Towards partial fulfillment for Undergraduate Degree Level Programme Bachelor of Technology in Computer Science and Engineering

An End Sem Project Evaluation Report on:

Detection of Hardware Trojan in On-Chip Network Using Machine Learning

Prepared by:

U21CS001 Rushil Jariwala

U21CS020 Bhumi Agrawal

U21CS057 Vedant Surti

U21CS064 Param Shah

Class: B.Tech. IV (Computer Science and Engineering) 7th Semester

Year: 2024-2025

Guided by: Dr. Anugrah Jain

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SARDAR VALLABHBHAI NATIONAL INSTITUTE OF TECHNOLOGY,
SURAT - 395007 (GUJARAT, INDIA)

Student Declaration

This is to certify that the work described in this project report has been actually carried out and implemented by our project team consisting of

Sr.	Admission No.	Student Name
1	U21CS001	Rushil Jariwala
2	U21CS020	Bhumi Agrawal
3	U21CS057	Vedant Surti
4	U21CS064	Param Shah

Neither the source code there in, nor the content of the project report have been copied or downloaded from any other source. We understand that our result grades would be revoked if later it is found to be so.

Signature of the Students:

Sr.	Student Name	Signature of the Student
1	Rushil Jariwala	
2	Bhumi Agrawal	
3	Vedant Surti	
4	Param Shah	

Certificate

This is to certify that the project report entitled **Detection of Hardware Trojan in On-Chip Network Using Machine Learning** is prepared and presented by

Sr.	Admission No.	Student Name
1	U21CS001	Rushil Jariwala
2	U21CS020	Bhumi Agrawal
3	U21CS057	Vedant Surti
4	U21CS064	Param Shah

Final Year of Computer Science and Engineering and their work is satisfactory.

SIGNATURE:

Supervisor/s JURY HEAD OF DEPARTMENT

Certificate of IPR

This is to certify that the project report entitled **Detection of Hardware Trojan in On-Chip Network Using Machine Learning** is prepared and presented by following students for final year *Computer Science and Engineering*

Sr.	Admission No.	Student Name
1	U21CS001	Rushil Jariwala
2	U21CS020	Bhumi Agrawal
3	U21CS057	Vedant Surti
4	U21CS064	Param Shah

Dr. Anugrah Jain
Department of Computer Science and Engineering

Acknowledgment

We wish to express our sincere gratitude and appreciation to several individuals and institutions whose support and guidance have been instrumental in the completion of this final year project on Detection of Hardware Trojan in On-Chip Network Using Machine Learning.

First and foremost, We extend our deepest thanks to Dr. Anugrah Jain for his invaluable insights, unwavering encouragement, and the wealth of knowledge he shared throughout this endeavor. His guidance has been a beacon, illuminating our path in the intricate world of On-Chip Networks.

We would also like to acknowledge the support of Head of Department(HOD) Dr. Mukesh Zaveri, professors and mentors at Sardar Vallabhbhai National Institute of Technology (SVNIT), Surat. Their teachings and expertise have been instrumental in shaping my understanding of the subject matter. Their dedication to fostering a spirit of inquiry and research has been a constant source of inspiration.

Our gratitude is also extended to the academic institutions and libraries, including SVNIT, that provided access to extensive resources, allowing us to delve into the depths of On-Chip Networks.

Furthermore, We appreciate the encouragement and patience of our peers and colleagues who provided their insights and feedback during the course of out research.

Last but not least, our heartfelt thanks go to our friends and families who stood by us with unwavering support and understanding, making this academic pursuit at SVNIT a truly enriching experience.

