2, P_2	X, 19, 1	X, 23, 1	X, 32, 1	–

Let's focus on the first row's 5th entry (✓, 24, 6, P_4). The '✓' indicates the time slot was occupied, and the last value specifies it was for packet P_4. The clock cycle at the start of the slot is 24; in other words, WH_0 received the token at cycle 24. The 6 cycles used by WH_0 correspond to 5 non-token flits and one token control flit. The original packet sequence shows that P_4 arrives the hub at cycle 21. The packet P_4 was dispatched in the very next time slot occupied by the source hub (WH_0) after the packet's arrival to WH_0 in cycle 21. ■

B. *Related Work*

Security attacks and defenses on NoC can be categorized by the compromised security requirements: confidentiality, integrity, anonymity, authenticity, availability, and freshness [7]. A malicious NoC collaborating with an application can launch various attacks on electrical NoC (ENoC), including attacks on anonymity [18], snooping attacks [19], [20], spoofing attacks [21], denial-of-service (DoS) attacks [20] and side-channel attacks [22]. For example, [19] uses a Hardware Trojan (HT) at the Network Interface to snoop and leak data to a malicious program at another IP. Our attack also involves a malicious NoC collaborating with an application.

The Remote Access Hardware Trojan (RAHT), introduced in [23], facilitates traffic analysis attacks in ENoC. RAHT is a minimal HT that can transmit sensitive information to external adversaries, enabling sophisticated attacks. Its detection is challenging due to its minimal impact on area, power, and timing. Recent efforts [23], [24] use an RAHT-based threat model in ENoC that can identify applications running in SoC through a ML-based traffic analysis attack. Specifically, they use packet counts at intermediate routers as features, and their countermeasure targets simulated annealing-based randomized routing. However, such attacks and countermeasures are not applicable to WiNoC-based SoCs for the following three reasons. (1) The approaches in [23], [24] rely on the routing mechanisms used in ENoC, specifically exploiting existing routing mechanisms for attacks and proposing robust routing mechanisms for defense. In contrast, WiNoC communication does not use routing; it employs channel arbitration, which is a fundamentally different concept. (2) Traffic patterns in ENoC and WiNoC differ significantly. (3) Our attack uses spatiotemporal features which are significantly different from the features used in [23], [24]. Security of wireless and optical technologies has been explored by [7]. Diverse attacks on WiNoCs and their mitigations have been discussed in the literature, including eavesdropping [8], [9], [10], packet tampering [8], spoofing [8], and, DoS attacks [9], [25], [10], [11], [12]. None of these attacks consider traffic analysis attacks in WiNoCs. *To the best of our knowledge, our proposed approach is the first attempt at traffic analysis attack on wireless NoCs.*

III. Motivation

NoC communicates cache coherence messages for applications running on IP cores in MPSoCs. Typically, two primary packet types are used in NoC: control packets and data packets. For example, a cache miss in a core's private cache triggers a control packet to request data from the shared cache or main memory. The memory controller responds with a data packet containing the requested cache block. In the network interface, packets are segmented into flits, the basic flow control units in NoC. Under the T-MAC protocol, when a packet is available, all its flits are transferred within the hub's dedicated time slot.

Figure 3 shows time slot occupancy for two SPLASH-2 [26] parallel benchmarks (*fft* and *fmm*) across four wireless hubs in a 2x2 fully wireless NoC, over a window of 300 time slots. We can see temporal (across time slots) and spatial (across hubs) differences in time-slot occupancy for two benchmarks. It is evident that the computing and communication nature of applications is reflected through the T-MAC protocol. Figure 4 shows the number of control and data packets over a 300-slot window for seven benchmarks from SPLASH-2 [26] and PARSEC [27], with experimental parameters consistent with those in Figure 3. The shared medium of WiNoC enables a malicious wireless hub to eavesdrop and collect data like hub-wise time slot occupancy and flit counts per time slot, as shown in Figure 3. More complex traffic features, such as packet/flit counts, can be derived from these raw features, as depicted in Figure 4. Preliminary results led us to use deep neural networks (DNNs) for traffic analysis attacks because of their advanced pattern recognition capabilities. We aim to predict applications running on WiNoC-based SoCs by analyzing traffic pattern leaks due to T-MAC usage. Identifying applications in MPSoCs raises significant security and privacy concerns, making it crucial to analyze these vulnerabilities and develop countermeasures for WiNoC-based SoCs.

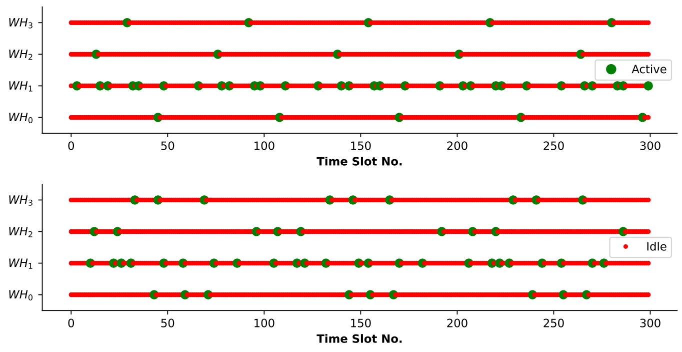

Fig. 3: Time slot occupancy using T-MAC in 2x2 mesh WiNoC with 4 wireless hubs (WHs): (top) *fft* and (bottom) *fmm*.

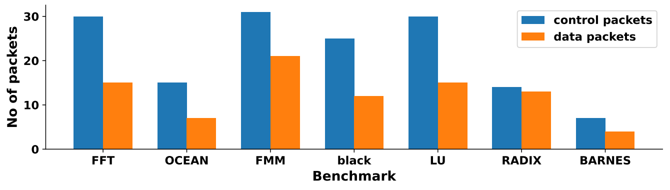

Fig. 4: Number of control and data packets transferred via WiNoC for seven different benchmarks.

This research faces unique challenges. Determining applications involves more than simple one-to-one pattern matching because: 1) different phases and mappings of the same application show varied traffic patterns, and 2) system noise and traffic trace collection by the adversary can obscure traffic patterns. Given the complexity and novelty of the ML problem and data, it's crucial to develop a robust, efficient, and lightweight DNN architecture. Website fingerprinting in traditional networks [28], [29] is closely related to proposed attack but differs significantly: 1) it relies on temporal features, whereas we use spatiotemporal features; 2) it involves more innate features and variance, as traditional networks transmit a wider range of packet sizes compared to NoC's two (control and data) packet sizes. A key challenge in securing T-MAC protocol against traffic analysis is enhancing communication security without compromising its lightweight design.

