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

Subodha Charles, Yangdi Lyu, Prabhat Mishra
Department of Computer and Information Science and Engineering
University of Florida, Gainesville, Florida, USA

***Abstract*—Network-on-Chip (NoC) is widely employed by multi-core System-on-Chip (SoC) architectures to cater to their communication requirements. The increased usage of NoC and its distributed nature across the chip has made it a focal point of potential security attacks. Denial-of-Service (DoS) is one such attack that is caused by a malicious intellectual property (IP) core flooding the network with unnecessary packets causing significant performance degradation through NoC congestion. In this paper, we propose a lightweight and real-time DoS attack detection mechanism. Once a potential attack has been flagged, our approach is also capable of localizing the malicious IP using latency data gathered by NoC components. Experimental results demonstrate the effectiveness of our approach with timely attack detection and localization while incurring minor area and power overhead (less than 6% and 4%, respectively).**

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], [2]. 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.

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 [3]. If an M3PIP targets the same node, the SoC performance will suffer significant degradation [4]. With the increased popularity of internet-of-things (IoT) and embedded devices, SoCs are used in well-defined and time-critical systems. These systems can be one of the main targets of DoS attacks due to their real-time requirements with task deadlines. Early detection of DoS attacks in such systems is crucial as increased latencies in packet transmission can lead to real-time violations and other consequences.

This work was partially supported by the NSF grant CNS-1526687.

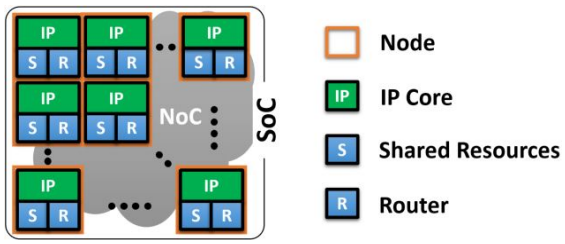


Fig. 1: NoC based many-core architecture connecting heterogeneous IPs on a single SoC. Each IP connects to a router via a network interface. Depending on the selected topology, routers will be arranged across the NoC.

Importance of NoC security has led to many prior efforts to mitigate DoS attacks in an NoC such as traffic monitoring [4], [5] and formal verification-based methods [6]. Other real-time traffic monitoring mechanisms have also been discussed in non-NoC domains [7]. 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. We propose an efficient method that focuses on detecting changes in the communication behavior in real-time to identify DoS attacks. It is a common practise to encrypt critical data in an NoC packet and leave only few fields as plain text [8][1]. This motivated our approach to monitor communication patterns without analyzing the encrypted contents of the packets.

Our major contributions can be summarized as follows;

1) We propose a real-time and lightweight DoS 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 malicious source(s) in real-time once a DoS attack is detected.

[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 [8].

3) We have evaluated the effectiveness of our approach against different NoC topologies using both real benchmarks and synthetic traffic patterns.

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. Finally, Section VI 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 [6]. If the Trojan gets triggered, there should be a threat detection mechanism. Previous studies explored latency monitoring [4], centralized traffic analysis [5], security verification techniques [6] and design guidelines to reduce performance impacts caused by DoS attacks [9]. In [5], 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 performance and power overhead.

Waszecki et al. [7] 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 where the attack is originating from 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 comparison, our approach is generalized to an NoC architecture and is valid across deterministic routing protocols and furthermore, has the ability to localize the malicious IP.

III. System and Threat Models

The following two subsections provide an overview of the threat model and communication system model used in our security framework.

A. Threat Model

Previous works have explored two main types of DoS attacks on NoCs [10] - (i) An M3PIP 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. As a result of the injected traffic from the malicious IP 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. Our proposed approach can detect both types of DoS attacks.

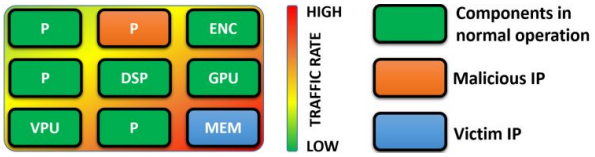


Fig. 2: Example DoS attack from a malicious IP to a victim IP in an NoC setup with Mesh topology.

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 [11]. We denote the set of all packets passing through router r during a program execution as a *packet stream* P_r. Figure 3 shows two packet streams within a specific time interval $[0, 16]$. The steam 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 [7] 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.

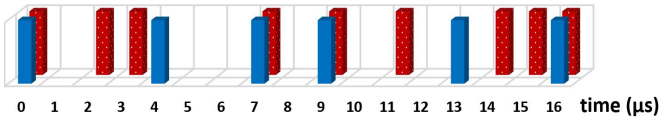


Fig. 3: Example of two event traces. Six blue event arrivals represent an excerpt of a regular packet stream P_r and nine red event arrivals represent a compromised packet stream $\widetilde{P_r}$.