Rushil Jariwala
Bhumi Agrawal
Vedant Surti
Param Shah

Abstract

With the increasing complexity in ICs and the globalization of semiconductor supply chains, hardware security has become a greater concern. Hardware trojans, which are malicious changes in circuitry, threaten the integrity and security of Network-on-Chips (NoCs). This project focuses on detecting Hardware trojans within an 8x8 mesh-based Network-On-Chip architecture using simulation data generated through the Garnet Simulator. In NoCs, interconnected IP cores present vulnerabilities that adversaries can exploit to inject trojans, potentially causing system performance issues or leaking sensitive information. The simulated communication traffic for an 8x8 mesh network is created using the Garnet 2.0 cycle-accurate simulator, which models and emulates NoC architectures. The output of each simulation includes various metrics and features like packet latency, hop count, and buffer utilization, which could indicate anomalous behavior introduced by a trojan. We apply advanced machine learning models in the detection phase, effective in handling high-dimensional data to identify normal and trojan-affected traffic patterns. These models use sophisticated techniques to optimize classification accuracy, providing robust predictions by analyzing network traffic features. By leveraging these algorithms, we aim to detect anomalies with high precision and reliability, ensuring accurate identification of trojan-infected components. The goal of this project is to determine whether an NoC contains hardware trojans. By training our models on simulation data and testing them, we aim to improve the precision and recall of trojan detection, minimizing false positives and false negatives. This approach could significantly enhance the security of on-chip networks in modern IC designs.

Contents

List of Figures

1 Introduction

Network-on-Chip (NoC) architectures have become essential in modern System-on-Chip (SoC) designs for enabling efficient communication among various cores and IP blocks in recent years. Nonetheless, the growing dependence on international supply chains and external Intellectual Property (IP) cores has brought about fresh security risks, such as Denial-of-Service (DoS) attacks. Attacks have the potential to greatly interfere with system performance by inundating the NoC with malicious traffic, leading to congestion and preventing authorized access to shared resources. Conventional approaches for identifying and controlling DoS attacks, like observing traffic timing or packet arrival frequencies, usually rely on predictable traffic behaviors. Although successful in stable settings, these techniques face challenges in adjusting to the changing traffic patterns observed in real-world situations with different application assignments and runtime circumstances. Also, current methods like identifying regular behavior and detecting anomalies usually do not consider intricate changes during runtime, resulting in subpar detection accuracy. In the field of NoC-based SoCs, machine learning methods have recently become a hopeful option for detecting attacks in real-time. ML methods can categorize network traffic in real-time by learning from normal and attack traffic, allowing them to detect abnormal behaviors signaling a potential DoS attack. These models are able to use different features obtained from NoC traffic, like flit traversal patterns, hop counts, and packet types, to accurately differentiate between legitimate and malicious activities.

1.1 Application

Many areas that rely on NoC-based systems for crucial processing and communication can use the suggested machine learning-based methodology for DoS detection. This technique effectively detects and mitigates harmful traffic patterns that might interfere with operations in fields where maintaining performance and security is critical, such industrial automation, automotive systems, and aerospace. In cloud and high-performance computing environments, where DoS assaults against NoCs can severely degrade system performance, it is equally advantageous. The framework helps safeguard against hardware-level vulnerabilities in consumer electronics like smartphones and Internet of Things devices, assuring steady performance and maintaining user experience. Additionally, the method improves data availability and integrity in secure computing platforms, such military and financial systems, offering a strong and fast defence against sophisticated assaults against NoC systems. Because of its adaptability and versatility, the suggested technique may be used to a wide range of domains,

enhancing the overall security of NoC-based systems.

1.2 Motivation

The rapid progress in semiconductor technology, coupled with the globalization of the hardware supply chain, has made integrated circuits increasingly susceptible to security threats, including hardware trojans. Hardware trojans are malicious modifications made at the circuit level during design or manufacturing, which can lead to significant failures or security breaches in sensitive applications. On-chip networks, which act as the communication backbone in multi-core and system-on-chip architectures, are especially vulnerable, as even small malicious changes can compromise data integrity and impact the overall functionality of the system.

The motivation for this project stems from the need to improve the security of on-chip networks by effectively detecting these hardware trojans. Traditional detection methods, such as physical inspection and functional testing, have limitations in scalability, cost, and detection accuracy. As integrated circuits become more complex, there is a growing need for automated and intelligent methods that can efficiently and accurately detect malicious modifications. This project seeks to use machine learning to develop a more robust and scalable solution for hardware trojan detection, ultimately enhancing security in modern electronic systems used in critical fields like defense, healthcare, and finance.

1.3 Objective

The main goal of this project is to create an effective approach for detecting hardware trojans in on-chip networks using machine learning techniques. The objectives targeted in this project :

1. Analyze existing approaches for hardware Trojan detection in Networks-on-Chip (NoC).
2. Simulate a Distributed Denial of Service (DDoS) attack in a 2D mesh NoC employing XY routing.
3. Generate a comprehensive dataset for hardware Trojan detection in NoC environments.
4. Develop a machine learning model to detect hardware Trojans in NoC architectures.

