Real-time Detection and Localization of Distributed DoS Attacks in NoC based SoCs

Subodha Charles, *Member, IEEE,* Yangdi Lyu, *Member, IEEE,* and Prabhat Mishra, *Senior Member, IEEE*

***Abstract*—Network-on-Chip (NoC) is widely employed by multi-core System-on-Chip (SoC) architectures to cater to their communication requirements. Increasing NoC complexity coupled with its widespread usage has made it a focal point of potential security attacks. Distributed Denial-of-Service (DDoS) is one such attack that is caused by malicious intellectual property (IP) cores flooding the network with unnecessary packets causing significant performance degradation through NoC congestion. In this paper, we propose an efficient framework for real-time detection and localization of DDoS attacks. This paper makes three important contributions. We propose a real-time and lightweight DDoS attack detection technique for NoC-based SoCs by monitoring packets to detect any violations. Once a potential attack has been flagged, our approach is also capable of localizing the malicious IPs using the latency data in the NoC routers. The applications are statically profiled during design time to determine communication patterns. These patterns are then used for real-time detection and localization of DDoS attacks. We have evaluated the effectiveness of our approach against different NoC topologies and architecture models using both real benchmarks and synthetic traffic patterns. Our experimental results demonstrate that our proposed approach is capable of real-time detection and localization of DDoS attacks originating from multiple malicious IPs in NoC-based SoCs.**

***Index Terms*—Network-on-chip, Denial-of-service**

I. Introduction

SYSTEM-ON-CHIP (SoC) design using third-party intellectual property (IP) blocks is a common practice today due to both design cost and time-to-market constraints. These third-party IPs, gathered from different companies around the globe, may not be trustworthy. Integrating these untrusted IPs can lead to security threats. A full system diagnosis for potential security breaches may not be possible due to lack of design details shared by the vendors. Even if they do, any malicious modifications (e.g., hardware Trojans) can still go undetected since it is not feasible to exhaustively explore millions of gates and their combinations that can trigger a certain hardware Trojan [1]. The problem gets aggravated due to the presence of Network-on-Chip (NoC) in today's complex and heterogeneous SoCs. Figure 1 shows a typical NoC-based many-core architecture with heterogeneous IPs. As NoC has direct access to all the components in an SoC, malicious third party IPs can leverage the resources provided by the NoC to attack other legitimate components. It can slow down traffic causing performance degradation, steal information, corrupt data, or inject power viruses to physically damage the chip. The problem of NoC security has been explored in two directions: (i) trusted NoC is used to secure the SoC from other untrusted IPs [2], [3], and (ii) NoC is untrustworthy and security countermeasures are required to secure the SoC [4], [5]. Our work is focused on the first scenario where NoC is trustworthy.

S. Charles, Y. Lyu and P. Mishra are with the Department of Computer & Information Science & Engineering, University of Florida, Gainesville, FL, 32611 USA. e-mail: {subodha96, lvyangdi, prabhat}@ufl.edu.

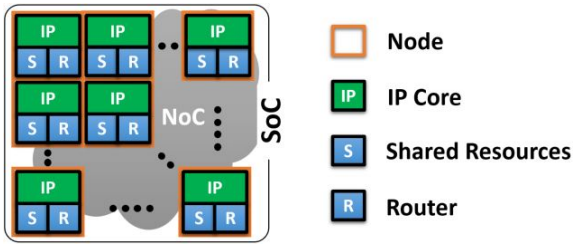


Fig. 1: NoC based many-core architecture connecting heterogeneous IPs on a single SoC. Each IP connects to a router via a network interface. Depending on the selected topology, routers will be arranged across the NoC.

Denial-of-Service (DoS) in a network is an attack preventing legitimate users from accessing services and information. In an NoC setup, DoS attacks can happen from malicious 3rd party IPs (M3PIP) manipulating the availability of on-chip resources by flooding the NoC with packets. The performance of an SoC can heavily depend on few components. For example, a memory intensive application will send many requests to memory controllers, and as a result, routers connected to them will experience heavy traffic [6]. If an M3PIP targets the same node, the SoC performance will suffer significant degradation [4]. Distributed DoS (DDoS) is a type of DoS attack where multiple compromised IPs are used to target one or more components in the SoC causing a DoS attack.

Unlike microcontroller based designs in the past, even resource constrained embedded and IoT (Internet-of-Things) devices nowadays incorporate one or more NoC-based SoCs. Many embedded and IoT systems have to deal with real-time requirements with soft or hard deadlines, where variations in applications as well as usage scenarios (inputs) are either well defined or predictable. In other words, if the applications are not predictable, it is impossible to provide any real-time guarantees. As expected, the communication patterns are known at design time for such systems, which we utilize in this paper. In fact, these assumptions are observed in a wide variety of prior research efforts involving soft [7], [8] as well as hard real-time systems [9], [10]. These embedded and IoT devices can be one of the main targets of DDoS attacks due to their real-time requirements with task deadlines. Early detection of DDoS attacks in such systems is crucial as increased latencies in packet transmission can lead to deadline violations.

Importance of NoC security has led to many prior efforts to mitigate DoS attacks in an NoC such as traffic monitoring [4], [11] and formal verification-based methods [12]. Other real-time traffic monitoring mechanisms have also been discussed

in non-NoC domains [9]. However, none of the existing techniques explored a lightweight and real-time mechanism to detect potential DoS attacks as well as localize the malicious source(s) in an NoC setup. As outlined in Section III-A, it is a major challenge to detect and localize a malicious IP in real-time. The problem is more challenging in the presence of multiple malicious IPs, and it gets further aggravated when multiple attackers help each other to mount the DDoS attack. In this paper, we propose an efficient method that focuses on detecting changes in the communication behavior in real-time to identify DDoS attacks. It is a common practice to encrypt critical data in an NoC packet and leave only few fields as plain text [13][1]. This motivated our approach to monitor communication patterns without analyzing the encrypted contents of the packets. To the best of our knowledge, this is the first attempt to detect and localize DDoS attacks originating from multiple malicious IPs in NoC-based SoCs.

Our major contributions can be summarized as follows;

1) We propose a real-time and lightweight DDoS attack detection technique for NoC-based SoCs. The routers store statically profiled traffic behavior and monitor packets in the NoC to detect any violations in real-time.
2) We have developed a lightweight approach to localize the M3PIP(s) in real-time once an attack is detected.
3) We have evaluated the effectiveness of our approach against different NoC topologies using both real benchmarks and synthetic traffic patterns considering DoS attacks originating from a single malicious IP as well as from multiple malicious IPs.
4) To further evaluate the applicability of our approach, we use an architecture model similar to one of the commercially available SoCs - Intel's KNL architecture [14].

The remainder of the paper is organized as follows. Section II describes related work. Section III discusses the threat model and communication model used in our framework. Section IV describes our real-time attack detection and localization methodology. Section V presents the experimental results. Section VI presents the case study using KNL. Section VII discusses the applicability and limitations of our proposed approach. Finally, Section VIII concludes the paper.

II. Related Work

Countermeasures for DoS attacks both in terms of bandwidth and connectivity have been studied in an NoC context. One such method tries to stop the hardware Trojan which causes the DoS attack from triggering by obfuscating flits through shuffling, inverting and scrambling [12]. If the Trojan gets triggered, there should be a threat detection mechanism. Previous studies explored latency monitoring [4], centralized traffic analysis [11], security verification techniques [12] and design guidelines to reduce performance impacts caused by DoS attacks [15]. In [11], probes attached to the network interface gather NoC traffic data and send it to a central unit for analysis. Such a centralized method can lead to bottlenecks and a single point of failure. Furthermore, the attack can be launched on this central unit itself to impair the security mechanism. In contrast, the method in [4] relies on injecting additional packets to the network and observing their latencies. However, when multiple IPs are communicating with each other, these additional packets can cause congestion and degrade performance as well as introduce overhead.

DoS attacks have been extensively studied in computer networks as well as mobile ad-hoc networks. In the computer network field, DoS attacks can be categorized as brute force attacks and semantic attacks. Brute force attacks overwhelm the system or the targeted resource with a flood of requests similar to our threat model. This can be achieved by techniques such as the attacker sending a large number of ICMP packets to the broadcast address of a network or by launching a DNS amplification attack [16]. It is common to use *botnets* rather than few sources to maximize the impact of the attacks. Semantic attacks on the other hand exploit some artificial limit of the system to deny services. Two popular examples are Ping-of-Death [17] and TCP SYN flooding [18]. Techniques such as botnet fluxing [19], back propagation neural networks [20] and TCP blocking [21] have been used to mitigate these attacks. However, using these techniques in SoC domain is not feasible due to the resource constrained nature and the architectural differences. There are methods to secure IoT devices such as lightweight encryption [22] and authentication [23]. However, these solutions does not address DoS attacks.

Waszecki et al. [9] discussed network traffic monitoring in an automotive architecture by monitoring message streams between electronic control units (ECU) via the controller area network (CAN) bus. Since multiple ECUs are connected on the same bus, it is difficult to localize the origin of attack, and therefore, the authors present the solution only as a detection mechanism. Moreover, this architecture is bus-based and fundamentally different from an NoC. In this paper, we propose a lightweight and real-time mechanism to detect DDoS attacks in an NoC-based SoC. Our proposed approach has the ability to localize any number of malicious IPs. Moreover, the proposed work is applicable on a wide variety of NoC architectures supporting diverse deterministic routing protocols. To the best of our knowledge, this is the first attempt to detect and localize DDoS attacks originating from multiple malicious IPs in NoC-based SoCs.