IV. Real-Time Attack Detection and Localization

Figure 4 shows the overview of our proposed security framework. The first stage (upper part of the figure) illustrates the DoS attack detection phase while the second stage (lower part of the figure) represents the localization of M3PIP. 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. The local IP can then localize the M3PIP.

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 DoS attack detection mechanism implemented at each router. Section IV-D describes the localization of M3PIP.

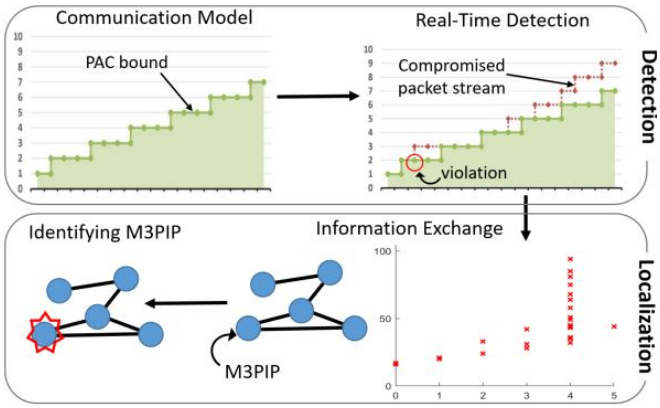


Fig. 4: Overview of our proposed framework: the system specification is analyzed to obtain the necessary packet arrival curves and detection parameters. These are used to design the real-time attack detection and localization framework.

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

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

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 3 are shown in Figure 5. During normal execution, the PACs should fall within the shaded area.

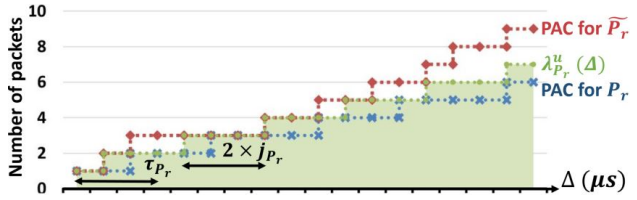


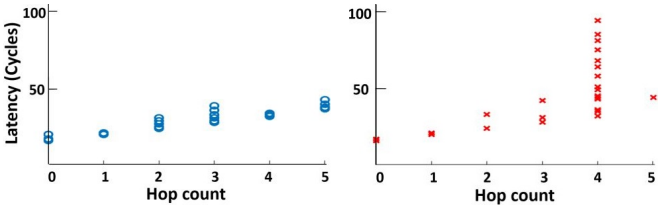
Fig. 5: Graph showing upper ($\lambda^u_{p_r}(\Delta)$) bound of PACs (green line with "●" markers) and the normal operational area shaded in green. The blue and red step functions show PACs corresponding to P_r and $\widetilde{P_r}$ respectively.

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} [12].

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

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 6(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 6(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 source 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.



(a) Normal operation (b) Attack scenario

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

C. Real-time Detection of DoS Attacks

The attack scenario can be formalized as follows;

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

The 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 5. However, this is not feasible because to construct the PAC, the entire packet stream should be observed. This doesn't lead to a real-time solution.

Therefore, we use the *leaky bucket* algorithm, which considers packet arrivals and the history of packet streams and gives a real-time solution [13]. 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. $\lambda^u_{pr}(\Delta)$, 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 [14].

Lines 1-4 initializes the timers (TIMER$_{r,s}$) to $\theta_{r,s}$ and packet counters at time zero (COUNTER$_{r,s}$) to corresponding initial packet numbers $\omega_{r,s}$, for each staircase function and packet stream P_r. The DoS attack detection process (lines 5-15) basically checks whether the initial packet capacities (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 shown in Section V, the algorithm allows real-time detection of DoS attacks under our threat model. Another important observation described in Section V-D1 drastically reduces the complexity of the algorithm allowing a lightweight implementation.

D. Real-time Localization of Malicious IPs

Figure 6(b) shows an example DLC during an attack scenario, where all IPs are injecting packets exactly the same way as shown in Figure 6(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 the real M3PIP (*attacker*). Therefore, we call it the *candidate M3PIP*.