5. Develop a machine learning model to localize routers affected by hardware Trojans in NoC.

1.4 Organization of Report

The report is structured into five chapters. In Chapter 2, we present a detailed literature survey, analyzing the relevant works and studies that form the foundation of our research. Chapter 3 outlines the proposed work, describing the design and methodologies employed in our solution architecture. Chapter 4 details the simulations and experiments conducted, highlighting the results and their implications. Finally, Chapter 5 discusses the future directions and potential improvements that we plan to implement as the project progresses.

2 Literature Survey

The literature review presents an overview of recent research addressing these challenges. Studies have explored the vulnerabilities in NoC architectures and proposed advanced detection and mitigation techniques, such as machine learning-based approaches and dynamic network adaptations. The following sections summarize key findings and methodologies related to detecting and mitigating hardware trojans and DoS attacks in NoC-based systems. These works lay the foundation for understanding current approaches to safeguarding modern chip architectures and form the basis for the proposed methodology in this research.

2.1 Overview of System-on-Chip (SoC) and Network-on-Chip (NoC)

2.1.1 System-on-Chip (SoC)

An integrated circuit, or SoC, is a single chip that contains every part required for a computer or electrical system. Typically, it consists of parts like memory, I/O controllers, processors (CPU/GPU), and other peripherals[13].

- **Functionality:** SoCs are built to run particular applications, and the arrangement of integrated circuits on the chip determines how functional the device is.
- **Communication:** A bus-based architecture or crossbar connection is typically used to facilitate communication between various components (such as the CPU and memory).
- **Application:** Where space and power efficiency are crucial, SoCs are frequently utilised in consumer electronics, embedded systems, and mobile devices.

2.1.2 Network-on-Chip (NoC)

NoCs are communication subsystems inside System-on-a-Chip (SoC) that transport data across various IP cores or components over a network[13].

- **Functionality:** NoCs replace conventional bus-based designs with packet-switched networks to enable communication between various cores and peripherals inside a SoC.
- **Communication:** Network interfaces, switches, and routers are used to transport data, enabling concurrent data transfers and better scalability.

- **Application:** High-performance computing systems, multi-core SoCs, and applications requiring effective on-chip communication are common uses for NoCs.

2.2 Layered Architecture of Network-on-Chip (NoC)

A Network-on-Chip (NoC) architecture is a layered communication framework that ensures efficient and reliable data transfer between components in a System-on-Chip (SoC). These layers provide abstraction, modularity, and scalability, facilitating seamless integration of various functionalities.

- **Application Layer:** The application layer is responsible for defining and scheduling the communication and computation tasks within the NoC [2].
  - **Key Functions**:
    * Breaks down target applications into a set of communication and computation tasks.
    * Optimizes performance metrics such as energy efficiency, communication latency, and throughput.
    * Handles task mapping and communication scheduling, which are critical for achieving the desired tradeoff between power and performance.
- **Transport Layer:** The transport layer manages data flow and prevents congestion in the network. It ensures that packets are delivered efficiently without buffer overflow or excessive delays [2].
  - **Key Functions**:
    * Implements flow control and congestion management to avoid deadlocks or livelocks.
    * Ensures low packet latency and high throughput.
- **Network Layer:** The network layer defines the topology of the NoC and handles routing of packets between processing elements [2].
  - **Key Functions**:
    * Determines the network topology, which impacts scalability, performance, and cost.
    * Routes packets using algorithms tailored to the application and topology.
    * Resolves contentions when multiple packets request the same route.

- **Data Link Layer:** The data link layer ensures reliable communication between adjacent nodes by handling error detection and correction [2].
  - **Key Functions**:
    * Detects and corrects errors caused by noise or faults in the physical layer.
    * Increases the reliability of data transmission over a link.
- **Physical Layer:** The physical layer deals with the actual hardware transmission of data in the form of electrical or optical signals [2].
  - **Key Functions**:
    * Ensures signal integrity over interconnects.
    * Handles issues like electrical noise, crosstalk, and electromagnetic interference (EMI).

2.3 XY Routing

XY Routing is a deterministic routing algorithm commonly used in 2D mesh Network-on-Chip (NoC) architectures. It is based on a simple principle where packets are routed first in the X (horizontal) direction and then in the Y (vertical) direction until they reach their destination. This routing scheme is highly efficient in regular mesh networks because it ensures deadlock-free communication and avoids unnecessary complexity in the routing decisions[11].

