LOKI: A Hardware Trojan Affecting Multiple Components of an SoC

Manju Rajan[1 2], Abhijit Das[1 3] and John Jose[1]
[1]*Indian Institute of Technology Guwahati, Assam, India*
[2]*Government Engineering College Idukki, Kerala, India*
[3]*Univ Rennes, Inria, Lannion, France*
{manju18, johnjose}@iitg.ac.in, abhijit.a.das@inria.fr

***Abstract*—Outsourcing Intellectual Properties (IPs) from vendors around the globe exposes System-on-Chip (SoC) designs to malicious implants like Hardware Trojans (HTs). Carefully crafted HTs with extremely rare trigger conditions evade conventional validation. While Machine Learning (ML) based validation increases detection accuracy, they themselves are vulnerable to Trojan attacks. Available detection and mitigation techniques fall short when an HT is capable of affecting multiple components of an SoC. This paper proposes an intermittent and robust HT that can attack the Network-on-Chip (NoC), the shared cache, and the cores, all at once. The proposed HT is mounted in the Network Interface (NI) of a malicious IP and can be triggered by specific inputs. Experimental evaluation shows that the proposed HT can increase packet latency by 3.44x (affecting the NoC), increase miss penalty by 15% (affecting the cache) and decrease system speedup by 10% (affecting the cores). Moreover, this paper also discusses why state-of-the-art defence mechanisms are insufficient to tackle the proposed HT and suggests better solutions.**

***Index Terms*—Hardware Trojan (HT), System-on-Chip(SoC), Network-on-Chip (NoC), Last-Level Cache (LLC).**

I. Introduction

To reduce design costs and meet aggressive time-to-market constraints, System-on-Chip (SoC) manufacturers outsource Intellectual Properties (IPs) from different vendors around the globe. This supply-chain exposes SoC designs to malicious implants like Hardware Trojans (HTs). Typically, there are two critical parts in a Trojan, *trigger* and *payload*. An example Trojan in Figure 1 shows that a trigger with 2 logic gates is added to the original 4-input circuit. The trigger is usually created using one or more extremely subtle events (rare inputs, signals, transitions). Once triggered, the payload initiates the attack, like information leakage, Denial-of-Service (DoS), performance degradation, etc. In the example, out of the 16 (2^4) combination, only when the input is 1101 (unlucky 13), trigger is activated, and the payload inverts the original output.

The exponential growth of input combination space in modern SoCs coupled with carefully crafted HTs makes it infeasible for conventional simulation based validation to detect Trojans [1] [2]. Machine Learning (ML) based validation offers better detection, but they are computationally expensive, and themselves are susceptible to Trojan attacks [3]. Moreover, if an HT is capable of attacking multiple components of an SoC, even the existing detection and mitigation techniques will fail to locate and neutralise it. To launch such an attack, the Trojan must be inserted in a place that has access to multiple SoC components. This is the motivation for the proposed HT.

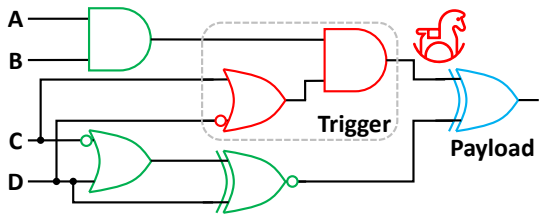

Figure 1: An example Hardware Trojan with trigger (red gates) and payload (blue gate). Only when the input is 1101, the trigger is activated, and the payload inverts the original output.

A. Threat Model

Due to the long and distributed supply-chain, HT like implants can be inserted into the Register-Transfer Level (RTL) or netlist of an IP from different access points. There are practical scenarios of Trojan insertion by Computer-Aided Design (CAD) tool, by designer or at the foundry by reverse engineering [4][5]. Network-on-Chip (NoC) facilitates communication between the IPs and has access to all the components of an SoC. Hence, it is more vulnerable to attacks, as a malicious IP can simply eavesdrop NoC packets to extract information without the need of hacking into individual IPs [6]. This work considers that the proposed HT is mounted on the Network Interface (NI) of the malicious IP, which connects it to the NoC IP (refer Figure 3). Sitting on the NI, the Trojan is able to attack the NoC, the shared Last-Level Cache (LLC) and other core IPs, all at once. The proposed Trojan attacks the SoC for performance degradation; however, based on intentions, it can always be used for other attacks.