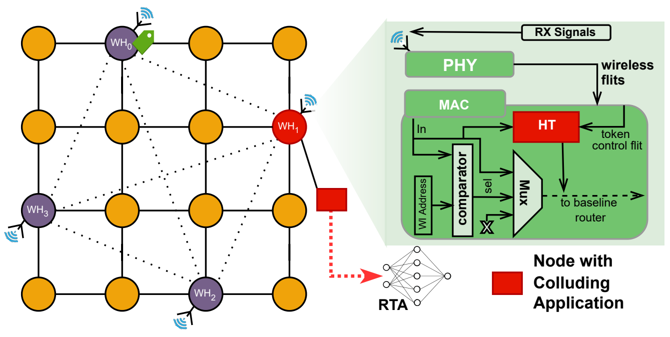

Fig. 5: A malicious wireless hub (WH) has an HT in the MAC layer, a colluding application runs on an MPSoC node, and a remote traffic analyzer resides outside the MPSoC.

IV. Threat Model

We consider a WiNoC-based MPSoC in a multi-tenant cloud, featuring multiple nodes with wireless hubs. The WiNoC architecture varies, including fully wireless, mesh-based hybrid, or small-world-based hybrid topologies. In a fully wireless architecture, each hub contains only a wireless interface, whereas in other setups, wireless hubs also include a baseline router for electrical communication. We assume the NoC packets are encrypted, thus not susceptible to eavesdropping attacks that leaks sensitive information such as the identity of the application. Figure 5 shows an overview of the threat model. A third-party-fabricated WiNoC could have a stealthy hardware Trojan (HT) inserted into a wireless hub during design or fabrication. This malicious hub exploits wireless communication's broadcast nature to collect T-MAC traffic traces. In a multi-tenant cloud, an adversary can deploy a colluding application (CA) on an MPSoC node, disguised as legitimate software. The HT compresses and packetizes the traces, sending them to the CA as legitimate NoC packets via wired or wireless connections. For wireless transmission, the adversary's observed traffic patterns may be slightly influenced by malicious trace packets; however, this impact is accounted for in the noise model, as discussed in Section VII-B.

The CA forwards traffic traces to a remote adversary for analysis via process-to-process communication. It also manages the activation and deactivation of the HT using a trigger. The trigger mechanism is outlined in Section V-A, while the activation and deactivation process of the HT is discussed in Section V-B. The remote traffic analyzer (RTA), equipped with a pre-trained DNN model, communicates with the CA via standard application layer communication. The trend of heterogeneous integration in packaging could enhance the proposed attack's efficiency by relocating the DNN model to an on-chip accelerator in future.

The adversary knows the system's micro-architecture, including operating frequencies, buffer capacities, wireless hub positioning, routing algorithms, MAC protocol, and topology. Since these elements are fixed, traffic patterns vary based on the application and its mapping. The goal is to demonstrate that it is feasible to gain insight into which applications are running on the system if an adversary has access to these architectural details and use a simulator or emulator to generate traffic traces. Detecting this attack is challenging with current tools and techniques due to multiple reasons. (1) The HT is minimal with low power and area footprint, making it undetectable during design inspections or operational monitoring [23]. This is experimentally elaborated in Table VII in Section VII. (2) Because HT passively leaks traffic information without interfering with the rest of the system, anomaly detection mechanisms are ineffective. (3) Detection becomes harder as the colluding application blends with diverse MPSoC applications, mimicking legitimate software, while HT-to-CA communication appears as regular cache coherence messages.

The attacker aims to identify specific applications running on the WiNoC by analyzing the T-MAC based traffic patterns. The target applications are part of a predefined suite and run alongside other general applications on the MPSoC. For example, the application suite could include encryption algorithms such as AES [30], RSA [31], and ChaCha20 [32]. The traffic from other applications complicates identification. This attack significantly impacts both privacy and security. For example, unauthorized discovery of the applications a user runs violates privacy and compromises the confidentiality essential in confidential computing. Knowing an application's identity not only enables attackers to tailor their attacks to exploit known vulnerabilities but also aids in crafting methods to bypass application-specific security measures. Additionally, it enables application-specific side-channel attacks, like inferring cryptographic keys after identifying the algorithm. Moreover, this knowledge can be used to launch targeted DoS attacks, maximizing disruption and resource consumption.

V. Traffic Analysis Attack

Figure 6 shows an overview of our proposed traffic analysis attack. The threat model (Figure 5) includes a hardware Trojan (HT) within a wireless hub, a colluding application (CA), and a remote traffic analyzer (RTA). Normally, wireless hubs ignore packets not addressed to them, but the HT bypasses this filter, leaking traffic patterns to the CA. The CA then relays this information to the RTA, which uses a pre-trained DNN model to identify the active application.

A. Hardware Trojan Design and Functionality

The primary goal of the HT is to exploit the broadcast nature and shared medium of the WiNoC, leaking T-MAC traffic patterns to the CA. The wireless hub consists of two key layers: the Physical (PHY) layer and the Medium Access Control (MAC) layer. Figure 5 shows the detailed structure of the hub, highlighting the connection between the receiver (RX) section of the PHY layer and the MAC layer. The PHY layer receives broadcast RX signals through the transceiver, converts them into flits, and sends them to the MAC layer. The MAC layer uses the T-MAC protocol to manage medium access, processing RX signals for two main functions. 1) It processes token control packets and assigns the token if destined for local hub. 2) It compares the destination address in the non-token packet header with the hub's address using a comparator

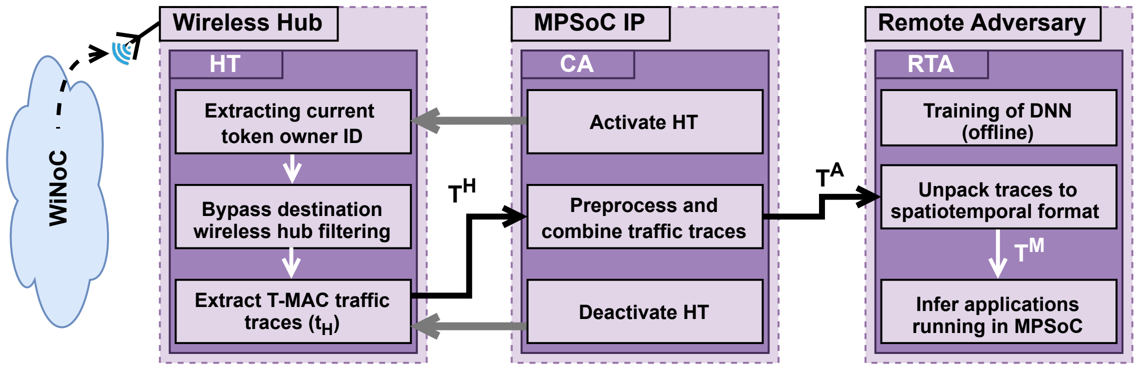
WiNoC
Wireless Hub
HT
Extracting current token owner ID
Bypass destination wireless hub filtering
Extract T-MAC traffic traces (t_H)
T^H
MPSoC IP
CA
Activate HT
Preprocess and combine traffic traces
Deactivate HT
T^A
Remote Adversary
RTA
Training of DNN (offline)
Unpack traces to spatiotemporal format
T^M
Infer applications running in MPSoC