In the XY routing algorithm, when a packet is injected into the network, the router first compares the destination's X-coordinate with its current position. If the X-coordinate of the destination is greater than the current router, the packet is routed to the east; if it is smaller, it is routed to the west. Once the X-coordinate matches the destination, the Y-coordinate is compared, and the packet is routed either north or south accordingly. This simple two-phase approach ensures that packets always follow a minimal path without any backtracking[11].

One of the major advantages of XY routing is its predictability and simplicity, making it easy to implement in hardware with minimal control logic. It also eliminates the possibility of cyclic dependencies between resources, which can lead to deadlock, a major concern in network routing. However, since XY routing is deterministic, it may suffer from poor load balancing in highly congested networks, as it always follows the same path for a given source-destination pair. This can lead to hotspots in the network, where certain links become overused while others remain underutilized[11].

2.4 Traffic Patterns

Traffic patterns are of prime importance considering the evaluation of NoC performance to understand the behavior of the interconnection network under different types of load. Garnet gem5 supports several traffic patterns that simulate a variety of communication scenarios within NoC environments. The following four kinds of traffic patterns have been used in this simulation: Uniform Random, Bit Complement, Tornado, and Transpose. Each is explained in detail below[8].

1. **Uniform Random Traffic**

   Definition: The Uniform Random traffic pattern represents that every source node sends packets to a randomly selected destination node with equal probability.

   Characteristics:

   - Any node has an equal opportunity to communicate with any other node in a network.
   - No spatial or temporal locality is enforced, making it a useful pattern to assess the network's capability to handle arbitrarily varying communication patterns.
   - Creates a balanced load in the network, but due to its randomness, may stress routing algorithms.

   Purpose:

   Uniform Random is usually used as a baseline since it exercises the network uniformly, providing insight into overall NoC throughput, latency, and fairness for a generic communication load.

2. **Bit Complement Traffic**

   Definition: In the Bit Complement pattern, each source node sends packets to a destination node with an address which is the bitwise complement of the source node address. For example, if the source node address is 1010, the address for its destination counterpart will be 0101.

   Characteristics:

   - Highly structured, with deterministic behavior.
   - Provides long-distance communication, which is generally needed in larger networks.
   - Creates significant pressure on certain links and paths in the network, exposing possible bottlenecks.

Purpose:

Worst-case analysis for routing and network congestion can be done through this pattern. It provides insight into how the NoC performs under highly structured traffic loads, which are most likely to cause hotspots and contention in network portions.

3. **Tornado Traffic**

   Definition: In the Tornado traffic pattern, every source node sends packets to a destination node that is a fixed number of nodes away in a circular manner. For example, consider a network with nodes numbered from 0 to N-1. If the offset is k, node i sends packets to node $(i + k) \% N$.

   Characteristics:

   - Creates a systematic shift in traffic that simulates circular or modular communication.
   - Causes a concentration of traffic on certain links, helping identify network congestion and imbalance in load distribution.

   Purpose:

   The Tornado traffic pattern is used to analyze the performance of the NoC concerning patterns that can cause directional and link-based contention. It is beneficial for capturing performance related to load balancing, link utilization, and fairness in communication.

4. **Transpose Traffic**

   Definition: The Transpose traffic pattern assumes the network to be constructed of a 2D grid structure, where every source node at coordinate (i, j) sends packets to a destination node at coordinate (j, i).

   Characteristics:

   - Highly structured and dependent on the grid topology.
   - Simulates communication that often occurs in matrix transposition operations in parallel computing tasks.
   - Can result in diagonal traffic across the network, with hotspots arising along some dimensions.

   Purpose:

   Transpose traffic is generally used for the performance evaluation of NoC when the applications include operations with matrices or any other tasks involving

systematic data rearrangement. It also stresses routing algorithms concerning diagonal broadcasting and effective load distribution.

These patterns represent a balanced mix of random, structured, and application-specific traffic scenarios, ensuring the NoC is rigorously tested for general-purpose and domain-specific workloads. Using these four traffic patterns, we ensure the simulation captures a wide spectrum of potential network behaviors, providing insights into the design's robustness, efficiency, and scalability. This comprehensive approach ensures confidence in the network's performance under diverse operating conditions.