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
```

Figure 7 shows four examples, where the attacker A is sending heavy traffic to a victim IP V, and as a result, local IP D is experiencing large latency for packets from source S. The first three examples in Figure 7 shows 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*.

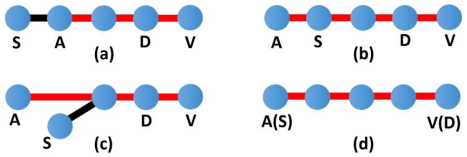


Fig. 7: Four scenarios of the relative positions of local IP (D), attacker IP (A), victim IP (V), and the candidate M3PIP (S) as found by D. The red line represents the congested path.

To avoid false positives, we propose a lightweight protocol implemented on the routers to detect the M3PIP. As the local IP D is informed by the attached router about an ongoing attack, D is definitely on the congested path. However, the source S detected by D may or may not be on the congested path (e.g., Figure 7(a) and Figure 7(c)). These false positives can be removed by checking the congestion status of the local router belonging to S. It is certain that S is not the attacker when its router is not congested. However, we cannot tell whether S is the attacker when the router of S is congested, e.g., the routers of Figures 7(b) and 7(d) are both congested. Our approach can successfully address all of these challenges. The event handlers of all routers for M3PIP localization are shown in Algorithm 2. Local IP sends the address of candidate M3PIP to its router. Each router maintains a three-state $flag$ for each port to identify the potential attacker. The $flag$ is 0, 1 and 2 to denote the potential attacker is undefined, local IP or others, respectively.

We describe the steps using Figure 7(b) as an example. The router of S will receive a message from the router of D saying that its local IP is a potential attacker. It will change the flag of the corresponding port to 1 to denote that the local IP is the potential attacker. S will receive another message from the router of V through the same port saying that A is the potential attacker. It will change the flag to 2 to denote that the attacker is some other IP. 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. Therefore, a broadcast is sent indicating that A is the attacker.

Our protocol relies on the victim to pinpoint the correct attacker and the 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 attacker plus the timeout. Therefore, the time complexity for localization is linear in the worst case with respect to the number of IPs α. It is important to note that the path from victim to attacker is not congested since each hop connecting two routers consists of two separate uni-directional links.

V. Experiments

We illustrate the feasibility of our approach in this section by presenting experimental results and discussing the overheads associated with it.

A. Experimental Setup

We tested the system on 5 real and 40 synthetic traffic traces. The synthetic traffic traces were generated by the cycle-accurate full-system simulator - gem5 [15]. The interconnection network was built on top of "GARNET2.0" model that is integrated with gem5. The default gem5 source was modified to include the detection algorithm. We evaluated 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.

Our approach was also evaluated using **real traffic patterns** based on 5 benchmarks (FFT, RADIX, OCEAN, LU, FMM) from the SPLASH-2 benchmark suite in Mesh 4×4 topology. The DoS attack was launched at a node connected to a memory controller. Relative placements of the M3PIP and victim IP were the same as for the synthetic traces running on Mesh 4×4 topology (test case ID 1 in Figure 8).

Algorithm 2: Event handlers for routers

```
1  upon event RESET:
2  flag[p_i] = 0 for all ports p_i

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

5  upon receiving address of the candidate M3PIP S from local IP:
6  send a query to the router of S for its congestion status
7  if S is congested then
8  |   sends a diagnostic message < S, D > to all routers in the path from S to
   |     D indicating that S is the potential attacker
9  end

10 upon receiving a diagnostic message < S, D > from port p_i:
11 start TIMEOUT if all flag == 0
12 if S is local IP and flag[p_i] == 0 then
13 |   flag[p_i] = 1              // local IP is the M3PIP
14 end
15 if S is not local IP then
16 |   flag[p_i] = 2              // local IP is not the M3PIP
17 end