B. Research Contributions

The proposed HT is named as *LOKI*[1], which is very mischievous in the sense that it sits on one IP but affects multiple components of the SoC, without actually hacking them. Specifically, the major contributions of this work are:

1) LOKI selectively duplicates NoC control packets to attack three SoC components. Using duplicate control packets to attack multiple components without hacking into them is a first of its kind in NoC based SoCs.

[1]God of Mischief in Norse Mythology

2) We show that when the attacker triggers LOKI, packet latency, miss penalty and system speedup are severely affected thereby degrading the overall SoC performance.
3) When multiple components are attacked at the same time, state-of-the-art defence mechanisms fall short to locate and isolate LOKI. Towards the end, we discuss possible ways to tackle LOKI-like malicious implants.

II. Related Work and Motivation

A. State-of-the-Art

Attackers typically use shared on-chip resources like LLC and NoC as a backdoor to gain access to the victim processes, applications and IPs. One of the earliest attacks on LLC called Prime+Probe tries to identify the cache line eviction pattern of a victim process by creating conflicts to steal sensitive information [7]. Other popular attacks on LLC includes Flush+Reload [8], Flush+Flush [9], and Streamline [10]. Being the communication backbone, NoC's positional advantage makes it a prime target for attackers. Moreover, its distributed nature eases attack replication and amplification. For example, a Prime+Probe like side-channel attack is also proposed in NoC, where its infrastructure is used to monitor the contents of the shared LLC [11]. There are a wide variety of Trojan based attacks launched on NoC based SoCs like, eavesdropping [12][13], data integrity [14], information leakage [15][16], DoS [17][18], delay-of-service [19][20], performance degradation [21][22], etc. Except [16], all other works actively use NoC resources to launch their attacks on the SoC. [16] proposed an attack called SIM+THANOS, which mounts the HT on the NI of one IP and steals information for an accomplice thread hiding in another IP. *The proposed Trojan LOKI is mounted on the NI of one IP and attacks multiple SoC components without other resources or accomplices, making it difficult to detect by state-of-the-art techniques.*

B. Motivation

IPs communicate with each-other by exchanging messages, having a *header* and a *payload*[2], as shown in Figure 2. The header contains necessary information, including source (SRC), destination (DEST), memory address (ADDR), message type (TYPE), etc., for the message traversal, whereas the payload carries data. There are two types of messages, *control* and *data*, and the payload remains empty in control messages as they are used to request data or send coherence messages. In an NoC based SoC, when a source (IP_{SRC}) wants to communicate with its destination (IP_{DEST}), it forwards the message to the NI (blue IP node), as shown in Figure 3. NI converts the message (yellow) into either a control or a data packet (brown) and forwards it to the router to travel through the NoC and reach destination. Similar steps in reverse are followed to receive the message at the destination (green IP node). Different techniques are proposed to encrypt (and decrypt) the messages and prevent data stealing in modern NoC based SoCs [23][13]. Usually, only the payload of the message (or packet) is encrypted as the header information is required for routing and arbitration decisions in the NoC. In other words, only the data packets are encrypted as control packets do not carry any payload. Our proposed Trojan LOKI exploits this vulnerability to duplicate control packets in the NI to attack the SoC. LOKI is carefully crafted to get triggered only with read request packets (refer Section III-A). The existing information leakage attack from NI, SIM+THANOS [16] ignores the possibility of encryption and duplicates data packets. *To the best of our knowledge, this is the first attempt to use read request packets to attack multiple SoC components.*

[2]Message payload and Trojan payload are entirely different

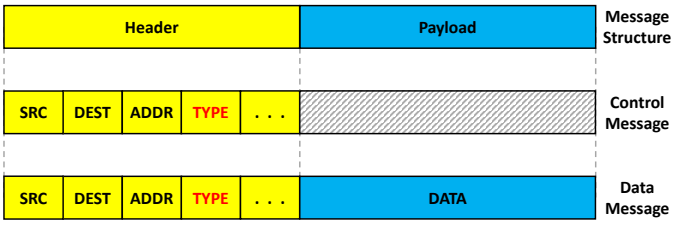

Figure 2: Structure of a message exchanged between IPs.