III. System and threat models

A. Threat Model

Previous works have explored two main types of DDoS attacks on NoCs [24] - (i) M3PIPs flooding the network with useless packets frequently to waste bandwidth and cause a higher communication latency causing saturation, and (ii) draining attack which makes the system execute high-power tasks and causes fast draining of battery. An illustrative example is shown in Figure 2 to demonstrate the first type of DDoS attack. As a result of the injected traffic from the malicious IPs to the victim IP (this can be a critical NoC component such as a memory controller), routers in that area of the NoC get congested and responses experience severe delays.

[1]On-chip encryption schemes introduce the notion of *authenticated encryption with associated data* in which the data is encrypted and associated data (initialization vectors, routing details etc.) are sent as plain-text [13].

A practical example of a draining attack was shown in [25]. A malware known as a worm spread through Bluetooth and multimedia messaging services (MMS) and infected the recipient's mobile phone. The code is crafted in such a way that it sends continuous requests to the Bluetooth module for paging and to scan for devices. Power consumption in the infected phone was increased up to 500% compared to the idle state causing significant degradation of battery lifetime. There are instances of draining attacks where even though the computation overhead increases, the communication traffic does not increase. Such attacks cannot be detected using a security mechanism implemented at the NoC, and therefore, are beyond the scope of this paper.

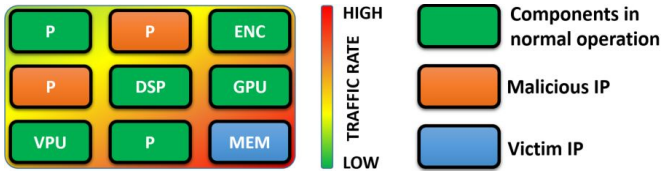


Fig. 2: Example DDoS attack from malicious IPs to a victim IP in a Mesh NoC setup. The thermal map shows high traffic near the victim IP (MEM). P-processor, DSP-digital signal processor, VPU-vector processing unit, GPU-graphics processing unit, ENC-encoder, MEM-memory controller.

Our threat model is generic, it does not make any assumption about the placement or the number of malicious IPs or victim IPs. Figure 3 shows four illustrative examples of malicious/victim IP placements that can lead to different communication patterns. Figure 3(a) shows a scenario involving one malicious IP and one victim IP. The other three examples represent scenarios where the packets injected from the malicious IPs to victim IPs are routed through paths that (b) partially overlap, (c) completely overlap and (d) form a loop. Our proposed approach is capable of both detecting and localizing all the malicious IPs in all these scenarios.

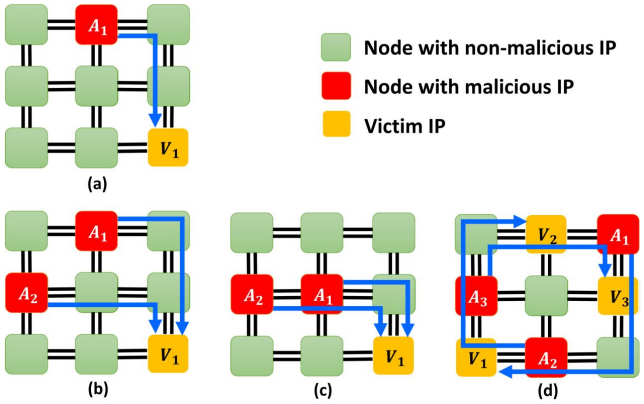


Fig. 3: Different scenarios of malicious and victim IP placement. Packet routing paths from malicious to victim IPs shown in blue. (a) only one attacker is present, (b) paths partially overlap, (c) paths completely overlap, (d) paths form a loop.

B. Communication Model

Since each packet injected in the NoC goes through at least one router, we identify it to be an ideal NoC component for traffic monitoring. The router also has visibility to the packet header information related to routing. Packet arrivals at a router can be viewed as *events* and captured using arrival curves [26]. We denote the set of all packets passing through router r during a program execution as a *packet stream* P_r. Figure 4 shows two packet streams within a specific time interval $[1, 17]$. The stream P_r (blue) shows packet arrivals in normal operation and $\widetilde{P_r}$ (red) depicts a compromised stream with more arrivals within the same time interval. The packet count $N_{p_r}[t_a, t_b)$ gives the number of packets arriving at router r within the half-closed interval $[t_a, t_b)$. Equation 1 formally defines this using $N_{p_r}(t_a)$ and $N_{p_r}(t_b)$ - maximum number of packet arrivals up to time t_a and t_b, respectively. $\forall t_a, t_b \in \mathbb{R}^+, t_a < t_b, n \in \mathbb{N}$:

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

Unlike [9] that monitors message streams at ECUs in a bus-based automotive architecture, our model is designed to monitor packets at routers of NoC-based SoC architectures.

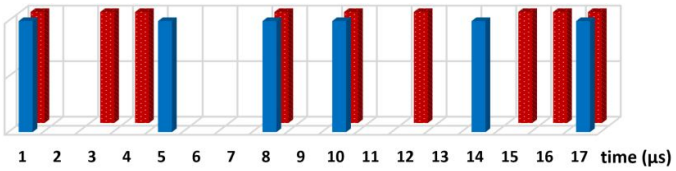


Fig. 4: Example of two event traces. Six blue event arrivals represent an excerpt of a regular packet stream P_r and nine red event arrivals represent a compromised packet stream $\widetilde{P_r}$.

IV. Real-Time Attack Detection and Localization

Figure 5 shows the overview of our proposed security framework to detect and localize DoS attacks originating from one or more M3PIPs. The first stage (upper part of the figure) illustrates the DDoS attack detection phase while the second stage (lower part of the figure) represents the localization of M3PIPs. During the detection phase, the network traffic is statically analyzed and communication patterns are parameterized during design time to obtain the upper bound of *packet arrival curves* (PAC) at each router and *destination packet latency curves* (DLC) at each IP. The PACs are then used to detect violations of communication bounds in real-time. Once a router flags a violation, the IP attached to that router (local IP) takes responsibility of diagnosis. It looks at its corresponding DLC and identifies packets with abnormal latencies. Using the source addresses of those delayed packets, the local IP communicates with routers along that routing path to get their congestion information to localize the M3PIPs. The remainder of this section is organized as follows. The first two sections describe parameterization of PAC and DLC. Section IV-C elaborates the real-time DDoS attack detection mechanism implemented at each router. Section IV-D describes the localization of M3PIPs.

A. Determination of Arrival Curve Bounds

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

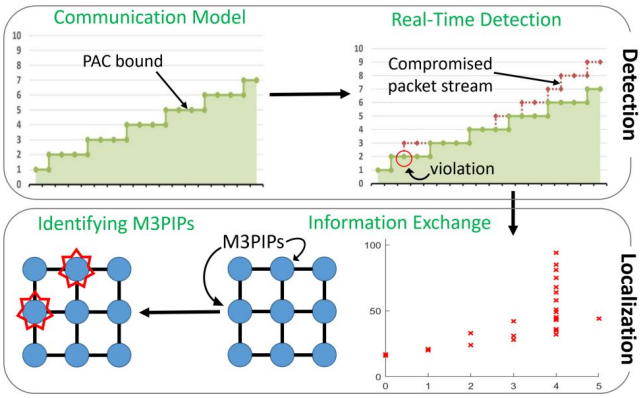


Fig. 5: Overview of our proposed framework: the system specification is analyzed to obtain the necessary packet arrival curves and detection parameters. These are used to design the real-time attack detection and localization framework.

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

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

Repeating this for several fixed Δ, constructs the upper PAC bound. These bounds are represented as *step functions*. A lower PAC bound can also be constructed by recording the minimum number of packets within the sliding window. However, we exclude it from our discussion since in a DoS attack, we are only concerned about violating the upper bound. An example PAC bound and two PACs corresponding to the packet streams in Figure 4 are shown in Figure 6. During normal execution, the PACs should fall within the shaded area.

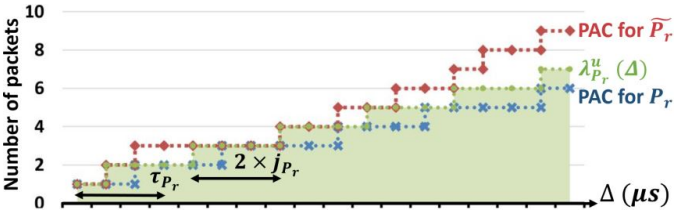
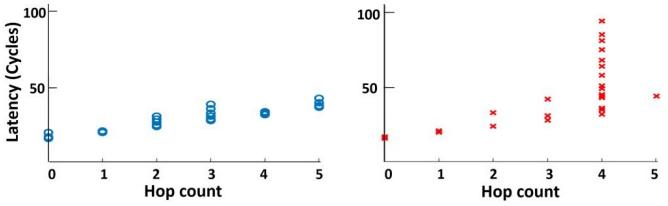


Fig. 6: Graph showing upper ($\lambda^u_{p_r}(\Delta)$) bound of PACs (green line with "●" markers) and the normal operational area shaded in green. The blue and red step functions show PACs corresponding to P_r and $\widetilde{P_r}$, respectively.

While NoCs in general-purpose SoCs may exhibit dynamic and unpredictable packet transmissions, for vast majority of embedded and IoT systems, the variations in applications as well as usage scenarios (inputs) are either well-defined or predictable. Therefore, the network traffic is expected to follow a specific trend for a given SoC. SoCs in such systems allow the reliable construction of PAC bounds during design time. To get a more accurate model, it is necessary to consider delays that can occur due to NoC congestion, task preemption, changes of execution times and other delays. To capture this, we consider the packet streams to be periodic with jitter. The jitter corresponds to the variations of delays. Equation 3 represents the upper PAC bound for a packet stream P_r with maximum possible jitter j_{P_r} and period τ_{P_r} [27].