18 upon event TIMEOUT:
19 if flag contains 1 then
20 |   broadcasting that its local IP is the attacker
21 |   RESET
22 end
```

B. Efficiency of Real-time DoS Attack Detection

Figure 8 shows the detection time across different topologies for synthetic traffic traces. 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 [7]. In addition to the time taken by the leaky bucket approach, the detection time is affected by the topology as well. When the victim is far from the malicious IP, the NoC traversal delay also contributes to the detection time. This is evident from Figure 8, which shows Point2Point topology where every node is one hop away resulting in less latency and Mesh 8×8 with 64 cores where some nodes can be multiple hops away. We observed the same trend when using the real-time traffic obtained by running the five benchmarks described earlier. These results confirm that the proposed approach can detect DoS attacks in real-time.

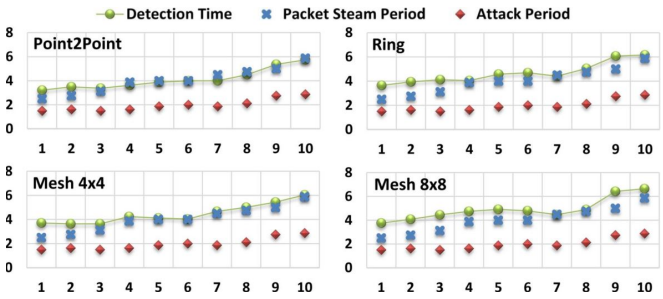


Fig. 8: Attack detection time for different topologies. Each graph shows time in microseconds (y-axis) against test case ID (x-axis).

C. Efficiency of Real-time DoS Attack Localization

We measure the efficiency of attack localization by measuring the time it takes from detecting the attack to localizing the malicious IP. 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 9 shows these statistics using the same set of synthetic traffic patterns. There was no difference in attack localization time when running the 5 real benchmarks compared to the synthetic traffic since the attack localization time depends only on V2AL and TOUT. The results show that both detection and localization can be achieved in real-time. If a system requires only detection and not localization, the architecture of our framework allows easy decoupling of the two steps.

D. Overhead Analysis

An important observation allows us to reduce the number of parameters required to model the PACs and as a result,

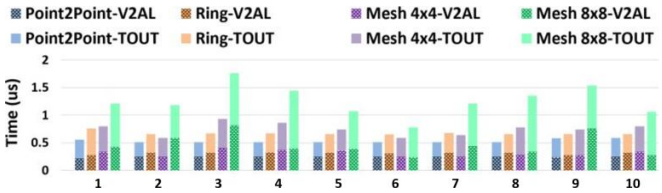


Fig. 9: Attack localizaion time for the used test cases.

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 [7] for message streams with similar 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 relationships between these parameters are derived in [14] 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 required 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 following sections evaluate the power, performance and area overhead of the optimized algorithms.

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 DoS attack detection.

2) Hardware overhead:

- **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 [16]. The design is synthesized with the 180nm GSCLib Library from Cadence using the Synopsys Design Compiler. It gave us area and power overheads of 5.93% and 3.87%, 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 [17]. 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 [17]. 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.

VI. Conclusions

This paper presents a real-time and lightweight DoS attack detection and localization mechanism for IoT and embedded systems. It relies on real-time network traffic monitoring to detect unusual traffic behavior. 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. Overhead calculations have revealed that the area overhead is less than 6% to implement the proposed framework on a realistic NoC model. This framework can be easily integrated with existing security mechanisms that address other types of attacks such as buffer overflow and information theft.

References

[1] Y. Huang *et al.*, "Scalable test generation for trojan detection using side channel analysis," *TIFS*, 2018.
[2] F. Farahmandi *et al.*, "Trojan localization using symbolic algebra," in *ASP-DAC*, 2017.
[3] S. Charles *et al.*, "Exploration of memory and cluster modes in directory-based many-core cmps," in *NOCS*, 2018, pp. 1–8.
[4] R. JS *et al.*, "Runtime detection of a bandwidth denial attack from a rogue network-on-chip," in *NOCS*, 2015.
[5] L. Fiorin *et al.*, "A security monitoring service for NoCs," *CODES+ISSS*, 2008.
[6] T. Boraten *et al.*, "Secure model checkers for Network-on-Chip (NoC) architectures," *Great Lakes Symposium on VLSI*, 2016.
[7] P. Waszecki *et al.*, "Automotive Electrical and Electronic Architecture Security via Distributed In-Vehicle Traffic Monitoring," *TCAD*, 2017.
[8] "Using TinyCrypt Library, Intel Developer Zone, Intel, 2016." https://software.intel.com/en-us/node/734330, [Online].
[9] D. Fang *et al.*, "Robustness analysis of mesh-based network-on-chip architecture under flooding-based denial of service attacks," *NAS*, 2013.
[10] L. Fiorin *et al.*, "Security aspects in networks-on-chips: Overview and proposals for secure implementations," *DSD*, 2007.
[11] S. Chakraborty *et al.*, "A General Framework for Analysing System Properties in Platform-Based Embedded System Designs," *DATE*, 2003.
[12] U. Suppiger *et al.*, "A simple approximation method for reducing the complexity of modular performance analysis," *Tech. Rep. 329*, 2010.
[13] J.-Y. Le Boudec and P. Thiran, *Network calculus: a theory of deterministic queuing systems for the internet*. Springer, 2001.
[14] K. Lampka *et al.*, "Analytic real-time analysis and timed automata: a hybrid method for analyzing embedded real-time systems," *ESWEEK*, 2009.
[15] N. Binkert *et al.*, "The gem5 simulator," *SIGARCH Computer Architecture News*, 2011.
[16] A. Monemi *et al.*, "ProNoC: A low latency network-on-chip based many-core system-on-chip prototyping platform," *Microprocessors and Microsystems*, 2017.
[17] M. Ramakrishna *et al.*, "GCA: Global congestion awareness for load balance in networks-on-chip," *TPDS*, 2016.