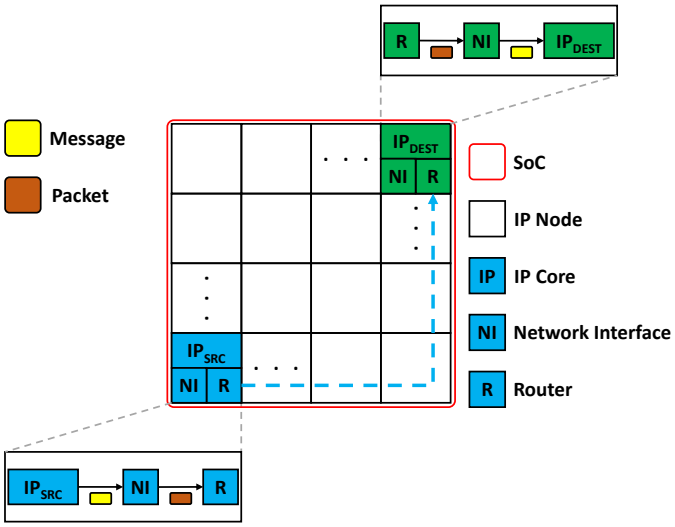

Figure 3: Communication between IPs in an NoC based SoC.

III. LOKI: Design and Demonstration

In this section, we first present the design of LOKI, including the circuit diagram of a possible way of inserting it into the NI. Then, we take multiple example scenarios to demonstrate how LOKI attacks and impacts multiple SoC components.

A. LOKI Design

As shown in Figure 3, NI is responsible for converting a message into a packet and vice-versa. Due to limited NoC channel width, a packet is divided into multiple smaller units called *flits*. A head flit carries the message header, and multiple body flits ended by the tail flit carries the payload. An NoC control and data packet can be represented as:

$$P^i_{CTRL} = \{F^i_{Head}\} \quad (1)$$

$$P^i_{DATA} = \{F^i_{Head} \mid F^i_{Body_0} \mid \ldots \mid F^i_{Body_{n-1}} \mid F^i_{Tail}\} \quad (2)$$

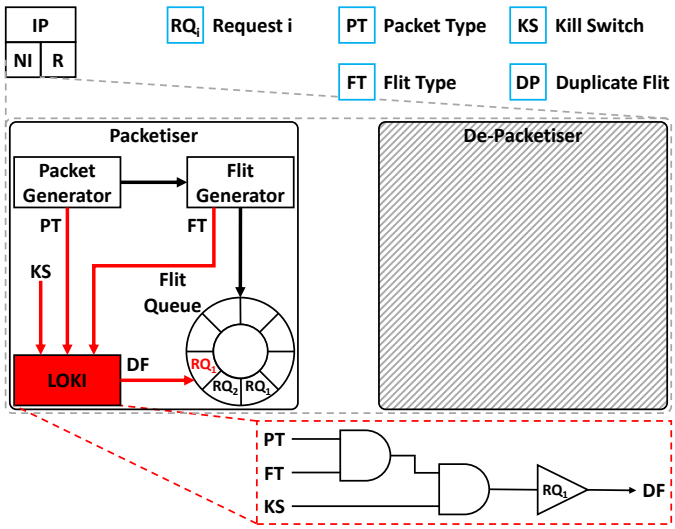

Figure 4: Insertion of the proposed Trojan LOKI in NI.

Figure 4 shows a possible way of inserting the proposed Trojan LOKI into the NI of an IP node. Usually, NI employs the *Packetiser* and *De-Packetiser* modules for the source and destination IP nodes, respectively. LOKI is inserted in the *Packetiser* and hence we omit the details about the *De-Packetiser* module in Figure 4. When NI receives a message from the source IP core, the *Packet Generator* sub-module converts it into a packet. The *Flit Generator* sub-module then converts the packet into one or more flits and stores them in the circular *Flit Queue*. Stored flits are inserted into the NoC router to travel towards their respective destination IP nodes.

As proposed in [17], a backdoor Kill Switch (KS) is used to trigger LOKI and initiate the attack. Packet Type (PT) provides the message type (TYPE from Figure 2), and Flit Type (FT) provides the type of flit (head, body or tail). We target read requests as they are control packets with just one head flit (refer Equation 1), thus less storage overhead to store a duplicate. LOKI uses a buffer of 1-flit size and can store only one duplicate read request packet at any given point of time. As shown in Figure 4, only when $PT = READ$, $FT = F^i_{Head}$ and $KS = EN$[3], LOKI gets triggered and copies a head flit (read request packet) from the *Flit Generator* sub-module into its buffer. LOKI then keeps inserting this duplicate flit into the *Flit Queue* until $KS = DS$. Please note that the duplicate flit is inserted only in the free locations to avoid interrupting the usual flow of inter IP communication.