Fig. 6: Attack Overview: The wireless hub with the HT tracks the current token owner, bypasses packet filtering, and extracts T-MAC traffic traces. The HT redirects traces to a node with a colluding application (CA), which then forwards them to a remote traffic analyzer (RTA). Then RTA will use the pre-trained DNN model to predict applications running on the MPSoC.

and then uses a multiplexer to either discard or forward the packet for further processing. The HT is embedded in the MAC layer of the wireless hub, as shown in Figure 5. The HT has a path from incoming flits and token control flits. Note that the HT bypasses the multiplexer so that it receives all the flits despite the destination wireless hub address. The HT will take incoming packet pkt_{in}, which can be either a token control packet or a non-token packet, and produce an outgoing pkt_{out} carrying trace information from HT (t^H) as the payload. pkt_{out} is forwarded to the baseline router. We propose two types of HTs (Counter and Counter Table HT). Notably, both HTs send traces to CA in compressed format to reduce communication overhead and hinder attack detection.

Type 1 [Counter HT]: The Counter HT features a 10-bit counter that tracks idle slots, as outlined in Algorithm 1. It primarily counts the idle time slots between two active slots and forwards this information to the node with the CA. Lines 5 to 12 are responsible for counting the number of idle slots. If the current packet is a token packet (line 5) and $isExpecting$ flag is true (line 6), it means that the previous time slot was idle, so we increment the counter by one (line 7).

If the counter reaches its maximum value (line 8), a packet is created addressing to the node with CA (line 9) and the counter is reset (line 10). Frequently, long sequences of idle slots occur. By compressing these data and sending it combined with information about non-idle slots, the HT can significantly reduce communication overhead. The 'else' block (lines 14 to 18) processes a non-idle slot when pkt_{in} is a non-token packet. The source hub ID and the number of flits are extracted from pkt_{in} (lines 14 and 15). A packet is then created, addressed to the node with CA, containing the current counter value, hub ID, and number of flits as payload (line 16). This counter value represents the idle slots since the last non-idle slot. The counter is reset (line 17) and the $isExpecting$ flag is cleared (line 18), marking the end of an idle period.

Example 2: Consider the T-MAC utilization shown in Table I for the packet sequence and parameters from Example 1. The payload sequence (T^H) for traffic trace packets from Type 1 HT is as follows:

Algorithm 1 Functionality of Counter HT

```
1:  $pkt_{in} \leftarrow$ incoming packet    ▷ token or non-token packet
2:  $isExpecting \leftarrow False$
3:  $counter \leftarrow 0$
4:  procedure COLLECTTRAFFICTRACES($pkt_{in}$)
5:      if $pkt_{in}$ is token packet then
6:          if $isExpecting$ is True then
7:              $counter++$
8:              if $isMaxedOut(counter)$ then
9:                  $pkt_{out} \leftarrow createPkt(MAX)$
10:                 $counter \leftarrow 0$
11:         else
12:             $isExpecting \leftarrow True$
13:     else
14:         $cHub \leftarrow pkt_{in}.srcHubID$
15:         $numFlits \leftarrow pkt_{in}.numFlits$
16:         $pkt_{out} \leftarrow createPkt(counter, cHub, numFlits)$
17:         $counter \leftarrow 0$
18:         $isExpecting \leftarrow False$
19:     return $pkt_{out}$
```

$$T^H = [(2,2,1),(1,0,1),(2,3,1),(1,1,5),(6,0,5),(5,2,5)]$$

This sequence represents data for the active time slots; in other words, the six elements correspond to six packets in Example 1. Each element in T^H is a tuple of the form (idle slots, source hub, flit count), where the first value indicates the number of idle slots between this packet and previous packet, the second value is the ID of the source hub, and the third value represents the number of flits in this packet. For example, the 3^{rd} element of the array, representing the trace for P_2, is (2, 3, 1). This indicates two idle slots (2^{nd} entry: (X, 7, 1) and 3^{rd} entry: (X, 8, 1) of 2^{nd} column in Table I) between P_1 and P_2. Furthermore, the source hub index is 3 (WH_3) and the packet contains one flit. ■

Type 2 [Counter Table HT]: The Counter Table HT features an 8-bit table-based counter and a counter table sized $n \times 8$. Unlike Type 1 HT, which tracks total idle slots, this

HT monitors idle slots per hub, providing more granular data. While Type 2 HT incurs more area than Type 1, it reduces communication overhead. Algorithm 2 describes its functionality. Similar to Algorithm 1, lines 5 to 13 detail how idle slots are counted. When pkt_{in} is a token packet (line 5) and the $isExpecting$ flag for $cHub$ is set (line 6), indicating the previous time slot was idle for $cHub$, the counter for $cHub$ increments (line 7). Notably, the *createPkt* functionality differs between the two HT types. For this HT, when the $cHub$ counter reaches its limit, the value along with the hub ID is sent to the CA (line 9). The current transmitting hub ($cHub$) is updated from each token packet to monitor counters (line 13). Lines 15-18 manage the reception of non-token packets: when received, the counter value for $cHub$, Hub ID, and flit count for $cHub$ are packetized (line 16) and sent to the CA.

Example 3: Consider the T-MAC utilization shown in Table I for the packet sequence and parameters from Example 1. The payload sequence (T^H) for traffic trace packets from Type 2 HT is as follows:

$$T^H = [(0,2,1),(1,0,1),(1,3,1),(2,1,5),(2,0,5),(4,2,5)]$$

This sequence represents data for the active time slots; in other words, the six elements correspond to six packets. Each element in T^H is a tuple of the form (idle slots, source hub, flit count), where the first value indicates the number of idle slots between this packet and previous packet from the same hub, the second value is the ID of the source hub, and the third value represents the number of flits in this packet. For example, the 5th element of the array corresponds to the trace for P_4 is (2, 0, 5). The last two values says source hub is WH_0 and there are 5 flits. First value indicates two idle slots (3rd entry: (X, 11, 1) and 4th entry: (X, 20, 1) of 1st row in Table I) spend by WH_0 since it last sent a packet (P_1). ■