2.5 Attacks in NoC

2.5.1 Denial-of-Service Attack in Network-on-Chip Architecture

DoS attacks are considered one of the major security threats in different NoC architectures, as they can degrade systems' performances by overloading networks with useless traffic. Generally, in the context of a NoC, a DoS attack consists of an IP core or node intentionally injecting into the network an excessive number of useless packets. The goal of the attack is to flood the network resources, which involve the routers and communication links, to deny the service to the legitimate traffic coming from other cores or components.

How Does a NoC DoS Attack Work?

- **Flooding with Malicious Traffic:** A compromised or malicious IP core generates an abnormally huge quantity of flits or packets. These are injected at a very fast rate inside the NoC, which saturates the routers, thereby causing congestion inside the network. This results in increased packet traversal time and increased buffer occupancy in the routers.
- **Router Buffer Overload:** Each of the routers in the NoC has a limited amount of buffer capacity, and the excessive traffic inside consumes those buffers, thereby resulting in a saturated router. The queue will be blocked for other valid packets from the different cores or may get dropped due to the unavailability of the buffer space.
- **Network Congestion:** Malicious traffic saturates links of the NoC, and on-path routers become hotspots. In this way, congestion diffuses, packet delays increase, and dropping rates increase; the overall performance of the whole system will

presumably drop considerably. The DoS attack in NoCs also targets critical components by memory controllers or cache controllers by issuing spurious requests. This blocks not only access requests to logical memories but starves the system of resources necessary for processing tasks and thus causes widespread failures at the system level.

The impact of Denial-of-Service (DoS) attacks on Network-on-Chip (NoC) systems is significant, as they cause severe delays in communication between processing elements, memory controllers, and caches, resulting in increased latency. Flits and packets take longer to reach their destinations due to network congestion, leading to a substantial reduction in system throughput as the NoC struggles to handle legitimate traffic. In systems that rely on real-time applications, DoS attacks can cause deadline violations, preventing critical tasks from being executed on time, which can have disastrous consequences for safety-critical applications. Additionally, the increased network activity and congestion lead to higher power consumption, reducing the energy efficiency of the NoC and making it less effective in terms of power usage[9].

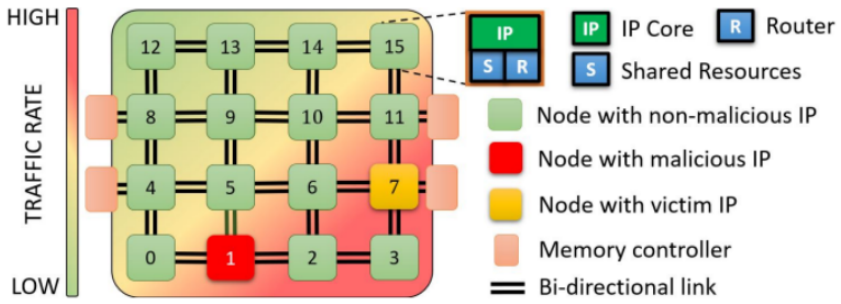

Fig 1. Example DoS attack from a malicious IP to a victim IP. The thermal map shows high traffic near the victim IP[14].

Figure 1, shown above depicts DoS attack from a malicious IP to a victim IP and the thermal map shows high traffic near the victim IP.

2.5.2 Delay Trojan Attack in Network-on-Chip (NoC)

In modern Network-on-Chip (NoC) architectures, a new form of hardware threat known as the Delay trojan (DT) poses a serious risk to system performance. Unlike traditional trojans that aim to leak data or disrupt system integrity, the Delay trojan is specifically designed to introduce random delays into the NoC by maliciously manipulating the packet processing times within specific routers. This malicious implant is placed within the NoC router and targets the input buffer of the router, introducing random delays to flits passing through it.

Mechanism of the Delay Trojan:

The Delay Trojan works by blocking the control signals that trigger routing operations for flits held in the input buffer of the infected router. During the attack:

- The trojan intermittently disables the routing operation for a certain number of cycles, creating random delays before allowing the packet to proceed through the router.
- These delays cause the virtual channels (VCs) within the router to remain engaged longer than usual, leading to network congestion as other flits and packets are delayed from accessing the VCs.
- The DT attack primarily affects latency-critical packets, such as those generated by cache miss requests, which results in significant degradation in system throughput and performance.