B. LOKI Demonstration

As shown in Figure 5, let us consider an NoC based SoC with 2-levels of on-chip caching. L1 instruction (L1I) and data (L1D) caches are private to each IP core, whereas L2 is shared and distributed and is the Last-Level Cache (LLC). When an IP core encounters an L1 cache miss, it needs to communicate with the LLC to get the data block. Since LLC is distributed as multiple banks, the destination LLC bank could be anywhere, from the nearest to the farthest IP node. This is when the L1 Cache Controller (L1 CTLR) sends a request message to the NI intended for the destination LLC bank. Upon receiving such a request, the destination LLC bank Controller (LLC CTLR) sends a reply message with the corresponding data block.

Let us consider that LOKI is mounted on the NI of the IP node shown in blue in Figure 5, and is now triggered by the KS. We take two example scenarios to demonstrate how LOKI launches the attack on multiple SoC components at once.

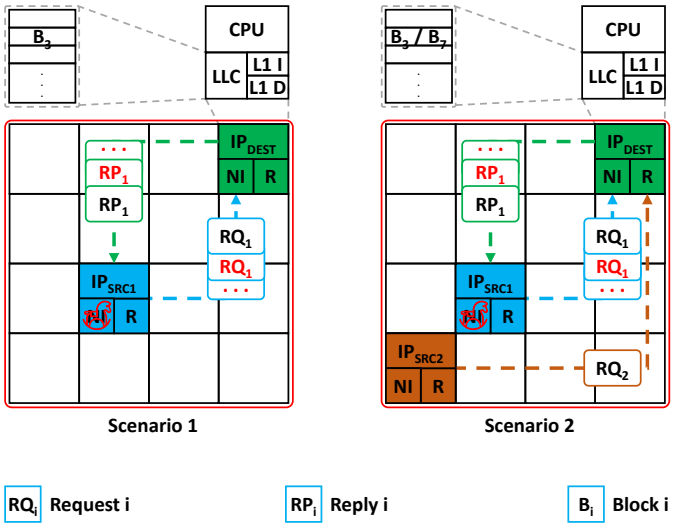

Figure 5: Demonstration of the attack launched by LOKI.

1) Scenario 1: IP_{SRC1} (blue) requests a data block B_3 and encounters an L1 cache miss. The corresponding L1 CTLR forwards a read request message RQ_1 to the NI for the destination LLC bank at IP_{DEST} (green) where B_3 is cached. The destination LLC CTLR sends the requested data block with a reply message RP_1. When LOKI is active, it copies RQ_1 from the *Flit Generator* into its buffer and keeps inserting this duplicate request in free locations of the *Flit Queue*. These duplicate requests reach the destination, and the LLC CTLR treats them like genuine by replying with RP_1. Since IP_{SRC1} and IP_{DEST} are sufficiently spaced, the duplicate requests and replies (red) spread over the entire SoC. This kind of scenario increases contention in the routers and links, resulting in increased packet latency (affecting the NoC).

2) Scenario 2: While Scenario 1 (III-B1) is in progress, IP_{SRC2} (brown) sends a read request message RQ_2 to IP_{DEST} for the data block B_7. According to the mapping strategy (say modulo 4) employed in the destination LLC bank, B_3 and B_7 are in the same set. Now, due to frequent requests for B_3 by duplicate RQ_1, B_7 could be evicted from the bank. In fact, the LLC CTLR enters a thrashing phase where useful blocks are evicted to service duplicate requests. Future requests for these B_7-like evicted blocks encounter LLC miss,

[3]READ = Read request, EN = Enable, and DS = Disable

Table 1: Workload mixes

Mix	Benchmarks	Copies	Characteristics
M1	GemsFDTD	1×16: 16	high MPKI
M2	astar		medium MPKI
M3	cactusADM		low MPKI
M4	GemsFDTD, lbm, xalancbmk, gobmk	4×4: 16	100% high MPKI
M5	astar, sjeng, omnetpp, sphinx		100% medium MPKI
M6	cactusADM, gromacs, perlbench, hmmer		100% low MPKI
M7	GemsFDTD, xalancbmk, astar, omnetpp		50% high MPKI, 50% medium MPKI
M8	astar, omnetpp, cactusADM, perlbench		50% medium MPKI, 50% low MPKI
M9	cactusADM, perlbench, GemsFDTD, xalancbmk		50% low MPKI, 50% high MPKI

Table 2: System configuration

Processor	16 OoO x86 cores
L1 Cache	16KB×16, 4-way, 64B blocks, private, split
L2 Cache (LLC)	256KB×16, 8-way, 64B blocks, shared
NoC	4×4 2D mesh, 4-VCs/port, 128-bit flit channel
Routing	2-stage routers, X-Y dimension-order routing
Packets	1-flit control packets, 5-flit data packets
Benchmarks	SPEC CPU2006 (multi-programmed)

which increases the miss penalty (affecting the caches).