Algorithm 2 Functionality of Counter Table HT

```
 1: $pkt_{in}$ ← incoming packet   ▷ token or non-token packet
 2: $isExpecting$ ← [ ] * *False*
 3: $cTable$ ← [ ]
 4: **procedure** COLLECTTRAFFICTRACES($pkt_{in}$)
 5:     **if** $pkt_{in}$ is *token packet* **then**
 6:         **if** $isExpecting[cHub]$ is *True* **then**
 7:             $cTable[cHub]++$
 8:             **if** *isMaxedOut($cTable[cHub]$)* **then**
 9:                 $pkt_{out}$ ← *createPkt(MAX, cHub)*
10:                 $cTable[cHub] \leftarrow 0$
11:         **else**
12:             $isExpecting[cHub]$ ← *True*
13:         $cHub \leftarrow pckt_{in}.srcHubID$
14:     **else**
15:         $numFlits \leftarrow pkt_{in}.numFlits$
16:         $pkt_{out}$ ← *createPkt(cTable[cHub],cHub,numFlits)*
17:         $cTable[cHub] \leftarrow 0$
18:         $isExpecting[cHub]$ ← *False*
19:     **return** $pkt_{out}$
```

HT Trigger Mechanism: Both HT types support three operational states: inactive, wake-up, and active. In the inactive state, it monitors for a trigger from the CA incoming without leaking T-MAC traffic. CA knows the pattern required to activate the HT trigger from design time. CA broadcasts the pattern by utilizing inherent broadcast WiNoC communication. Upon trigger detection, it enters the wake-up state to establish a covert channel with the CA for transmitting traffic traces. First, HT replies to CA with its identity and then CA provides a memory address for the HT to record these traces, simulating legitimate cache coherence traffic. Once the channel is established, the HT enters the active state and begins leaking T-MAC traffic traces, remaining responsive to CA triggers to deactivate trace collection.

B. Colluding Application (CA)

The CA can operate on any node within the MPSoC and does not need to be on the same node as the malicious wireless hub with HT. The CA's main functions are: (1) collecting the trace data stream (T^H) from the HT, processing it, and forwarding it to the RTA, and (2) activating and deactivating the HT. First, let's focus on the first functionality. The HT transmits traffic trace data to the CA as a stream of packets over time, with each packet's payload defined as t_i^H for the i^{th} packet. The payload structure varies between the two types of HT. We denote T^H as the stream received by the CA during the traffic monitoring period. This period lasts as long as the HT is in the active state and is dynamically determined by the time needed to gather traces covering the l-length window used by the RTA. The CA processes each packet payload t_i^H as soon as it received during the monitoring period and updates T_A, a n-length 2-dimensional array representing each wireless hub. Each sub-array corresponds to a hub and contains traffic trace elements for that hub. Each element, $(s,c)_{j,k}$, within these sub-arrays, represents the number of idle time slots (s) between consecutive packets from the same hub and the number of flits (c) in the k^{th} packet, where j is the hub id and k is the packet number. This compressed format of T^A reduces the bandwidth needed to transmit data to the RTA, optimizing for the frequent idle periods in T-MAC traffic.

Example 4: Consider the T^H generated by HT in Example 2 or Example 3, using the packet sequence from Example 1. The value of l is set to 5. The resulting 2D array (T^A) containing traffic traces produced by CA from T^H is as follows:

$$T^A = [[(1,1),(2,5)],[(2,5)],[(0,1),(4,5)],[(1,1)]]$$

T^A has a length of 4, corresponding to 4 wireless hubs ($n = 4$). The 3rd entry indicates that WH_2 has sent 2 packets (P_0 and P_5). The entry (0,1) corresponds to first packet (P_0) shows one flit in and no idle slots before it, and the second entry (4,5) corresponds to second packet (P_5) from WH_2 with five flits and four idle slots in WH_2 between P_0 and P_5. ■

The CA activates and deactivates the hardware Trojan by sending pre-established patterns to the wireless hub with HT, activating it only when necessary to collect T-MAC traces.

The initial activation message is sent as a broadcast message since only one wireless hub is malicious. Deactivation occurs once sufficient traces are gathered. This controlled minimal activation helps prevent HT detection by runtime monitoring.

C. Remote Traffic Analyzer (RTA)

The RTA's main component is a pre-trained DNN model, trained offline during the attack's design phase, as discussed in Section VII. At the time of the attack, the RTA uses this model to identify the application running in the MPSoC based on T-MAC traffic traces. Initially, RTA converts the incoming T^A from the CA into a spatiotemporal matrix T^M by unpacking T^A's compressed data into a format that uses 0s to represent idle slots. T^M is a wide matrix with dimensions $n \times l$, where n and l correspond to the number of hubs and the window width in time slots, respectively.

Example 5: Consider the T^A generated by CA in Example 4 for the original packet sequence in Example 1. The unpacked traffic trace matrix (T^M) :

$$T^M = \begin{bmatrix} 0 & 1 & 0 & 0 & 5 \\ 0 & 0 & 5 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

T^M is of size 4×5 since there are 4 hubs and $l = 5$. Let's focus on first row (0 1 0 0 5), which corresponds to traces in WH_0. Slots 0, 2, and 3 in WH_0 are idle, while slots 1 and 4 use 1 and 5 cycles for data transfer of packets P_1 and P_4, respectively. In other words, first row of T^M is the expanded version of first element ([(1,1),(2,5)]) of T^A in Example 4. The T^M matrix resembles the original occupancy Table I in terms of rows and columns. Note that the P_5 in time slot 6 is emitted from matrix because $l = 5$. ■

Selection of DNN Model: Selecting the right model for an ML task is crucial. We considered the characteristics of the input data and model overhead for the model selection. The matrix T^M features both temporal (row-wise) and spatial (column-wise) aspects, making RNNs (e.g. GRUs and LSTMs), CNNs, hybrid models, and transformers viable options. RNNs excel in processing temporal patterns but struggle with scalability and have longer training and inference times. CNN, in contrast, is faster on GPU accelerators and effectively manages spatial features. Despite transformers' robust capabilities, their complexity and overhead make them less suitable for our need of a lightweight model. CNN is the preferred choice since our goal is to enable quick and accurate detection of the application running in the system. Although popular for image analysis, CNNs have also shown effectiveness with multivariate time series data [33]. Our trace traffic matrix (T^M) is a multivariate time series, which spans multiple hubs over time.

Defining DNN Architecture: Convolutional Neural Networks (CNNs) use layers of sliding convolutional (conv) filters to detect features in input data. For any conv layer x, the hyperparameters include H_x and W_x representing the height and width of the filters. SW_x and SH_x represent stride width and height, indicating the filters' step size across the input, and K_x represents the number of filters. State-of-the-art CNN architectures [34], [35] use square filters for square image inputs, making them unsuitable for the wide matrix in our case ($n << l$). To address this, we designed two basic CNN models with wide filters: a 2-layer CNN (2-CNN) and a 4-layer CNN (4-CNN). We further optimized these models for our task through extensive hyperparameter tuning.