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



(a) Normal operation (b) Attack scenario

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

The equation captures the shift of the upper PAC bound because of the maximum possible jitter j_{P_r} relative to a nominal period τ_{P_r}. This method of modeling upper PAC bounds is validated by the studies in *modular performance analysis* (MPA) that uses *real-time calculus* (RTC) as the mathematical basis. MPA is widely used to analyze the best and worst case behavior of real-time systems. Capturing packet arrivals as event streams allows the packet arrivals to be abstracted from the time domain and represented in the interval domain (Figure 6) with almost negligible loss in accuracy [27]. The same model is used in the MATLAB RTC toolbox [28].

B. Determination of Destination Latency Curves

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

C. Real-time Detection of DoS Attacks

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

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

An obvious way to detect violations with the upper bound would be to construct the PAC and check if it violates the bound as shown in Figure 6. However, to construct the PAC, the entire packet stream should be observed. In other words, all packet arrivals at a router during the application execution should be recorded to construct the PAC. While it is feasible during upper PAC bound construction at design time, it doesn't lead to a real-time solution. Therefore, we need an efficient method to detect PAC bound violations during runtime.

To facilitate runtime detection of PAC bound violations, we use the *leaky bucket* algorithm, which considers packet arrivals and the history of packet streams and gives a real-time solution [29]. Once $\lambda^u_{p_r}(\Delta)$ is parameterized, the algorithm checks the number of packet arrivals within all time intervals for violations. Algorithm 1 outlines the leaky bucket approach where $\theta_{r,s}$ denotes the minimum time interval between consecutive packets in a staircase function s at router r, and $\omega_{r,s}$ represents the burst capacity or maximum number of packets within interval length zero. $\lambda^u_{p_r}(\Delta)$, which is modeled as a staircase function can be represented by n tuples - ($\theta_{r,s}$, $\omega_{r,s}$), $s \in \{1, n\}$ sorted in ascending order with respect to $\omega_{r,s}$. This assumes that each PAC can be approximated by a minimum on a set of periodic staircase functions [30].

Algorithm 1: Detecting compromised packet streams

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

Lines 1-4 initializes the timers ($\text{TIMER}_{r,s}$) to $\theta_{r,s}$ and packet counters at time zero ($\text{COUNTER}_{r,s}$) to corresponding initial packet numbers $\omega_{r,s}$, for each staircase function and packet stream P_r. The DDoS attack detection process (lines 5-15) basically checks whether the initial packet limits ($\text{COUNTER}_{r,s}$) have been violated. Upon reception of a packet (line 5), the counters are decremented (line 10), and if it falls below zero, a potential attack is flagged (line 12). If the received packet is the first within that time interval (line 7), the corresponding timer is restarted (line 8). This is done to ensure that the violation of PAC upper bound can be captured and visualized by aligning the first packet arrival to the beginning of the PAC bound. When the timer expires, values are changed to match the next time interval (lines 17-20). As demonstrated in Section V, the algorithm allows real-time detection of DDoS attacks under our threat model. Another important observation described in Section V-D1 drastically reduces the complexity of the algorithm allowing a lightweight implementation. The leaky bucket algorithm is originally proposed to check the runtime conformity of event arrivals in the context of network calculus. Its correctness is proven by [31].

D. Real-time Localization of Malicious IPs

Figure 7(b) shows an example DLC during an attack scenario, where all IPs are injecting packets exactly the same way as shown in Figure 7(a) except for one M3PIP, which injects a lot of packets to a node attached to a memory controller. Those two nodes are 4-hops apart in the Mesh topology. This makes the latency for 4-hop packets drastically higher than usual. For every hop count, we maintain the traffic distribution as a normal distribution using $\mu_{i,k}$ and $\sigma_{i,k}$. Once a potential threat is detected at a router, it sends a signal to the local IP. The local IP then looks at its DLC and checks if any of the curves have packets that took more than $\mu_{i,k} + 1.96\sigma_{i,k}$ time (95% confidence level). One simple solution is to examine source addresses of those packets and conclude that the source with most number of packets violating the threshold is the M3PIP. However, this simple solution may lead to many false positives. As each IP is distributed and examines the latency curve independently, the IP found using this method may or may not be a real M3PIP (*attacker*). Therefore, we call it a *candidate M3PIP*.

To illustrate the difference between an attacker and a candidate M3PIP, we first examine four scenarios with only one attacker as shown in Figure 8. In these scenarios, the attacker A is sending heavy traffic to a victim IP V, and as a result, local IP D is experiencing large latency for packets from source S. The first three examples in Figure 8 show examples where candidate M3PIP S is not the real attacker A. Since a large anomalous latency is triggered by the congestion in the network, the only conclusion obtained by the local IP from its DLC is that at least part of the path from candidate M3PIP to local IP is congested. We call the path from attacker A to victim V as the *congested path*.

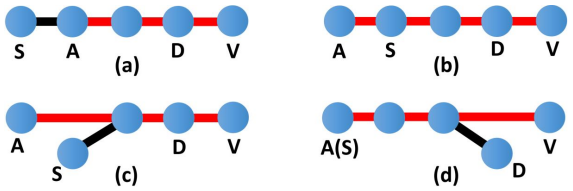


Fig. 8: Four scenarios of the relative positions of local IP (D), attacker IP (A), victim IP (V), and the candidate M3PIP (S) as found by D. The red line represents the congested path.

In Figure 8(a) and Figure 8(c), the false positives of the candidate M3PIP S can be removed with global information of congested paths, by checking the congestion status of path from S to its first hop. It is certain that S is not the attacker when this path is not congested. However, we cannot tell whether S is the attacker when the path of S is congested. For example, the routers of Figures 8(b) and 8(d) are both congested, but S is not the attacker in 8(b).

Things get much worse when multiple attackers are present. If we look at the example in Figure 9, the path from candidate M3PIP S to local IP D is part of all paths along which three

different attackers are sending packets to different victims. We define the *congested graph* as the set of all congested paths and all the routers in the paths. Since each hop connecting two routers consists of two separate uni-directional links, a congested graph is a bi-directional graph as shown in Figure 9. In order to detect attackers and avoid false positives, one simple solution would be building the entire congested graph by exchanging information from all the other routers and analyzing the graph to detect the actual M3PIPs. However, it would add a lot of burden on the already congested paths.

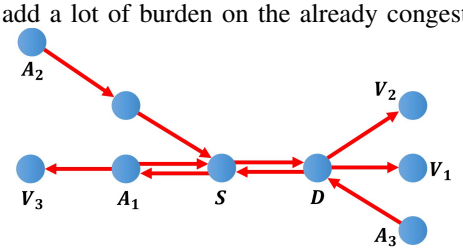


Fig. 9: Congested graph of three attackers.

To overcome the bottlenecks, we propose a distributed and lightweight protocol implemented on the routers to detect the attackers. The event handler for each router for M3PIP localization is shown in Algorithm 2. The description of the steps of our complete protocol are shown below:

1) The router R detects an ongoing attack and sends a signal to the local IP (line 4). In Figure 8, both D and V will send a signal to their local IPs.
2) The local IP D looks at its DLC and responds to its router with a *diagnostic message* $< S, D >$ indicating the address of the candidate M3PIP S and destination D. The local router then forwards the packet towards S.
3) Each port in each router maintains a three-state flag to identify the attacker. The flag is 0, 1 and 2 to denote the attacker is undefined, local IP or others, respectively. When a diagnostic message $< S, D >$ comes in, R checks if the candidate M3PIP S is the local IP. If yes and its flag is not set yet, it will set the flag to be 1 (line 9). If S is not the local IP, it first finds out its neighbor N which sits in the path from S to R. If the one-hop path from N to R is congested, it sends the message to N (line 19) and sets the flag to 2, to indicate other IP as a potential attacker (line 20). Except for these two scenarios, the received message is a false positive and no action is taken (line 12 and 23), which will be explained in our examples. Note that the flag cannot decrease except for the reset signal which sets it to undefined (line 2). Therefore, if a diagnostic message already mentioned that other IPs may be the potential attackers, a new diagnostic message from the same port claiming that the local IP is the attacker will be ignored.
4) Each router maintains a timer. The timer starts as soon as any one of the router ports receive a diagnostic message. A pre-defined timeout period is used by each router. If the flag of any port is 1 after timeout, it broadcasts a message alerting that its local IP (line 28) is the attacker. Finally, a reset signal is triggered (line 29).

First, we will show that our approach works when a DoS attack is originating from only one M3PIP in the NoC. Later, we will describe how the proposed approach works in the presence of multiple M3PIPs mounting a DDoS attack.

1) DoS Attack by a Single M3PIP: We use Figure 8(b) to illustrate how our approach will localize the attacker when a DoS is caused by a single M3PIP. The router of S will receive two messages, one from the router of D saying that its local IP is a candidate M3PIP, and the other from the router of V saying that A is a candidate M3PIP, i.e., $< S, D >$ and $< A, V >$. Depending on the arrival time of these two messages, there are two scenarios. (a) $< S, D >$ comes first. It will change the flag of the corresponding port to 1 to denote that the local IP is the potential attacker. Then, S will receive $< A, V >$ through the same port. In this example, A is also the neighbor N. As the one-hop path from A to S is congested, the flag will be set to 2, denoting that the attacker is some other IP. (b) $< A, V >$ comes first. It will change the flag of the corresponding port to 2 to denote that the other IP is the potential attacker. Then, S will receive $< S, D >$ through the same port. As the flag is already set to 2, the received message is a false positive (line 12). When timeout occurs, nothing happens at the router of S. However, the router of A receives only the message from V indicating that its local IP is the potential attacker and its flag remains 1 when timeout occurs. A broadcast is sent indicating that A is the attacker.