From the increase in packet latency and miss penalty, it is intuitive that sources like IP_{SRC2} will suffer from delayed instruction execution. It has a cascading effect since delayed execution means delayed commit and thus delayed issue of new instructions. This decreases the system speedup (affecting the core). Even though LOKI is mounted on a single NI, it is capable of affecting multiple components without directly intruding in them, thereby paralysing the entire SoC.

IV. Experimental Analysis

We consider the following two architectures for evaluation:

- **Baseline:** Without any Trojan.
- **LOKI:** With the proposed Trojan on NI of one IP node.

A. Simulation Setup and Workloads

Baseline and LOKI architectures are modelled on event-driven gem5 simulator [24], and the system configuration is given in Table 2. We modify the GARNET [25] interconnection module inside gem5 to model LOKI and mount it on the NI of IP core 5 of a 4×4 NoC based SoC (similar to Figure 5).

To evaluate and analyse the attack, we consider multi-programmed SPEC CPU2006 benchmarks to mimic a modern NoC based SoC running multiple applications in parallel. We select benchmarks that show varying Misses Per Kilo Instructions (MPKIs) and categorise them as *high* (MPKI $\geq$ 40), *medium* (20 $\leq$ MPKI $<$40), and *low* (MPKI $<$ 20). Table 1 presents the workload mixes, where *M1* through *M3* run 16 copies of the same benchmark on all the 16 cores (1×16: 16). Whereas *M4* through *M9* run a random combination of 4 different benchmarks with 4 copies each (4×4: 16). All the workloads are run with 5 different combinations (application to core mapping) and the average of each of them are reported. For a relative comparison of the evaluation metrics, all the results are normalised with respect to the baseline architecture.

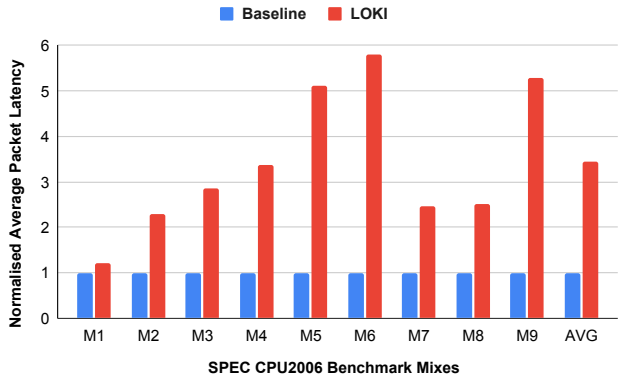

Figure 6: Normalised average packet latency

B. Impact on NoC

The number of cycles required for a packet to reach from source to its destination is called packet latency. Figure 6 shows the normalised average packet latency when LOKI is active. Duplicate request and reply packets create NoC congestion and increase average packet latency by 3.44x.

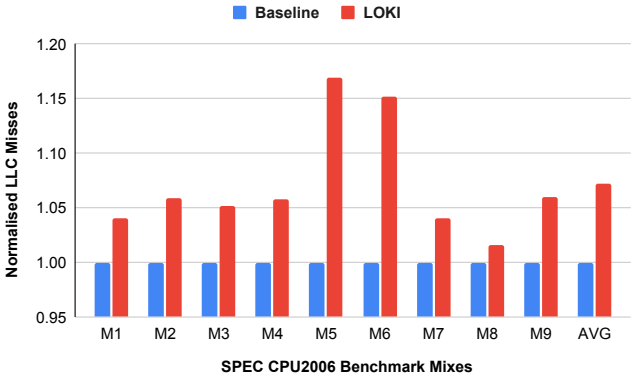

Figure 7: Normalised LLC misses

C. Impact on Caches

Figure 7 shows the normalised LLC misses when LOKI is active. We observe that duplicate requests create an intermittent probing of the same data block in the LLC bank, which results in the eviction of the genuine blocks. Hence, LLC misses increase across all the workloads with an average of 7%. Significant increase of more than 15% is observed in

workloads *M5* and *M6*, as they are made of *medium* and *low* MPKIs. Due to low packet injection rate in *M5* and *M6*, LOKI is able to inject more duplicate requests into the *Flit Buffer*. Workloads *M2* and *M3* also inject many duplicate requests, but the impact on LLC bank is less, as all the cores run the same benchmark with minimum interference (refer Table 1).