Both models share a common architecture beyond the conv layers. They incorporate dropout layers with a rate of 0.2 after the last conv layer to prevent overfitting. After conv layers, there are four fully connected layers with 3000, 800, 100, and N neurons, where N denotes the number of output classes corresponding to different applications in the suite. The ReLU activation function is used in all layers except for the output layer, which employs a softmax activation. The 2-CNN aims to capture local spatiotemporal features by the first conv layer and more global features by the latter. All five parameters (H_x, W_x, SH_x SW_x, and K_x) are kept as hyperparameters for both conv layers. Each convolutional layer is followed by a max pooling layer with a (1,2) filter and stride of the same size. The 4-CNN model uses its first two layers with smaller filters and strides to capture local spatiotemporal features, while the last two layers are aimed at broader global features. Parameters for the first two layers ($x = 0, 1$) are set as $H_x = 5, W_x = 7, SH_x = 1, SW_x = 1$, with filter counts $K_0 = 32$ and $K_1 = 64$. The latter layers ($x = 2, 3$) have fixed filter counts $K_2 = 128$ and $K_3 = 256$, but their other parameters (H_x, W_x, SH_x, SW_x) are adjustable. Pooling of size (1,2) and stride follows every two conv layers. Training and hyper-parameter tuning details are outlined in Section VII. The results show that the model performs traffic analysis attacks with high accuracy.

VI. Virtual MAC based Countermeasure

This section introduces virtual MAC (VMAC), a lightweight countermeasure designed to protect WiNoC from traffic analysis attacks. The key innovation of VMAC is the implementation of n_v virtual MAC modules per wireless hub, each with a unique virtual hub ID (v-hub ID) across the WiNoC. This results in WiNoC perceiving $n \times n_v$ virtual hubs instead of n actual hubs. Therefore, the T-MAC token circulates between virtual hubs instead of physical hubs. We ensure the first v-hub ID matches the actual hub ID for each hub, maintaining coherent transmissions across the network and allowing other hubs to send packets addressing the actual hub IDs. Figure 7 illustrates implementation of VMAC by modifying transmission (TX) section of the MAC layer in original wireless hub.

VMAC features a v-hub ID table sized at $n_v \times \log(n \times n_v)$ bits for storing identifiers. The original comparator, which matches the next token owner from the T-MAC token control packet with the hub ID, is replaced by an array that checks the next hub ID against all v-hub IDs, including the actual hub ID. If there's a match, it activates the PHY layer's TX section for packet transmission. Our defense dynamically updates v-hub IDs, managed and securely distributed by the OS or firmware

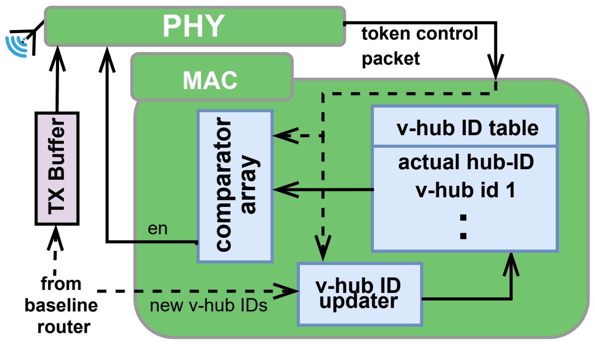

Fig. 7: Structure of updated MAC module with the countermeasure (VMAC). VMAC introduce virtual MACs in the TX layer to dissect and obscure T-MAC traffic traces.

from node to hub. A v-hub updater table, mirroring the main ID table, synchronizes these updates, storing new IDs from the OS and updating the main table upon receiving a control packet through the WiNoC. This ensures synchronized ID updates across all hubs.

Example 6: Consider packet sequence in Example 1 with two distinct VMAC mappings ($maps$): $map1$ maps actual hub IDs (0, 1, 2, 3) to (4, 5, 6, 7), and $map2$ maps them to (7, 4, 6, 5). In both cases, the first four v-hub IDs remain as actual IDs (0-3), creating a total of 8 v-hub IDs. The unpacked traffic trace matrices by adversary are different for the two mappings:

$$T^M_{map1} = \begin{bmatrix} 0 & 0 & 5 & 0 & 0 \\ 0 & 5 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 5 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix} \quad T^M_{map2} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 5 & 5 & 0 & 0 \\ 1 & 5 & 0 & 0 & 0 \end{bmatrix}$$

Both matrices are 8×5 as there are 8 total hubs and $l = 5$. Similar to T^M in Example 5, each row represent traces for virtual-hub. For example, 5^{th} row (0 1 0 0 0) of T^M_{map2} corresponds to WH_4, and it says only slot 1 of WH_4 is used and rest are idle. The two mappings result in two different trace patterns. For example, P_2 arrives to WH_3 in 7^{th} cycle (Example 1). In $map2$, the assignment of v-hub IDs results in packet P_2 being sent in the second time slot because when hub WH_5 (virtual hub of WH_3) gets token at cycle 6 and P_2 is not arrived yet. Whereas in $map1$, it is sent in the first slot as hub WH_7 (virtual hub of WH_3) receives token at cycle 9 and P_2 is already arrived. This reassignment also shifts the scheduling of P_3 to *WH*$_6$ from *WH*$_2$ in $map1$. ■

Security Analysis: Random arbitration in a MAC protocol without the proposed countermeasure would still allow an adversary to uniquely identify traffic traces using the token packet and analyze them to reconstruct the trace, as the adversary does not need the true identity of a hub but only requires a pseudo-identity. In contrast, our proposed countermeasure with virtual hubs significantly complicates the adversary's task. The NoC decouples communication from computation, meaning the malicious hub can only observe communication patterns, not directly link them to computations. There are two primary ways an attacker can try to exploit the system. (1) The attacker can attempt to reconstruct full traces by combining multiple virtual MAC traces. Since communication patterns are split across multiple virtual hubs and mixed with other communication splits, it becomes considerably more difficult for an attacker to group the correct hubs and reconstruct a complete trace. For example, if two virtual hub per hub is used, there are 8128 ways to group two form 128 virtual hubs. The frequent v-hub id updates makes it much harder for adversaries to succeed. (2) The attacker can analyze all the virtual hub traces using a similar model (with increased input size to facilitate virtual hubs) to attempt to identify patterns and eventually identify applications running. This is evaluated in the experiments section and shows traffic analysis attack significantly hinders its ability to identify applications.