Algorithm 2: Event handler for router R

```
1  upon event RESET:
2  R.flag[p_i] = 0 for all ports p_i
3  upon event attacked == TRUE:
4  send a signal to local IP
5  upon receiving a diagnostic message < S, D > from port p_i:
6  start TIMEOUT if all R.flag == 0
7  if S is local IP then
8      if flag[p_i] == 0 then
9          flag[p_i] = 1                    // local IP is the M3PIP
10     end
11     if flag[p_i] == 2 then
12         // false positive, do nothing
13     end
14 end
15 else
16     //S is not local IP
17     Let N be the neighbor of R that sits in the path from S to R
18     if path from N to R is congested then
19         sends a diagnostic message < S, D > to N indicating that S is a
             candidate attacker
20         flag[p_i] = 2                    // other IP is the M3PIP
21     end
22     else
23         // false positive, do nothing
24     end
25 end
26 upon event TIMEOUT:
27 if any flag in R.flag is 1 then
28     broadcasting that its local IP is the attacker
29     RESET
30 end
```

For the case in Figure 8(a), A will receive a message from D indicating that S is a candidate M3PIP. However, when A checks the congestion status of the one-hop path from S to A, it will find out that the path is not congested. Therefore, the message is a false positive (line 23), and A will not change its flag. In other words, the flag of A will be set to 1 after receiving the message from V, and will not be changed by the message from D to S. After timeout, A will be identified as the attacker.

2) DDoS Attack by Multiple M3PIPs: Before giving an illustrative example of how our approach will localize attacks by multiple malicious IPs, we formally prove the correctness of our approach by proving the following theorem.

Theorem 1. *If the congested graph contains no loops, Algorithm 2 can localize at least one attacker.*

Proof. We merge multiple diagnostic messages with the same destination as one message and ignore all false positive messages detected in line 12 and line 23 of Algorithm 2. We define message φ_i as a diagnostic message which points out that A_i is a candidate M3PIP. Consider the port of any attacker A_i that receives message φ_i. Such a port always exists in a DDoS attack scenario due to the fact that victim V_i will send a message φ_i to A_i saying that A_i is a candidate M3PIP. If φ_i is the only message received from this port, our algorithm can declare A_i as an attacker.

Our algorithm fails only when all routers connected to the attackers have flags set to either 0 or 2 in each of their ports as illustrated in Algorithm 2. This can only happen when each port that receives a diagnostic message, receives another diagnostic message which causes the flag to be set to 2. Assume that a port in router of A_i receives messages $MS_i = \{\varphi_i, \varphi_j, ...\}$. It will digest the message φ_i and send out the remaining ones. We will construct a diagnostic message path in the following way. First, we add A_i to the path. Then, we select any message from MS_i other than φ_i, e.g., φ_j. Next, we follow the diagnostic message path from A_i to A_j, and add all routers to the path. By the same process, we select one message other than φ_j from MS_j, e.g., φ_k. Next, we follow the path from A_j to A_k. We can do this one by one since for every message set MS_u at attacker A_u, there is at least one message other than φ_u to select from. Therefore, the constructed diagnostic message path contains an infinite number of attackers, as shown in Figure 10. The infinite number of attackers implies that this path contains repeated attackers. Without loss of generality, we can assume that $A_k = A_i$. Since A_i cannot be sending out diagnostic messages MS_i through the same port that receives MS_i, the diagnostic path must form a loop. It is easy to see that diagnostic paths are the reverse of congested paths. As a result, there exists a loop in the congested graph, which contradicts the assumption made. Hence, Theorem 1 is proven. □

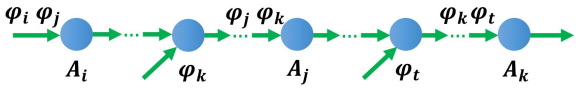


Fig. 10: An example of a diagnostic message path constructed by following the flow of a diagnostic message in each attacker.

Thus, there always exists a port of the router connected to attacker A_i which receives only one diagnostic message φ_i given that there are no loops. This is a sufficient condition to detect A_i using Algorithm 2. Using our approach for localizing multiple malicious IPs gives rise to three cases that behave differently depending on how the M3PIPs are placed.

Case 1: If the congested paths do not overlap, all M3PIPs will be localized in one iteration using the process outlined above. This is the best case scenario for our approach and localizes M3PIPs in minimum time.

Case 2: If at least two paths overlap, it will need more than one iteration to localize all M3PIPs. To explain this scenario, an illustrative example is shown in Figure 11. Figure 11(a) shows the placement of the four M3PIPs (A_1, A_2, A_3, A_4) attacking the victim IP (V). Once the attack is detected, in the first iteration, A_1, A_3 and A_4 are detected as shown in Figure 11(b). Due to the nature of our approach, A_2 is not marked as an attacker. This is caused by two diagnostic messages going in the paths $V \rightarrow A_2$ and $V \rightarrow A_3$. The router of A_2 will receive a message from the router of V saying that its local IP is a candidate M3PIP. It will change the flag of the corresponding port to 1 to denote that A_2 is the potential attacker. A_2 will receive another message from the router of V through the same port saying that A_3 is a candidate M3PIP. In this example, A_3 is also the neighbor of A_2. As the one-hop path from A_3 to A_2 is congested, the flag will be set to 2, denoting that the attacker is some other IP. When timeout occurs, nothing happens at the router of A_2. However, the router of A_3 receives only the message from V indicating that its local IP is the potential attacker and its flag remains 1 when timeout occurs. Therefore, A_3 is detected as an attacker whereas A_2 is not. In the case of A_1 and A_4, there is no overlap of congested paths and the two attackers are detected without any false negatives. Once the system resumes with only A_2 being malicious, the attacker will be detected and localized in the second iteration (Figure 11(c)). This case consumes more time since an additional detection phase is required to localize all M3PIPs. The number of iterations will depend on how many overlapped paths can be resolved at each iteration. In the worst case (where all congested paths can overlap and each iteration will resolve one path), the number of iterations will equal to the number of M3PIPs. However, our approach is guaranteed to localize all M3PIPs.

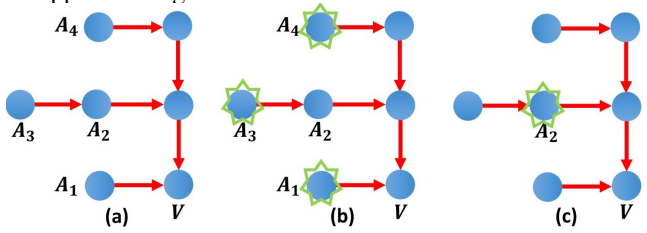


Fig. 11: Illustrative example to show how our detection and localization framework works. (a) Placement of attackers and victim that causes an overlap of congested paths of attackers A_2 and A_3. (b) Attacker(s) detected from first iteration. (c) Attacker(s) detected from second iteration.

Case 3: The proof of Theorem 1 had the assumption that the congested graph contains no loops. Therefore, using our approach as it is, will not lead to localizing all M3PIPs if the congested graph forms a loop as shown in Figure 12. One solution is that any router in the congested loop can randomly "stop working" and resume after a short while. By breaking the loop, our approach will detect attackers with the new congested graph. The router "stopping work" can be triggered by the system observing that a DDoS attack is going on (during the detection phase), but no M3PIPs being localized.

In summary, our approach will detect one or more M3PIPs at each iteration depending on whether congested paths over-

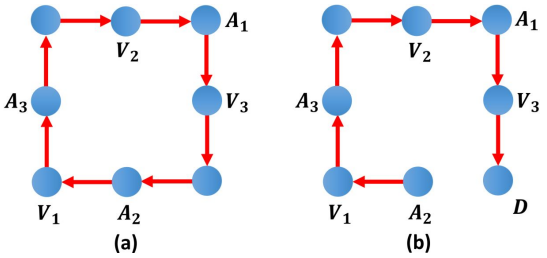


Fig. 12: (a) Three attackers cooperate and construct a loop in the congested graph. Algorithm 2 will fail to detect any attacker in the loop. (b) When a router randomly "stops working", an attacker A_2 is revealed after breaking the loop.

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

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

V. Experiments

We have explored DoS attacks caused by a single M3PIP as well as multiple M3PIPs using the architecture shown in Figure 13. In Section VI, we evaluate the efficiency of our approach in an architecture model similar to one of the commercially available SoCs [14].

A. Experimental Setup

Our approach was evaluated by modeling an NoC-based SoC using the cycle-accurate full-system simulator - gem5 [32]. The interconnection network (NoC) was built on top of the "GARNET2.0" model that is integrated with gem5 [33]. The default gem5 source was modified to include the detection and localization algorithms. We experimented using several **synthetic traffic patterns** (*uniform_random, tornado, bit_complement, bit_reverse, bit_rotation, neighbor, shuffle, transpose*), topologies (*Point2Point (16 IPs), Ring (8 IPs), Mesh4×4, Mesh8×8*) and XY routing protocol to illustrate the efficiency of our approach across different NoC parameters. A total of 40 traffic traces were collected using the simulator by varying the traffic pattern and topology. Synthetic traffic patterns were only tested using one M3PIP in the SoC launching the DoS attack and an application instance running in 50% of the available IPs. These traffic traces act as test cases for our algorithms. The placement of the M3PIP, victim IP and IP(s) running the traffic pattern were chosen at random for the 40 test cases.