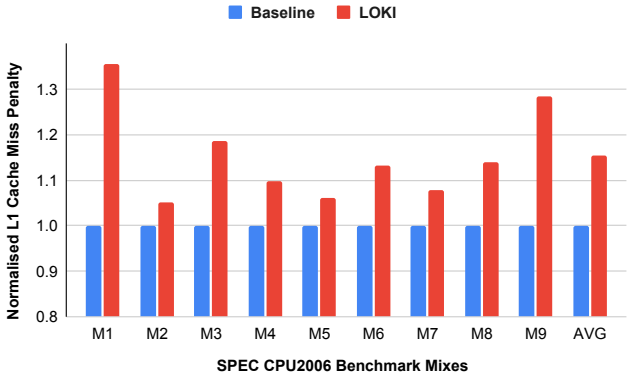

Figure 8: Normalised L1 cache miss penalty

The number of cycles required to service an L1 cache miss by bringing in the requested data block is called L1 cache miss penalty. With the increase in number of LLC misses, L1 cache miss penalty is bound to increase, as the missed data blocks are now fetched from the off-chip main memory. Figure 8 shows a maximum of up to 36% and an average of 15% increase in L1 cache miss penalty among the evaluated workloads.

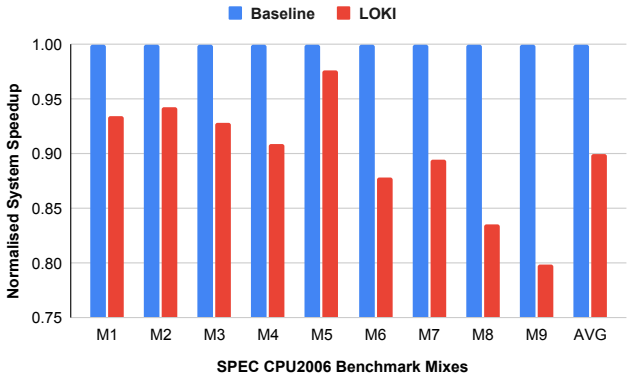

Figure 9: Normalised system speedup

D. Impact on Cores

An increase in packet latency and L1 cache miss penalty directly impacts instruction execution time. We use Instructions Per Cycle (IPC) to compare the system speedup between baseline and LOKI. A maximum of up to 20% and an average of 10% decrease in system speedup can be seen in Figure 9. It shows how LOKI is able to attack multiple SoC components indirectly and bring down the overall system performance.

V. Sensitivity and Overhead Analysis

A. LOKI vs An Existing Trojan

A recently proposed Trojan in NoC router alters the destination (DEST in Figure 2) in the header of packets to attack an SoC for performance degradation [21]. For ease of reference, we call that work as Packet Header Attack Trojan (PHAT). Since they considered L1 cache miss penalty as one of the evaluation metrics, we compare LOKI and PHAT with the same. To have a fair comparison, we identify that *cactusADM* and *hmmer* benchmarks dominated their workloads and thus consider workload *M6* for analysis. As shown in Figure 10, LOKI has a greater impact on the L1 cache miss penalty without any direct presence on the NoC or the shared cache.