Overhead Analysis: The effectiveness of this countermeasure is evident in our results, which show that traffic analysis attacks fail to accurately identify specific applications due to the random dispersion of traffic across multiple virtual hubs. It is important to note that this countermeasure does not introduce randomness explicitly. Instead, it leverages the inherent randomness of network traffic, which results in randomness of division of traffic across virtual MACs of a hub. Most importantly, there is no degradation in performance, as the original time slot allocation is scaled proportionally to accommodate the increased number of v-hub IDs. In other words, without the countermeasure, each hub gets one time slot for every n T-MAC time slots, while with the countermeasure, each virtual hub gets n_v time slots for every $n \times n_v$ time slots. The virtual hub ID update process runs in the background without incurring performance overhead. Since VMAC control packets and ID updates are infrequent, their communication overhead is negligible. This efficiency makes the countermeasure ideal for resource-constrained WiNoCs.

VII. Experiments

In this section, we first describe the experimental setup. Next, we present the results for traffic analysis attack. Finally, we demonstrate the effectiveness of the countermeasure.

A. Experimental Setup

The WiNoC experiments with T-MAC protocol were conducted in the cycle-accurate Noxim simulator [36].We simulated a 64-node MPSoC and evaluated attack and countermeasure effectiveness across three topologies:

- **topo 1**: a hybrid mesh topology with 16 wireless hubs [15] (for example, Figure 2 shows a small-scale hybrid topology with 4 wireless hubs).
- **topo 2**: a Small-World topology comprising 8 subnets, each featuring a single wireless hub [14], [37]
- **topo 3**: a fully wireless topology with 64 wireless hubs [38] (for example, Figure 1 shows a small-scale fully wireless topology with 16 wireless hubs).

We used seven benchmarks from SPLASH-2 [26] and PARSEC [27]: *fft*, *fmm*, *lu*, *barnes*, *radix*, *blacksholes*, and *ocean*. Full system simulations were performed on the gem5 [39] cycle-accurate simulator to collect traces. The traces were used to do a trace-based simulation in Noxim. Details of system configurations used in simulations are outlined in Table II.

TABLE II: System and interconnect configuration.

Parameter	**Details**
Processor configurations	X86, 2GHz
Coherency Protocol	MESI two-level
L1 instruction & data cache	32KB, 32KB (private)
L2 cache	512KB (shared)
Routing protocol	XY

B. CNN Training and Hyperparameter Tuning

To develop our CNN models, we created a comprehensive dataset for each network topology by varying the application mapping across benchmarks. We collected ten distinct feature matrices (T^Ms) for each benchmark-mapping combination, by monitoring the system over an extended period and dividing the T-MAC traffic traces into l-length time slots. This method ensures that different phases of the same benchmark are represented, enhancing the dataset's generalizability. The baseline dataset comprises 4480 feature matrices (T^Ms) across seven classes. We modeled other applications (out of the target application suite) running on the system as noise. We determined the base noise rate (bnr) by averaging the number of packets injected to WiNoC by each benchmark during l-length windows and used various multiples of bnr in our experiments. For example, a $2\times$ noise rate implies injecting packets equivalent to twice bnr, simulating two additional active applications. Unless specified, we consider $2\times$ as the default noise ratio for all experiments.

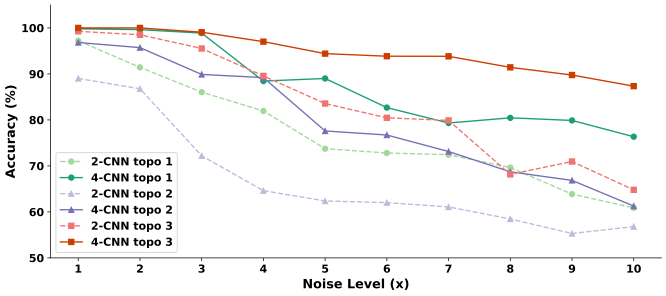

Fig. 8: Accuracy of 2-CNN and 4-CNN models for increasing noise on three topologies.

We created a noise-enhanced dataset by adding noise-augmented data of the same size as the base dataset, resulting in 8960 feature matrices (T^Ms) for seven classes. This will help the model to learn the underlying traffic features as well as the noise parameters better. We partitioned the dataset into training, testing, and validation sets, allocating 88%, 6%, and 6% of the data to each respective split. For the noise-inclusive datasets, we ensured that the test and validation sets mimicked real-world conditions by including only the feature matrices with noise. We conducted rigorous hyperparameter tuning via grid search for all three topologies and both model types (2-CNN and 4-CNN). Chosen hyperparameter spaces and best values for 4-CNN on 'topo 1' traffic are listed in Tables III. We have performed similar tuning for 'topo 2' and 'topo 3'. Here, lr and bs are the learning rate and batch size respectively. For all other experiments, we fixed $lr = 0.005$ and $bs = 64$.

TABLE III: Hyperparameter tuning for 4-CNN on topo 1.

Hypeparam	**value**		**space**	
	x=2	x=3	x=2	x=3
H_x	2	1	1, 2, 4	1, 2, 4
W_x	20	50	10, 20, 25, 50	25, 50, 100
SH_x	1	1	1,2	1, 2
SW_x	10	20	5, 10	10, 20, 50
lr	0.0001		0.001, 0.005, 0.0001	
bs	64		8, 16, 32, 64	

Table IV summarises the best parameters for 2-CNN and 4-CNN across the three topologies with testing accuracy. For all experiments, we employed the Adam optimizer and conducted training over 25 epochs, integrating an early stopping mechanism with a patience setting of five. We set $l = 4000$ for topo 1 and 2, and $l = 4000$ for topo 3, determined based on training curve observations of CNNs. These values are significantly lower than the required time slots for application completion.

TABLE IV: Best 2- and 4-CNN models across 3 topologies.

	Topology	**Best Model**	**Accuracy**
2-CNN	topo 1	(8, 10, 4, 2, 100, 2, 50, 1, 25, 500)*	91.43%
	topo 2	(8, 5, 8, 1, 100, 1, 50, 1, 25, 500)*	86.78%
	topo 3	(32, 5, 16, 1, 100, 2, 100, 1, 50, 500)*	98.51%
4-CNN	topo 1	(2, 20, 1, 10, 1, 50, 1, 20)**	99.63%
	topo 2	(2, 50, 1, 10, 1, 100, 1, 10)**	95.72%
	topo 3	(8, 10, 4, 5, 4, 25, 2, 10)**	100%

*$(H_0, W_0, SH_0, SW_0, K_0, H_1, W_1, SH_1, SW_1, K_1)$,
**$(H_2, W_2, SH_2, SW_2, H_3, W_3, SH_3, SW_3)$

C. Performance of the Traffic Analysis Attack

Table V presents the accuracy, precision, recall, and F1 score for the 2-CNN and 4-CNN models across three topologies. Note that these experiments use default parameters mentioned in the previous section including $2\times$ noise rate. All performance metrics indicate a high likelihood of a successful adversary attack in identifying applications through T-MAC traffic on WiNoC-based SoCs. The 4-CNN, with its additional convolutional layers, consistently demonstrates superior performance over all topologies, reflecting its enhanced capability to approximate traffic patterns. The fully wireless topology (topo 3) yields the highest performance, attributed to the adversary's full visibility of application traffic. In contrast, topo 2 exhibits relatively lower performance due to its limited exposure to application traffic, as relatively large proportion of the communication occurs solely over wired connections. Even though, its accuracy remains high at 95.72%, highlighting the potential for an adversary to gain substantial insights even with partial wireless traffic observation. The effectiveness of the attack on hybrid WiNoC topologies (topo 1, topo 2) highlights that a portion of T-MAC traffic can be sufficient for analysis.