Our approach was also evaluated using **real traffic patterns** based on 5 benchmarks (FFT, RADIX, OCEAN, LU, FMM) from the SPLASH-2 benchmark suite [34] in Mesh 4×4 topology. Traffic traces from real traffic patterns were used to test both single-source DoS attacks as well as multiple-source DDoS attacks. The attack was launched at a node connected to a memory controller. Relative placements of the M3PIP and victim IP used to test the single-source DoS attack were the same as for the synthetic traces running on Mesh 4×4 topology (test case IDs 1 through 5 in Figure 15). For the DDoS attack involving multiple M3PIPs, we ran tests using the same set of benchmarks and topology with the victim and M3PIP placements as shown in Figure 13. The placement captures both Case 1 and Case 2 discussed in Section IV-D2. Each node with a non-malicious IP ran an instance of the benchmark while the four nodes in the four corners were connected to memory controllers. The jitter for all applications was calculated using the method proposed in [35].

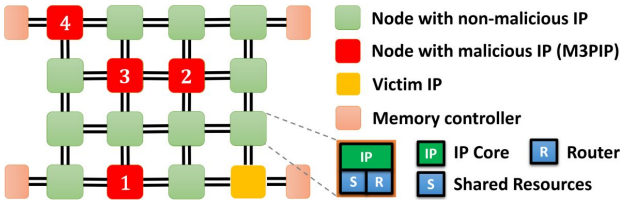


Fig. 13: M3PIP and victim IP placement when running tests with real benchmarks on a 4x4 Mesh NoC.

B. Efficiency of Real-time DoS Attack Detection

Before showing results of our experimental evaluation, we will first give an illustrative example to show how the parameters associated with the leaky bucket algorithm (Algorithm 1) is calculated and used in attack detection.

An important observation allows us to reduce the number of parameters required to model the PACs, and as a result, implement a lightweight scheme with much less overhead. The model in Equation 3 is derived using the fact that the packet streams are periodic with jitter. As proposed in [9] and [36], for message streams with such arrival characteristics, the PACs can be parameterized by using only worst case jitter j_{P_r}, period τ_{P_r} and an additional parameter ϵ_r which denotes the packet counter decrement amount. The relationship between these parameters are derived in [30] as shown in Equation 5.

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

$$\omega_r = 2 \times \epsilon_r - \frac{\tau_{P_r} - j_{P_r}}{\theta_r} \quad (5b)$$

$$\epsilon_r = \frac{\tau_{P_r}}{\theta_r} \quad (5c)$$

To use these parameters, the only changes to Algorithm 1 are at line 10 ($\text{COUNTER}_{r,s} = \text{COUNTER}_{r,s} - \epsilon_r$) and one tuple per packet stream instead of n tuples ($s \in \{1\}$). The illustrative example is based on this observation.

Illustrative Example: Consider the example packet streams shown in Figure 4. Assume that the packet steam P_r has a period $\tau_{P_r} = 3\mu s$ and jitter $j_{P_r} = 1.5\mu s$. During an attack scenario, this stream is changed to stream $\widetilde{P_r}$ with $\tau_{\widetilde{P_r}} = 2\mu s$ and no jitter. Using these values in Equation 5 will give $\theta_r = 1.5\mu s, \omega_r = 3$ and $\epsilon_r = 2$, which are the parameters used in the leaky bucket algorithm. Therefore, $COUNTER_{r,s}$ is initialized with 3 (line 3, line 18) and decremented by 2 at each message arrival (line 10). $TIMER_{r,s}$ is initialized to $1.5\mu s$ (line 2, line 19). Using these values and running the detection algorithm during the attack scenario will lead to a detection time of $4\mu s$. Figure 14 shows the values of the parameters changing with each packet arrival and timeout leading to the detection of the attack at $t = 4\mu s$.

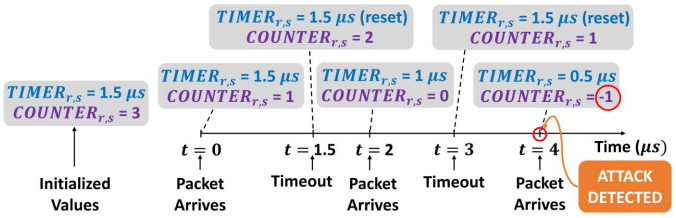


Fig. 14: Illustrative example of parameter changes in the leaky bucket algorithm with packet arrivals and timeouts.

The experimental evaluation follows the same process as the illustrative example using the experimental setup described in Section V-A. Figure 15 shows the detection time across different topologies for synthetic traffic traces in the presence of one M3PIP. The 40 test cases are divided into different topologies, 10 each. The packet stream periods are selected at random to be between 2 and 6 microseconds. Attack periods are set to a random value between 10% and 80% of the packet stream period. The detection time is approximately twice the attack period in all topologies. This is expected according to Algorithm 1 and consistent with the observations in [9].

In addition to the time taken by the leaky bucket approach, the detection time also depends on the topology. For example, attack detection in Point2Point topology (Figure 15(a)), where every node is one hop away, requires less time to detect compared to Mesh8×8 (Figure 15(d)) where some nodes can be multiple hops away. The topology mainly affects attack localization time due to the number of hops from detector to attacker. But for detection, topology plays a relatively minor role since the routers are connected to each IP and detection mechanism neither takes into account the source nor the destination of packets. The routers only look at how many packets arrived in a given time interval. It is also important to note that any router in the congested path can detect the attack, not only the router connected to the victim IP. A combination of these reasons have led to the topology playing a relatively minor role in attack detection time. These results confirm that the proposed approach can detect DoS attacks in real-time.

Results for DDoS attack detection in the presence of multiple attacking M3PIPs are shown in Figure 16 and Figure 17. For all of these experiments, packet stream period is fixed at $2.5\mu s$ and attack period is set to $1.5\mu s$. Figure 16 shows detection time variation in the presence of different number of IPs across benchmarks. The time to detect an ongoing

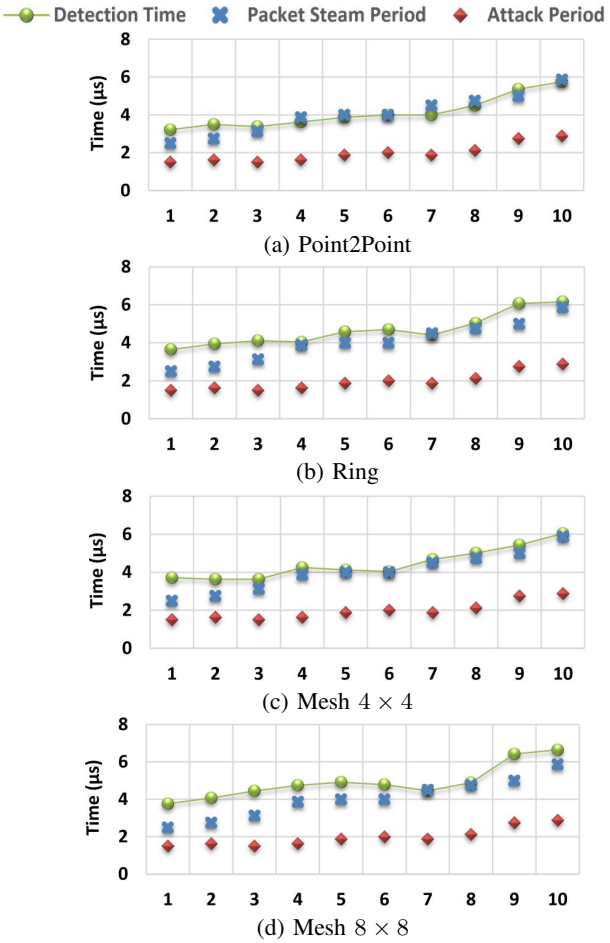


Fig. 15: Attack detection time for different topologies when running synthetic traffic patterns with the presence of one M3PIP. Each graph shows time in microseconds (y-axis) against test case ID (x-axis).

attack in the multiple M3PIP scenario is typically less than the single M3PIP scenario. When more IPs are malicious, the detection time shows a decreasing trend. This is expected since multiple attackers flood the NoC faster and cause PAC bound violations quicker. To compare detection time with packet stream period and attack period, we have shown the detection time variation in the presence of four M3PIPs across benchmarks in Figure 17.

C. Efficiency of Real-time DoS Attack Localization

We measured the efficiency of attack localization by measuring the time it takes from detecting the attack to localizing the malicious IPs. According to our protocol, this is mainly dominated by the latency for packet traversal from victim to attacker (V2AL) as well as the timeout (TOUT) described in Section IV-D. Figure 18 shows these statistics using the same set of synthetic traffic patterns for the single M3PIP scenario. The experimental setup for the localization results corresponds to the experimental results for the detection results in Figure 15. Unlike the detection phase, since the localization time depends heavily on the time it takes for the diagnostic packets to traverse from the IPs connected to the routers that flagged the attack to the potentially malicious IPs, the localization time varies for each topology. For example, in a

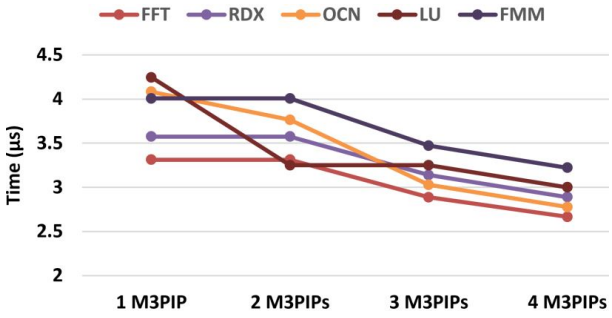


Fig. 16: Attack detection time when running real benchmarks with the presence of different number of M3PIPs.