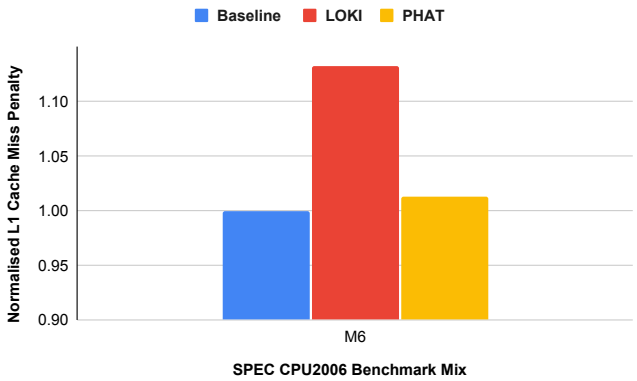

Figure 10: LOKI vs An Existing Trojan

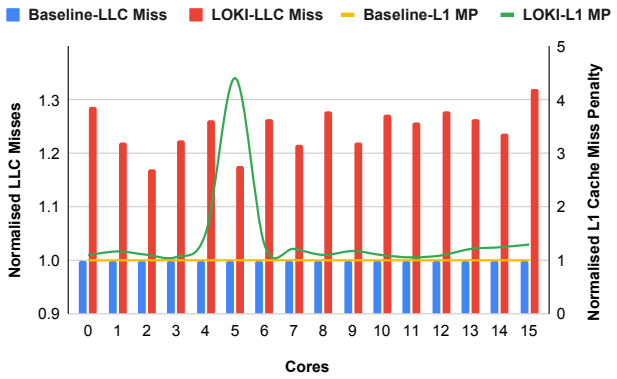

Figure 11: LLC Misses vs L1 Cache Miss Penalty

B. LLC Misses vs L1 Cache Miss Penalty

To understand the relationship between LLC misses and L1 cache miss penalty, we choose to dissect workload *M9*, which performs the worst in system speedup (refer Figure 9). We show a core-by-core distribution of LLC misses and L1 cache miss penalty of *M9* in Figure 11. Except in core 5, the relation between LLC misses and L1 cache miss penalty is almost uniform. The high spike in core 5 is due to the presence of LOKI in its NI. This is owing to the unnecessary delay by duplicate reply packets in the coherence protocol buffer.

C. LLC Misses vs Block Re-Reference

When an evicted LLC block is requested again in the near future, it is called block re-reference [26]. Using the same workload *M9*, we conduct an analysis to understand how the increased LLC misses on genuine block requests affect the number of their re-references. Figure 12 shows that the number

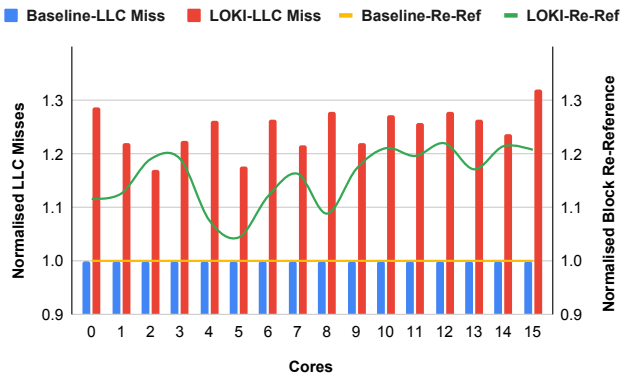

Figure 12: LLC Misses vs Block Re-Reference

of block re-reference linearly increases across cores with the increase in LLC misses, with an average of 16%. This increase is relatively less in core 5, as the duplicate block requests from this core evict genuine blocks and not the other way around.

D. Timing and Area Overhead

We use ProNoC [27], that facilitates prototyping of NoC based SoCs. The baseline and LOKI architectures are modelled in Verilog for which ProNoC generates the equivalent RTL. Vivado HLS 2020.2 webpack targeting Xilinx Kintex UltraScale FPGA board is used to synthesize the RTL. LOKI has a negligible increase of 0.29% LookUp Tables (LUTs) and 1.26% FlipFlops (FFs) while meeting the timing constraints.

VI. Discussion on Detection and Mitigation

The use of KS prevents logic testing from the accidental triggering of LOKI during the verification process [17]. As LOKI is intermittent and affects multiple SoC components indirectly, locating its position is difficult. For example, any existing detection technique employed in NoC, LLC or even the cores will not be effective [17][18][19]. Prevention and mitigation techniques like obfuscation, encryption, anonymous routing, etc., work well with eavesdropping, data integrity and information leakage attacks, but not LOKI [15][12][13]. This is because LOKI uses control packets for the attack, and they are usually not modified, as the header information is required at every intermediate router for making routing and arbitration decisions. If a technique to mask the header is proposed, unmasking it at every router along the way will fall in the critical path of execution thereby increasing the packet latency.

The mitigation technique proposed in SIM+THANOS [16] can not directly tackle LOKI as the header information remain unchanged. However, a similar approach that involves generating a unique message identity between a pair of source and destination can be explored at the IP cores. In this direction, a Packet Leak Detection Unit (PLDU) [28] that generates a unique tag for every packet can also be tried. A modified version of packet certification from [15] can be implemented to counteract LOKI. Another way of tackling LOKI and other SIM+THANOS-like Trojan attacks could be using something like a Miss Status Handling Register (MSHR) at the NI.

VII. Conclusion

This work introduces an HT that has minimal presence yet maximum impact in an NoC based SoC. In a first of its kind, the proposed Trojan uses control packets to attack the NoC, the shared cache and the core IPs to degrade overall system performance without directly intruding in any of them. Being selective and intermittent in nature, detection and mitigation of the Trojan pose unique challenges for future exploration.