TABLE V: Accuracy, precision, recall, and F1 score for 2-CNN and 4-CNN attacks across three topologies.

	2-CNN				**4-CNN**			
	accuracy	*precision*	*recall*	*F1-score*	*accuracy*	*precision*	*recall*	*F1-score*
topo 1	91.43%	91.98%	91.43%	91.49%	99.63%	99.64%	99.63%	99.63%
topo 2	86.78%	87.69%	86.78%	86.94%	95.72%	95.76%	95.72%	95.71%
topo 3	98.51%	98.53%	98.51%	98.51%	100%	100%	100%	100%

TABLE VI: Accuracy, precision, recall, and F1 score for the 4-CNN attack across three topologies with VMAC.

	2 virtual hubs per hub ($n_v = 2$)				4 virtual hubs per hub ($n_v = 4$)			
	accuracy	*precision*	*recall*	*F1-score*	*accuracy*	*precision*	*recall*	*F1-score*
topo 1	85.47%	86.41%	85.47%	85.61%	13.59%	1.85%	13.59%	3.25%
topo 2	68.16%	72.51%	68.16%	68.78%	14.71%	2.16%	14.71%	3.77%
topo 3	89.76%	90.03%	89.76%	89.78%	58.47%	67.25%	58.47%	60.39%

D. Robustness of the Traffic Analysis Attack

We conduct a robustness analysis of our attack through experiments conducted at different noise levels. Figure 8 shows the accuracy of the 4-CNN and 2-CNN models in all three topologies for increasing noise rates up to 10X. Note that, 10X means ten other applications are running in MPSoC in addition to the target application. In lower noise levels, 2-CNN and 4-CNN models show relatively good performance. However, 4-CNN tends to perform significantly better compared to 2-CNN in higher noise levels, highlighting its robustness. The figure shows that even with high noise rates, the fully wireless topology identifies applications with high accuracy (≈ 90).

E. Performance of the Countermeasure

Table VI presents the effectiveness of proposed countermeasure (VMAC) against the 4-CNN-based attack across three topologies. This table focuses exclusively on the 4-CNN attack due to its high efficacy. We explore two VMAC configurations: (1) employing two virtual hubs per actual hub ($n_v = 2$), and (2) using four virtual hubs per hub ($n_v = 4$). While the deployment of two virtual hubs somewhat diminishes the attack's effectiveness, it appears insufficient. Specifically, for topo 2 with eight hubs, two virtual hubs provide adequate protection. However, implementing four virtual hubs significantly reduces the attack's ability to identify applications, making this a more effective strategy across most topologies. Note that in the fully wireless topology, although the countermeasure achieves an accuracy of approximately 60%, this represents a substantial reduction from an original accuracy of 100%, highlighting its significant impact. Therefore, we recommend adopting four virtual hubs or more to enhance security effectively.

F. Overhead of Hardware Trojans and Countermeasure

Table VII shows the area and power overhead of implementing both the hardware Trojans (HTs) for the attack and the VMAC for the countermeasure. The implementation details of HT and VMAC are discussed in Section V-A and VI, respectively. We synthesized HTs and VMAC for 'topo 1' with 64 nodes and 16 wireless hubs (WHs) using 65nm technology using the Synopsys Design Compiler. When we focus on HTs, Type 2 HT incurs more overhead than Type 1 HT due to its use of a counter table instead of a single counter. Both HTs incur minimal overhead relative to the WH, and even less compared to the overall NoC, as the attack requires only one malicious hub with an HT. For example, Type 2 HT increases area by 1.77% relative to the WH and just 0.043% relative to the NoC. The negligible area and power overhead of HTs makes their detection challenging. In our VMAC implementation, we used four virtual hubs, resulting in a v-hub table of 4×6 bits and an array of four 6-bit comparators. Table VII highlights that area and power overhead of the VMAC is negligible when compared to overall NoC. Our experiments also confirmed that VMAC has zero performance overhead, highlighting it as an effective countermeasure without compromising efficiency.

TABLE VII: Area and power overhead of Hardware Trojan (HT) for traffic analysis attack and proposed countermeasure (VMAC).

	Area			**Power**		
	mm^2	% increase		(μW)	% increase	
		w.r.t WH	w.r.t NoC		w.r.t WH	w.r.t NoC
Type 1 HT	0.001	0.28%	0.007%	4.9	0.013%	0.001%
Type 2 HT	0.008	1.77%	0.043%	26.6	0.071%	0.004%
VMAC	0.025	0.56%	0.22%	269.5	0.025%	0.024%

VIII. CONCLUSION

Wireless Network-on-Chip (WiNoC) represents a vital advancement in communication technologies for System-on-Chips (SoCs). While WiNoCs offers significant benefits compared to electrical NoCs, the shared and broadcast nature of wireless communication also introduces unique vulnerabilities. In this paper, we demonstrated how these vulnerabilities can be exploited using the traffic patterns of the medium access control (MAC) protocol to identify applications running on the system, thereby raising substantial security and privacy concerns from both SoC design and user perspectives. Our attack methodology proved effective across various WiNoC topologies, consistently recovering application identities.We proposed a lightweight countermeasure using virtual wireless hubs to defend against traffic analysis attacks, with experiments showing its effectiveness and negligible hardware overhead. Future SoC designs with WiNoC should consider obscuring information leakage from the shared wireless medium while carefully balancing security and hardware overhead.

References

[1] Ampere, "Ampere Altra Max 64-Bit Multi-Core Processor," https://amperecomputing.com/processors/ampere-altra/, 2022, [Online].

[2] M. S. Lundstrom and M. A. Alam, "Moore's law: the journey ahead," *Science*, vol. 378, no. 6621, pp. 722–723, 2022.

[3] Intel, "Intel® xeon® processor scalable family technical overview," https://www.intel.com/content/www/us/en/developer/articles/technical/xeon-processor-scalable-family-technical-overview.html, 2022, [Online].