The Delay Trojan introduces significant challenges in Network-on-Chip (NoC) systems by holding packets in the input buffer for extended periods, causing unpredictable delays and contributing to network congestion. This delay results in a back-pressure effect, where neighboring routers also suffer from increased buffer occupancy, further exacerbating congestion throughout the network. As a result, the trojan creates congestion hotspots, making it difficult to differentiate between normal traffic delays and maliciously induced delays, leading to overall network performance degradation. The impact is particularly severe in latency-sensitive applications such as real-time video streaming, VoIP, and interactive gaming, where timely packet delivery is critical. The delays caused by the trojan can result in application stalling and timeouts, disrupting user experiences. Additionally, the Delay trojan increases the L1 cache miss penalty by delaying cache miss request packets, which prolongs memory access times, ultimately slowing down the entire system. This leads to more cache misses and a reduction in system efficiency, as critical memory operations are hindered by the trojan's interference[9].

2.6 Handling of attacks in NoC

2.6.1 Hardware based solution:

Mitigating the effects of Delay Trojan (DT) attacks in Network-on-Chip (NoC) architectures is essential for maintaining system performance and security. Delay trojans introduce random delays to packets, mimicking congestion and impacting latency-critical

applications. The Dynamic Adaptive Caging (DAC) framework is a key approach to countering this threat, comprising three phases: detection, caging, and re-routing. Detection is achieved by monitoring packet delays across routers, comparing metrics like Average Time per Router (ATR) and Time spent in Previous Router (TPR) to detect abnormal delays. If a Delay trojan is identified, the system isolates the infected router using a dynamic caging process, which communicates the threat to surrounding routers and prevents packet routing through the infected node. Once the cage is in place, packets are rerouted via alternate paths, ensuring they avoid the compromised router and minimizing overall delays. This adaptive method also manages routing deadlocks through a re-injection technique, maintaining smooth system operation[9].

2.6.2 Machine Learning based solution:

To counter Denial-of-Service (DoS) attacks in NoC architectures, machine learning-based detection is a highly effective strategy. The system undergoes offline training using normal and attack traffic data to create machine learning models capable of distinguishing between benign and malicious traffic patterns. These models, commonly based on algorithms like Gradient Boosting Machines (GBM) or XGBoost, are integrated into the system to provide real-time monitoring. Probes at various NoC routers collect traffic data, which the machine learning model analyzes to detect abnormalities that signal a DoS attack. The dynamic nature of machine learning allows the system to adapt to varying traffic conditions and application workloads, ensuring accurate detection even under unpredictable traffic scenarios. Once a DoS attack is detected, the system quickly isolates and mitigates the impact, ensuring minimal performance disruption while maintaining network security[14].

Given the focus of our project on mitigating the impact of DoS attacks in NoC architectures, the paper explores various features that are extracted from NoC traffic to enable effective detection of such attacks. These features, which are either directly obtained from NoC traffic data or engineered through feature transformation, are essential in training machine learning models that can differentiate between normal and attack patterns. Below is a list of features that have been considered for this purpose:

- **Outport (A):** Port used by the flit to exit the router.
- **Inport (B):** Port used by the flit to enter the router.
- **Cache Coherence Type (C):** Type of cache coherence packet.
- **Flit ID (D):** Unique identifier for each flit.

- **Flit Type (E):** Indicates if the flit is head, body, or tail of a packet.
- **Virtual Network (F):** The virtual network used by the flit.
- **Virtual Channel (G):** The virtual channel used for communication.
- **Traversal ID (H):** Groups all packet transfers related to a NoC traversal.
- **Hop Count (J):** Total number of hops between source and destination.
- **Current Hop (K):** Number of hops from the source to the current router.
- **Hop Percentage (L):** Ratio between current hop and total hop count.
- **Enqueue Time (M):** Time spent by the flit inside the router.
- **Packet Count (Decrementing) (N):** Number of flits arriving within a given time window (decrementing count).
- **Packet Count (Incrementing) (O):** Similar to decrementing count but incremented as flits arrive.
- **Max Packet Count (P):** Maximum number of flits transferred within a time window.
- **Packet Count Index (Q):** Product of incrementing and decrementing packet counts.
- **Port Index (R):** Combination of inport and outport values.
- **Traversal Index (S):** Combination of cache coherence type, flit ID, flit type, and traversal ID.
- **Cache Coherence-Virtual Network (T):** Combined index for cache coherence type and virtual network.
- **Virtual Network-Virtual Channel-Cache Coherence Index (U):** A more complex combination involving virtual network, virtual channel, and cache coherence type.