References

[1] Y. Lyu and P. Mishra, "Scalable Activation of Rare Triggers in Hardware Trojans by Repeated Maximal Clique Sampling," *IEEE TCAD*, 2020.
[2] Z. Pan and P. Mishra, "Automated Test Generation for Hardware Trojan Detection using Reinforcement Learning," in *ASP-DAC*, 2021.
[3] Z. Pan and P. Mishra, "Design of AI Trojans for Evading Machine Learning-based Detection of Hardware Trojans," in *DATE*, 2022.
[4] M. Areno, "Supply Chain Threats Against Integrated Circuits," *Intel Whitepaper*, 2020.
[5] P. Mishra and S. Charles, *Network-on-Chip Security and Privacy*. Springer, 2021.
[6] S. Charles and P. Mishra, "A Survey of Network-on-Chip Security Attacks and Countermeasures," *ACM CSUR*, 2021.
[7] D. A. Osvik *et al.*, "Cache Attacks and Countermeasures: The Case of AES," in *Cryptographers' Track at The RSA Conference*, 2006.
[8] Y. Yarom and K. Falkner, "FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack," in *USENIX Security*, 2014.
[9] D. Gruss *et al.*, "Flush+Flush: A Fast and Stealthy Cache Attack," in *DIMVA*, 2016.
[10] G. Saileshwar *et al.*, "Streamline: A Fast, Flushless Cache Covert-Channel Attack by Enabling Asynchronous Collusion," in *ASPLOS*, 2021.
[11] C. Reinbrecht *et al.*, "Side Channel Attack on NoC-based MPSoCs are Practical: NoC Prime+Probe Attack," in *SBCCI*, 2016.
[12] S. Charles *et al.*, "Lightweight Anonymous Routing in NoC based SoCs," in *DATE*, 2020.
[13] S. Charles and P. Mishra, "Securing Network-on-Chip using Incremental Cryptography," in *ISVLSI*, 2020.
[14] S. Charles and P. Mishra, "Lightweight and Trust-Aware Routing in NoC-based SoCs," in *ISVLSI*, 2020.
[15] D. M. Ancajas *et al.*, "Fort-NoCs: Mitigating the Threat of a Compromised NoC," in *DAC*, 2014.
[16] V. Y. Raparti and S. Pasricha, "Lightweight Mitigation of Hardware Trojan Attacks in NoC-based Manycore Computing," in *DAC*, 2019.
[17] T. Boraten and A. K. Kodi, "Mitigation of Denial of Service Attack with Hardware Trojans in NoC Architectures," in *IPDPS*, 2016.
[18] S. Charles *et al.*, "Real-Time Detection and Localization of DoS Attacks in NoC based SoCs," in *DATE*, 2019.
[19] R. Manju *et al.*, "SECTAR: Secure NoC using Trojan Aware Routing," in *NOCS*, 2020.
[20] R. Manju *et al.*, "Trojan Aware Network-on-Chip Routing," in *Network-on-Chip Security and Privacy*. Springer, 2021.
[21] V. J. Kulkarni *et al.*, "Packet Header Attack by Hardware Trojan in NoC based TCMP and its Impact Analysis," in *NOCS*, 2021.
[22] M. H. Khan *et al.*, "Dead Flit Attack on NoC by Hardware Trojan and its Impact Analysis," in *NoCArc*, 2021.
[23] J. Sepúlveda *et al.*, "Towards Protected MPSoC Communication for Information Protection Against a Malicious NoC," *Elsevier Procedia Computer Science*, 2017.
[24] N. Binkert *et al.*, "The gem5 simulator," *ACM SIGARCH CAN*, 2011.
[25] N. Agarwal *et al.*, "GARNET: A Detailed On-Chip Network Model inside a Full-System Simulator," in *ISPASS*, 2009.
[26] A. Das *et al.*, "Reducing Off-Chip Miss Penalty by Exploiting Under-utilised On-Chip Router Buffers," in *ICCD*, 2020.
[27] A. Monemi *et al.*, "ProNoC: A Low Latency Network-on-Chip based Many-Core System-on-Chip Prototyping Platform," *Elsevier MICPRO*, 2017.
[28] M. Hussain and H. Guo, "Packet Leak Detection on Hardware-Trojan Infected NoCs for MPSoC Systems," in *ICCSP*, 2017.