[4] K. Chang, S. Deb, A. Ganguly, X. Yu, S. P. Sah, P. P. Pande, B. Belzer, and D. Heo, "Performance evaluation and design trade-offs for wireless network-on-chip architectures," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 8, no. 3, pp. 1–25, 2012.

[5] F. Farahmandi, Y. Huang, and P. Mishra, "System-on-chip security," *Cham, Switzerland: Springer*, 2020.

[6] Semiconductor Research Corporation (SRC), "Microelectronics and advanced packaging technologies (mpat) roadmap," https://srcmapt.org/, 2023, [Online].

[7] H. Weerasena and P. Mishra, "Security of electrical, optical, and wireless on-chip interconnects: A survey," *ACM Transactions on Design Automation of Electronic Systems*, vol. 29, no. 2, pp. 1–41, 2024.

[8] F. Pereñíguez-García and J. L. Abellán, "Secure communications in wireless network-on-chips," in *Proceedings of the 2nd International Workshop on Advanced Interconnect Solutions and Technologies for Emerging Computing Systems*, 2017, pp. 27–32.

[9] B. Lebiednik, S. Abadal, H. Kwon, and T. Krishna, "Architecting a secure wireless network-on-chip," in *2018 Twelfth IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*. IEEE, 2018, pp. 1–8.

[10] A. Vashist, A. Keats, S. M. P. Dinakarrao, and A. Ganguly, "Securing a wireless network-on-chip against jamming based denial-of-service attacks," in *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2019, pp. 320–325.

[11] M. M. Ahmed, A. Ganguly, A. Vashist, and S. M. P. Dinakarrao, "Aware-wi: A jamming-aware reconfigurable wireless interconnection using adversarial learning for multichip systems," *Sustainable Computing: Informatics and Systems*, vol. 29, p. 100470, 2021.

[12] A. Ganguly, M. Y. Ahmed, and A. Vidapalapati, "A denial-of-service resilient wireless noc architecture," in *Proceedings of the great lakes symposium on VLSI*, 2012, pp. 259–262.

[13] H. Weerasena, S. Charles, and P. Mishra, "Lightweight encryption using chaffing and winnowing with all-or-nothing transform for network-on-chip architectures," in *2021 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2021, pp. 170–180.

[14] S. Deb, A. Ganguly, P. P. Pande, B. Belzer, and D. Heo, "Wireless noc as interconnection backbone for multicore chips: Promises and challenges," *IEEE Journal on emerging and selected topics in circuits and systems*, vol. 2, no. 2, pp. 228–239, 2012.

[15] S. S. Rout, M. Sinha, and S. Deb, "2dmac: A sustainable and efficient medium access control mechanism for future wireless nocs," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 19, no. 3, pp. 1–25, 2023.

[16] M. Palesi, M. Collotta, A. Mineo, and V. Catania, "An efficient radio access control mechanism for wireless network-on-chip architectures," *Journal of Low Power Electronics and Applications*, vol. 5, no. 2, pp. 38–56, 2015.

[17] S. Jog, Z. Liu, A. Franques, V. Fernando, S. Abadal, J. Torrellas, and H. Hassanieh, "One protocol to rule them all: Wireless {Network-on-Chip} using deep reinforcement learning," in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, 2021, pp. 973–989.

[18] H. Weerasena and P. Mishra, "Breaking on-chip communication anonymity using flow correlation attacks," *ACM Journal on Emerging Technologies in Computing Systems*, 2023.

[19] V. Y. Raparti and S. Pasricha, "Lightweight mitigation of hardware trojan attacks in noc-based manycore computing," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2019, pp. 1–6.

[20] M. Hussain, A. Malekpour, H. Guo, and S. Parameswaran, "Eetd: An energy efficient design for runtime hardware trojan detection in untrusted network-on-chip," in *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2018, pp. 345–350.

[21] H. Weerasena and P. Mishra, "Lightweight multicast authentication in noc-based socs," in *2024 25th International Symposium on Quality Electronic Design (ISQED)*. IEEE, 2024, pp. 1–8.

[22] H. Weerasena and P. Mishra, "Revealing cnn architectures via side-channel analysis in dataflow-based inference accelerators," *ACM Transactions on Embedded Computing Systems*, vol. 23, no. 6, pp. 1–25, 2024.

[23] M. M. Ahmed, A. Dhavlle, N. Mansoor, P. Sutradhar, S. M. P. Dinakarrao, K. Basu, and A. Ganguly, "Defense against on-chip trojans enabling traffic analysis attacks," in *2020 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*. IEEE, 2020, pp. 1–6.

[24] A. Dhavlle, M. M. Ahmed, N. Mansoor, K. Basu, A. Ganguly, and S. M. PD, "Defense against on-chip trojans enabling traffic analysis attacks based on machine learning and data augmentation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2023.

[25] S. S. Rout, A. Singh, S. B. Patil, M. Sinha, and S. Deb, "Security threats in channel access mechanism of wireless noc and efficient countermeasures," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2020, pp. 1–5.

[26] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The splash-2 programs: Characterization and methodological considerations," *ACM SIGARCH computer architecture news*, vol. 23, no. 2, pp. 24–36, 1995.

[27] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: Characterization and architectural implications," in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, 2008, pp. 72–81.

[28] A. Panchenko, F. Lanze, J. Pennekamp, T. Engel, A. Zinnen, M. Henze, and K. Wehrle, "Website fingerprinting at internet scale." in *NDSS*, 2016.

[29] J. Hayes and G. Danezis, "k-fingerprinting: A robust scalable website fingerprinting technique," in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 1187–1203.

[30] J. Daemen, "Aes proposal: Rijndael," 1999.

[31] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[32] D. J. Bernstein *et al.*, "Chacha, a variant of salsa20," in *Workshop record of SASC*, vol. 8, no. 1. Citeseer, 2008, pp. 3–5.

[33] B. Zhao *et al.*, "Convolutional neural networks for time series classification," *Journal of Systems Engineering and Electronics*, vol. 28, no. 1, pp. 162–169, 2017.

[34] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[35] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[36] V. Catania, A. Mineo, S. Monteleone, M. Palesi, and D. Patti, "Cycle-accurate network on chip simulation with noxim," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 27, no. 1, pp. 1–25, 2016.

[37] S. Wang and T. Jin, "Wireless network-on-chip: a survey," *The Journal of Engineering*, vol. 2014, no. 3, pp. 98–104, 2014.

[38] S. Abadal, A. Cabellos-Aparicio, E. Alarcon, and J. Torrellas, "Wisync: An architecture for fast synchronization through on-chip wireless communication," *ACM SIGPLAN Notices*, vol. 51, no. 4, pp. 3–17, 2016.

[39] N. Binkert *et al.*, "The gem5 simulator," *SIGARCH Computer Architecture News*, 2011.