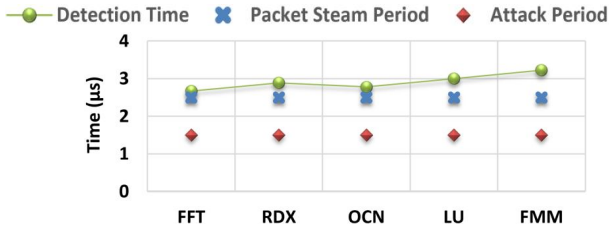


Fig. 17: Attack detection time when running real benchmarks with the presence of four M3PIPs.

Point2Point topology, localization needs diagnostic message to travel only one hop, whereas a Mesh8x8 topology may require multiple hops. Therefore, localization is faster in Point2Point compared to Mesh8x8 as shown in Figure 18. The localization time is less compared to detection time because the localization process completes once the small number of diagnostic packets reach all the potentially malicious IPs, whereas detection requires many packets before violating a PAC bound during runtime.

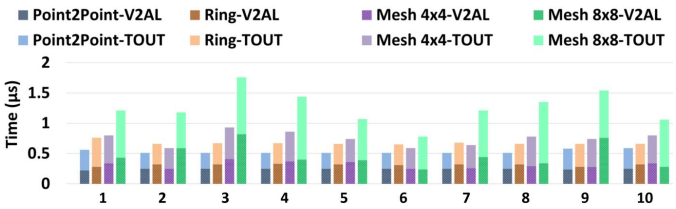


Fig. 18: Attack localization time for synthetic traffic patterns in the presence of one M3PIP. Figure shows time in microseconds (y-axis) against test case ID (x-axis) across different topologies. Test cases correspond to the test cases in Figure 15.

Results for DDoS attack localization in the presence of multiple M3PIPs when running real benchmarks is shown in Figure 19. Similar to the experiments done for DDoS attack detection efficiency, localization results are shown for one, two, three and four M3PIPs attacking the victim IP at the same time. The time is measured as the time it takes since launching the attack, until the localization of all M3PIPs. Once the first iteration of localization and detection is complete, the attack has to be detected again before starting the localization procedure. Therefore, the y-axis shows detection as well as localization time. For clarity of the graph, unlike in Figure 18, we have shown total localization time for each iteration rather than dividing the localization time as V2AL and TOUT. For both one and two M3PIP scenarios, only one iteration of detection and localization is required. When the third M3PIP is added, the two congested paths from victim to second M3PIP and from victim third M3PIP overlap. Therefore, only the first and third M3PIPs are localized during the first iteration leaving the second M3PIP to be detected during the second iteration. Similarly, in the four M3PIP scenario, first, third and fourth M3PIPs are localized during the first iteration and the second M3PIP, during the second iteration. This is consistent with our discussion presented in Section IV-D2. The results show that both detection and localization can be achieved in real-time. If a system requires only detection, the architecture of our framework allows easy decoupling of the two steps.

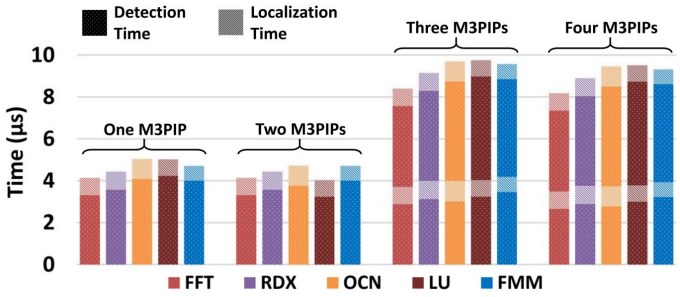


Fig. 19: Attack localization time when running real benchmarks with the presence of different number of M3PIPs.

D. Overhead Analysis

The overhead is caused by the additional hardware that is required to implement the DoS attack detection and localization processes. The detection process requires additional hardware components and memory implemented at each router to monitor packet arrivals as well as store the parameterized curves. The localization process uses DLCs stored at IPs and the communication protocol implemented at the routers. Figure 20 shows an overview of how our security components are integrated into the NoC components. The observation made in Section V-A allows us to reduce the number of parameters required to model the PACs, and as a result, reduces the additional memory requirement and improves performance. The following sections evaluate the power, performance and area overhead of the optimized algorithms.

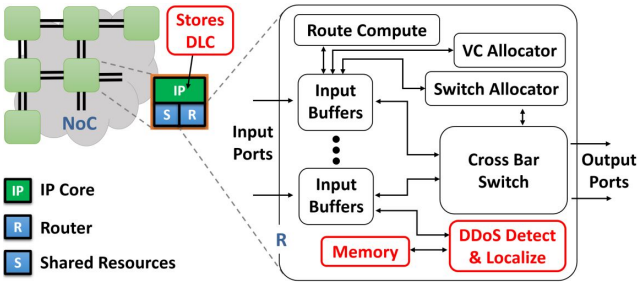


Fig. 20: Block diagram of NoC architecture showing additional hardware required to implement our security protocol in red.

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

During the localization phase, the diagnostic messages do not lead to additional congestion for two reasons. (1) As shown in Algorithm 2, the diagnostic message is transmitted along the reverse direction of the congested path. Since routers utilize two separate uni-directional links, the diagnostic messages are not sent along the congested path. (2) While it is unlikely, it is possible for multiple M3PIPs to carefully select multiple victims to construct a congested path in both directions. Even in this scenario, the number of diagnostic messages is negligible. This is because when an attack is flagged by the detection mechanism, diagnostic messages are sent to the source IPs which have violated the DLC threshold. Since the number of such source IPs can be at most the number of IPs communicating with the node that detected the attack, the performance impact by diagnostic messages is negligible.

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

Router: The proposed leaky bucket algorithm is lightweight and can be efficiently implemented with just three parameters per PAC bound as discussed above. The localization protocol requires two-bit flags at each port resulting in 10 bits of memory per router in Mesh topology. To evaluate the area and power overhead of adding the distributed DoS attack detection and localization mechanism at each router, we modified the RTL of an open-source NoC Router [37]. The design is synthesized with the 180nm GSCLib Library from Cadence using the Synopsys Design Compiler. It gave us area and power overhead of 6% and 4%, respectively, compared to the default router.

Packet Header: In a typical packet header, the header flit contains basic fields such as source, destination addresses and the physical address of the (memory) request. Some cache coherence protocols include special fields such as flags and timestamps in the header. If the header carries only the basic fields, the space required by these fields are much less compared to the wide bit widths of a typical NoC link. Therefore, most of the available flit header space goes unused [38]. We used some of these bits to carry the timestamp to calculate latency. This eliminates the overhead of additional flits, making better utilization of bits that were being wasted. If the available header bit space is not sufficient, adding an extra "monitor tail flit" is an easily implementable alternative [38]. In most NoC protocols, the packet header has a *hop count* or *time-to-live* field. Otherwise, it can be derived from the source, destination addresses and routing protocol details.

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

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

VI. Case Study with Intel KNL architecture

In the previous section, we have applied our approach using a regular 4x4 Mesh architecture (Figure 13). In order to demonstrate the applicability of our approach across NoC architectures, in this section, we evaluate the efficiency of our approach in an architecture model similar to one of the commercially available SoCs - Intel's KNL architecture. Knights Landing (KNL) is the codename for the second generation Xeon-Phi processor introduced by Intel [14]. We model the architecture on gem5 according to a validated simulator model [39] and show results for both DDoS attack detection and localization.

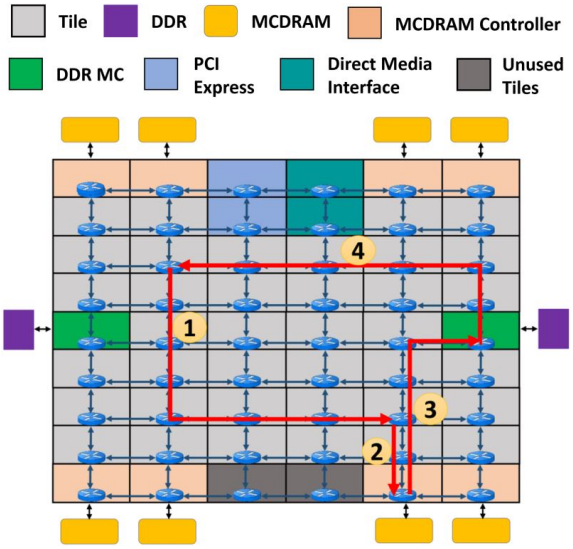


Fig. 21: Overview of the KNL architecture together with an example of MCDRAM miss in *cache* memory mode and *all-to-all* cluster mode: (1) L2 cache miss. Memory request sent to check the tag directory, (2) request forwarded to MCDRAM which acts as a cache after miss in tag directory, (3) request forwarded to memory after miss in MCDRAM, (4) data read from memory and sent to the requester [14].

TABLE I: System configuration parameters used when modelling KNL on gem5 simulator.

Processor Configuration	
Number of cores	32
Core frequency	1.4 GHz
Instruction set architecture	x86
Memory System Configuration	
L1 cache	private, separate instruction and data cache. Each 16kB in size.
Cache coherence	distributed directory-based protocol
Memory size	4GB DDR
MCDRAM	shared, direct mapped cache
Access latency	300 cycles
Interconnection Network Configuration	
Topology	4x8 Mesh
Routing scheme	X-Y deterministic
Router	4 port, 4 input buffer router with 5 cycle pipeline delay
Link latency	1 cycle

The KNL architecture, which is designed for highly parallel workloads, provide 36 tiles interconnected on a Mesh NoC. An overview of the KNL architecture is shown in Figure 21. It implements a directory-based cache coherence

protocol and supports two types of memory (i) multi-channel DRAM (MCDRAM) and (ii) double data rate (DDR) memory. The architecture gives the option of configuring these two memories in several configurations which are called *memory modes*. Furthermore, the affinity between cores, directories and memory controllers can be configured in three modes which are known as *cluster modes*. The memory and cluster modes allow configuration of the architecture depending on the application characteristics to achieve optimum performance and energy efficiency. Each combination of memory and cluster modes cause different traffic patterns in the NoC [40]. Our goal is to simulate the NoC traffic behavior in a realistic architecture and evaluate how our security framework performs in it.

We model a similar architecture on gem5 to evaluate how our DDoS attack detection and localization framework will perform in a realistic setup. The gem5 model is adopted from our previous work in [39] which validated the gem5 simulator statistics with the actual hardware behavior of a Xeon Phi 7210 platform [41]. In this model, 32 tiles connect on a Mesh NoC. Each tile is composed of a core that runs at 1.4 GHz, private L1 cache, tag directory and a router. Each cache is split into data and instruction caches with 16kB capacity each. The complete set of simulation parameters are summarized in Table I. The memory controllers are placed to match the architecture shown in Figure 21. We made few modeling choices that deviates from the actual KNL hardware due to the following reasons;

- 32 tiles are used instead of the 36 in KNL since the number of cores in gem5 must be a power of 2. This can be considered as a use-case where the KNL hardware has switched off cores in four of its tiles.
- The cache sizes we used are less compared to the actual KNL hardware numbers. This was done to get 95% hit rate in L1 cache, which is usually the hit rate when running embedded applications for the benchmarks we used. If we used a larger cache size, the L1 hit rate would be 100%, and NoC optimization will not affect cache performance.
- KNL runs AVX512 instructions whereas the gem5 model runs X86. gem5 is yet to support AVX512 instructions.
- Each tile in KNL consists of two cores. Our detection mechanism is capable of detecting DDoS attacks irrespective of whether one or both cores in a tile are active. However, the localization method can only pinpoint which tile is malicious. Since detection as well as localization happens at the router level, it is not possible to pinpoint the malicious core in a tile if both cores are active. Therefore, in our experimental setup, we assumed that one core per tile is active simulating 50% utilization.

Therefore, our gem5 model is a simplified version of the real KNL hardware. However, our previous work has validated the model and related performance and energy results to show that it accurately captures relative advantages/disadvantages of using different memory and cluster modes [39]. To evaluate our security framework, out of the memory and cluster modes, we model the *cache* memory mode and *all-to-all* cluster mode.

- **Cache memory mode:** In the *cache* mode, MCDRAM acts as a last level cache which is placed in between the DDR memory and the private cache. All memory requests first go to the MCDRAM for a cache memory lookup, if there is a cache miss, they are sent to the DDR memory.
- **All-to-all cluster mode:** In this mode, there is no affinity between the core, memory controller and directory. That is, a memory request can go from any directory to any memory controller.

The traffic flow when applications are running is defined by these modes. Figure 21 shows an example traffic flow.

We ran the same real traffic patterns (benchmarks) we used in Section V-A. To mimic the highly parallel workloads executable by the KNL architecture, we utilized 50% of the total available cores when running each application by running an instance of the benchmarks in each active core. The DDR address space was used uniformly for each benchmark. Attackers were modeled and placed randomly in 25% of the tiles that doesn't have an application instance. The DDoS attack was launched at the memory controller that experienced highest traffic during normal operation. Given that our model has 32 cores, 16 of them ran instances of the benchmark and 4 of the non-active cores injected packets directed at the memory controller to simulate the behavior of malicious IPs launching a DDoS attack. The packet stream period and attack period were selected as explained in Section V-B. Figure 22 shows the placement of the four M3PIPs, cores running the benchmarks (active cores) and the victim IP when running the RADIX benchmark. The victim IP depends on the benchmark since it is the IP connected to the memory controller experiencing highest traffic during normal operation.

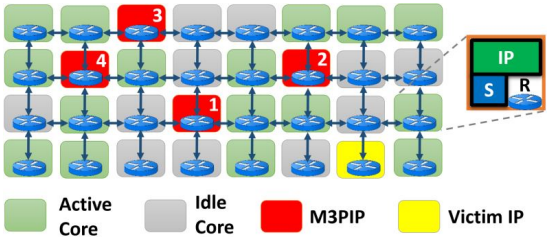


Fig. 22: 4×8 Mesh NoC architecture used to simulate DoS attacks in an architecture similar to KNL.

Similar to the experimental results presented in Section V-A, the DDoS attack detection results are shown in Figure 23 and Figure 24. Figure 23 shows detection time variation across benchmarks and number of M3PIPs. A zoomed-in version of the four M3PIP scenario is shown in Figure 24. Attack localization results are shown in Figure 25. Until the fourth M3PIP is added, there are no overlapping congested paths. Therefore, the M3PIPs are localized using only one iteration. Once the fourth M3PIP is added, the first, third and fourth M3PIPs are localized during the first iteration and a second iteration is required to localize the second M3PIP. This is reflected in localization time in Figure 25. From these as well as the previous results we notice that our detection and localization framework gives real-time results across different topologies and architectures.

VII. Discussion

Our proposed approach is designed for DDoS attack detection and localization, and therefore, it is not suitable to

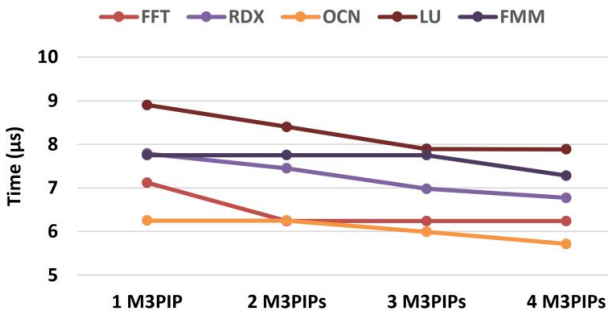


Fig. 23: Attack detection time when running real benchmarks on an architecture similar to KNL with the presence of different number of M3PIPs.

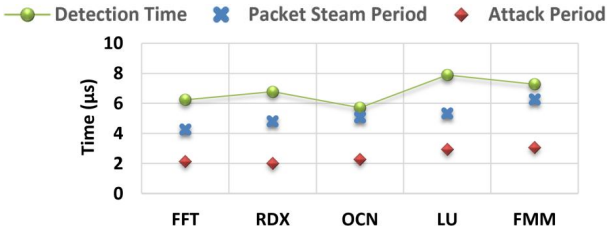


Fig. 24: Attack detection time when running real benchmarks on an architecture similar to KNL with four M3PIPs.

capture other forms of security violations such as eavesdropping, snooping and buffer overflow. Specific security attacks would require other security countermeasures which are not covered in this paper. Due to the low implementation cost, our approach can be easily coupled with other security countermeasures. For example, [42] discussed a snooping attack in which the header of the packet is modified before injecting into the NoC. This will alter the source address of the packet. While our detection mechanism does not depend on any of the header information of the packet, since our localization method uses the source address to localize the M3PIPs, an address validation mechanism needs to be implemented at each router to accommodate header modification. The address validation can be implemented as follows. Before a router injects each packet that comes from the local IP into the NoC, the router can check the source address and if it not the address of the local IP attached to that router, the router can drop it without injecting in to the NoC.

Our proposed work is targeted for embedded systems with real-time constraints. Such systems allow only a specific set of scenarios in order to provide real-time guarantees. Features commonly observed in general purpose computing such as task mapping, runtime task-migration, adaptive routing and introduction of new applications during runtime are beyond the scope of this work. In order to apply our proposed approach in general purpose systems, we need to store PACs and DLCs corresponding to each scenario and select the respective curves during runtime. As discussed in Section V-D, the hardware overhead to store the parameterized curves for each scenario is minimal, which consists of two major parts (i) overhead for storing the curves ($1 \times m \times 4$ bytes), and (ii) overhead for runtime monitoring (6% of NoC area). For example, if we consider an 8x8 Mesh, the memory overhead to store the curves would be 56 bytes ($m = 14$). If N scenarios are

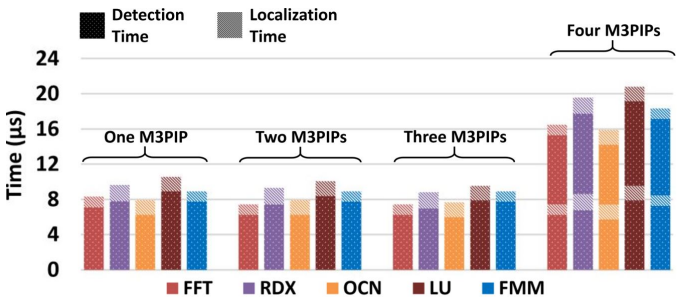


Fig. 25: Attack localization time when running real benchmarks on an architecture similar to KNL with the presence of different number of M3PIPs.

considered, the overhead would be 6% + $N \times 56$. Therefore, it may be feasible to consider a small number of scenarios (e.g., $N < 10$) without violating area overhead constraints.

VIII. Conclusions

This paper presented a real-time and lightweight DDoS attack detection and localization mechanism for IoT and embedded systems. It relies on real-time network traffic monitoring to detect unusual traffic behavior. This paper made two major contributions. It proposed a real-time and efficient technique for detection of DDoS attacks originating from multiple malicious IPs in NoC-based SoCs. Once an attack is detected, our approach is also capable of real-time localization of the malicious IPs using the latency data in the NoC routers. We demonstrated the effectiveness of our approach using several NoC topologies and traffic patterns. In our experiments, all the attack scenarios were detected and localized in a timely manner without causing any false positives or false negatives. Overhead calculations have revealed that the area overhead is less than 6% to implement the proposed framework on a realistic NoC model. This framework can be easily integrated with existing security mechanisms that address other types of attacks such as eavesdropping, snooping and buffer overflow.

Acknowledgment

This work was partially supported by the National Science Foundation (NSF) grant SaTC-1936040.

References

[1] F. Farahmandi, Y. Huang, and P. Mishra, "Trojan localization using symbolic algebra," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2017, pp. 591–597.
[2] "Alteris FlexNoC Resilience Package," http://www.arteris.com/flexnoc-resilience-package-functional-safety, [Online].
[3] A. Saeed, A. Ahmadinia, M. Just, and C. Bobda, "An id and address protection unit for noc based communication architectures," in *International Conf. on Security of Information and Networks*, 2014, pp. 288–294.
[4] R. JS, D. M. Ancajas, K. Chakraborty, and S. Roy, "Runtime detection of a bandwidth denial attack from a rogue network-on-chip," in *International Symposium on Networks-on-Chip*, 2015, pp. 1–8.
[5] S. Charles, M. Logan, and P. Mishra, "Lightweight Anonymous Routing in NoC based SoCs," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2020.
[6] S. Charles, Y. Lyu, and P. Mishra, "Real-time detection and localization of dos attacks in noc based socs," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 1160–1165.
[7] W. Wang, P. Mishra, and A. Gordon-Ross, "Dynamic cache reconfiguration for soft real-time systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 11, no. 2, pp. 1–31, 2012.

[8] S. Charles, H. Hajimiri, and P. Mishra, "Proactive thermal management using memory-based computing in multicore architectures," in *International Green and Sustainable Computing Conference (IGSC)*. IEEE, 2018, pp. 1–8.

[9] P. Waszecki, P. Mundhenk, S. Steinhorst, M. Lukasiewycz *et al.*, "Automotive electrical and electronic architecture security via distributed in-vehicle traffic monitoring," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 11, pp. 1790–1803, 2017.

[10] J. Xie and M. Xie, "Delay bound analysis in real-time networks with priority scheduling using network calculus," in *International Conference on Communications (ICC)*. IEEE, 2013, pp. 2469–2474.

[11] L. Fiorin, G. Palermo, and C. Silvano, "A security monitoring service for nocs," in *IEEE/ACM/IFIP international conference on Hardware/Software codesign and system synthesis*, 2008, pp. 197–202.

[12] T. Boraten, D. DiTomaso, and A. K. Kodi, "Secure model checkers for network-on-chip (noc) architectures," in *International Great Lakes Symposium on VLSI (GLSVLSI)*. IEEE, 2016, pp. 45–50.

[13] "Using TinyCrypt Library, Intel Developer Zone, Intel, 2016." https://software.intel.com/en-us/node/734330, [Online].

[14] A. Sodani, R. Gramunt, J. Corbal, H.-S. Kim *et al.*, "Knights landing: Second-generation intel xeon phi product," *Ieee micro*, vol. 36, no. 2, pp. 34–46, 2016.

[15] D. Fang, H. Li, J. Han, and X. Zeng, "Robustness analysis of mesh-based network-on-chip architecture under flooding-based denial of service attacks," in *International Conference on Networking, Architecture and Storage*. IEEE, 2013, pp. 178–186.

[16] S. Kumar, "Smurf-based distributed denial of service (ddos) attack amplification in internet," in *International Conference on Internet Monitoring and Protection (ICIMP)*. IEEE, 2007, pp. 25–25.

[17] K. M. Elleithy, D. Blagovic, W. K. Cheng, and P. Sideleau, "Denial of service attack techniques: Analysis, implementation and comparison," 2005.

[18] W. Eddy *et al.*, "Tcp syn flooding attacks and common mitigations," RFC 4987, August, Tech. Rep., 2007.

[19] L. Zhang, S. Yu, D. Wu, and P. Watters, "A survey on latest botnet attack and defense," in *International Conference on Trust, Security and Privacy in Computing and Communications*. IEEE, 2011, pp. 53–60.

[20] J. Li, Y. Liu, and L. Gu, "Ddos attack detection based on neural network," in *International Symposium on Aware Computing*. IEEE, 2010, pp. 196–199.

[21] Z. Chao-yang, "Dos attack analysis and study of new measures to prevent," in *International Conference on Intelligence Science and Information Engineering (ISIE)*. IEEE, 2011, pp. 426–429.

[22] F. De Santis, A. Schauer, and G. Sigl, "Chacha20-poly1305 authenticated encryption for high-speed embedded iot applications," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, 2017, pp. 692–697.

[23] G. Alpár, L. Batina, L. Batten, V. Moonsamy *et al.*, "New directions in iot privacy using attribute-based authentication," in *ACM International Conference on Computing Frontiers*, 2016, pp. 461–466.

[24] L. Fiorin, C. Silvano, and M. Sami, "Security aspects in networks-on-chips: Overview and proposals for secure implementations," in *Euromicro Conference on Digital System Design Architectures, Methods and Tools (DSD 2007)*. IEEE, 2007, pp. 539–542.

[25] J. Niemela, "F-Secure Virus Descriptions," *F-Secure*, December 2007.

[26] S. Chakraborty, S. Künzli, and L. Thiele, "A general framework for analysing system properties in platform-based embedded system designs." in *Date*, vol. 3. Citeseer, 2003, p. 10190.

[27] U. Suppiger, S. Perathoner, K. Lampka, and L. Thiele, "A simple approximation method for reducing the complexity of modular performance analysis," *Tech. Rep. 329*, 2010.

[28] E. Wandeler and L. Thiele, "Real-Time Calculus (RTC) Toolbox," http://www.mpa.ethz.ch/Rtctoolbox, 2006, [Online].

[29] J.-Y. Le Boudec and P. Thiran, *Network calculus: a theory of deterministic queuing systems for the internet*. Springer, 2001, vol. 2050.

[30] K. Lampka, S. Perathoner, and L. Thiele, "Analytic real-time analysis and timed automata: a hybrid method for analyzing embedded real-time systems," in *ACM int. conf. on Embedded software*, 2009, pp. 107–116.

[31] K. Huang, C. Buckl, G. Chen, and A. Knoll, "Conforming the runtime inputs for hard real-time embedded systems," in *Design Automation Conference (DAC)*. IEEE, 2012, pp. 430–436.

[32] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt *et al.*, "The gem5 simulator," *ACM SIGARCH computer architecture news*, vol. 39, no. 2, pp. 1–7, 2011.

[33] N. Agarwal, T. Krishna, L.-S. Peh, and N. K. Jha, "Garnet: A detailed on-chip network model inside a full-system simulator," in *International symposium on performance analysis of systems and software*. IEEE, 2009, pp. 33–42.

[34] 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.

[35] M. Lukasiewycz, S. Steinhorst, and S. Chakraborty, "Priority assignment for event-triggered systems using mathematical programming," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2013, pp. 982–987.

[36] S. Baruah, G. Buttazzo, S. Gorinsky, and G. Lipari, "Scheduling periodic task systems to minimize output jitter," in *International Conf. on Real-Time Computing Systems and Applications*. IEEE, 1999, pp. 62–69.

[37] A. Monemi, J. W. Tang, M. Palesi, and M. N. Marsono, "Pronoc: A low latency network-on-chip based many-core system-on-chip prototyping platform," *Microprocessors and Microsystems*, vol. 54, pp. 60–74, 2017.

[38] M. Ramakrishna, V. K. Kodati, P. V. Gratz, and A. Sprintson, "Gca: Global congestion awareness for load balance in networks-on-chip," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 7, pp. 2022–2035, 2015.

[39] S. Charles, C. A. Patil, U. Y. Ogras, and P. Mishra, "Exploration of memory and cluster modes in directory-based many-core cmps," in *IEEE/ACM Int. Symposium on Networks-on-Chip*, 2018, pp. 1–8.

[40] S. Charles, A. Ahmed, U. Y. Ogras, and P. Mishra, "Efficient cache reconfiguration using machine learning in noc-based many-core cmps," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 24, no. 6, pp. 1–23, 2019.

[41] "Intel Xeon Phi Processor 7210." http://ark.intel.com/products/94033/Intel-Xeon-Phi-Processor-7210-16GB-1_30-GHz-64-core, [Online].

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

Subodha Charles is a Ph.D candidate at the department of Computer and Information Sciences and Engineering at the University of Florida. He received his B.Sc from the department of Electronics and Telecommunications Engineering, University of Moratuwa, Sri Lanka in 2015. His current research interests include energy-aware computing, reconfigurable architectures and machine learning. He serves as Vice Chair - Education & Awards of IEEE Entrepreneurship steering committee.

Yangdi Lyu received his B.E. degree in Department of Hydraulic Engineering from Tsinghua University, Beijing, China in 2011. He is currently pursuing the Ph.D. degree with the Department of Computer and Information Sciences and Engineering at the University of Florida. His research interests include the development of test generation techniques for hardware trust, the security validation of system-on-chip, and microarchitectural side-channel analysis.

Prabhat Mishra (SM'08) is a Professor in the Department of Computer and Information Science and Engineering at the University of Florida. His research interests include hardware security and trust, energy-aware computing, and system-on-chip validation. Prof. Mishra currently serves as an Associate Editor of ACM Transactions on Design Automation of Electronic Systems and IEEE Transactions on VLSI Systems. He is an ACM Distinguished Scientist and a Senior Member of IEEE.