Feature Selection is a crucial step in optimizing the machine learning model's accuracy and complexity. While all these features were considered during the initial evaluations, some were discarded based on their importance in contributing to the model's predictive capabilities. Feature importance, as determined using the Gradient Boosting technique, was used to evaluate each feature's contribution to the model's decision-making process. Those features contributing less to accurate predictions were discarded to balance model complexity and performance.

Ultimately, the selected features were those with significant impact on the model, marked with asterisks in the final analysis:

- **Outport (A):** Important for tracking flit routing.
- **Virtual Channel (G):** Helps differentiate the communication channel used.
- **Traversal ID (H):** A key feature for tracking NoC traversals.
- **Hop Count (J):** Indicates the complexity of the routing path.
- **Enqueue Time (M):** Reflects traffic congestion at the router.
- **Packet Count Decrementing (N) and Incrementing (O):** Tracks traffic flow.
- **Max Packet Count (P) and Packet Count Index (Q):** Provide statistical insights into NoC traffic.

These selected features were found to be most effective in accurately distinguishing between normal and malicious NoC traffic, ultimately aiding in the successful detection of DoS attacks.

2.7 Summary

So, the research highlights the challenge of Delay trojans, which introduce random delays in the NoC that mimic congestion, affecting the performance of latency-sensitive applications. To detect and mitigate these threats, Delay Comparators (DCs) and Detection Units (DUs) are embedded in NoC routers to monitor packet delays. When anomalies are detected, the Dynamic Adaptive Caging (DAC) mechanism isolates infected routers and reroutes traffic to maintain system performance[9]. For Denial-of-Service (DoS) attacks, machine learning models are employed to differentiate between normal and malicious traffic in real-time. These models are trained during the design phase and use data from sensors at NoC routers to detect attacks during operation. The XGBoost classifier is found to be particularly effective in ensuring high accuracy and minimal false positives, all while maintaining network performance through the use of separate monitoring networks[14]. In conclusion, while both threats require adaptive defense mechanisms, we will focus on addressing DoS attacks using machine learning to improve detection and response in NoC architectures.

3 Proposed work for detection of DoS Attacks in NoC

The proposed algorithm leverages machine learning models to detect Denial-of-Service (DoS) attacks in Network-on-Chip (NoC) architectures through a two-phase approach: Training Phase and Testing Phase.

3.1 Training Phase

- **Data Collection:**
  - To generate training data, a NoC simulator, such as Garnet 2.0, is employed. Multiple scenarios are emulated, including both normal and attack conditions, using different network parameters and configurations. The simulation is executed using commands like:

```
anugrah@Anugrah-VirtualBox:~/Downloads/gem5-fa70478413e4650d0058cbfe81fd5ce362101994$
 ./build/NULL/gem5.debug configs/example/garnet_synth_traffic.py --num-cpus=64 --num-
dirs=64 --network=garnet2.0 --topology=Mesh_XY --mesh-rows=8 --sim-cycles=10000 --syn
thetic=uniform_random --injectionrate=0.05
```

  - This command generates network traffic data in the form of a stats file, capturing various performance metrics and traffic patterns for different injection rates.
- **Feature Extraction:**
  - Key features are extracted from the stats file. These features are pre-determined based on insights from previous research and include metrics such as packet latency, flit traversal time, and buffer utilization.
  - The extracted features are stored in a structured format to serve as input for the machine learning models.
- **Data Pre-processing:**
  - The collected data is pre-processed using normalization and scaling techniques to ensure consistency and enhance the performance of the ML models.
  - This step standardizes the data values, which helps mitigate the influence of varying scales and distributions of different features.

- **Model Training:**
  - Two machine learning models, Random Forest Classifier (RFC) and eXtreme Gradient Boosting (XGB), are trained using the processed dataset. The models learn to differentiate between normal and attack traffic patterns.
  - The trained models are then stored in a central security engine, which will be used for real-time classification during the runtime phase.

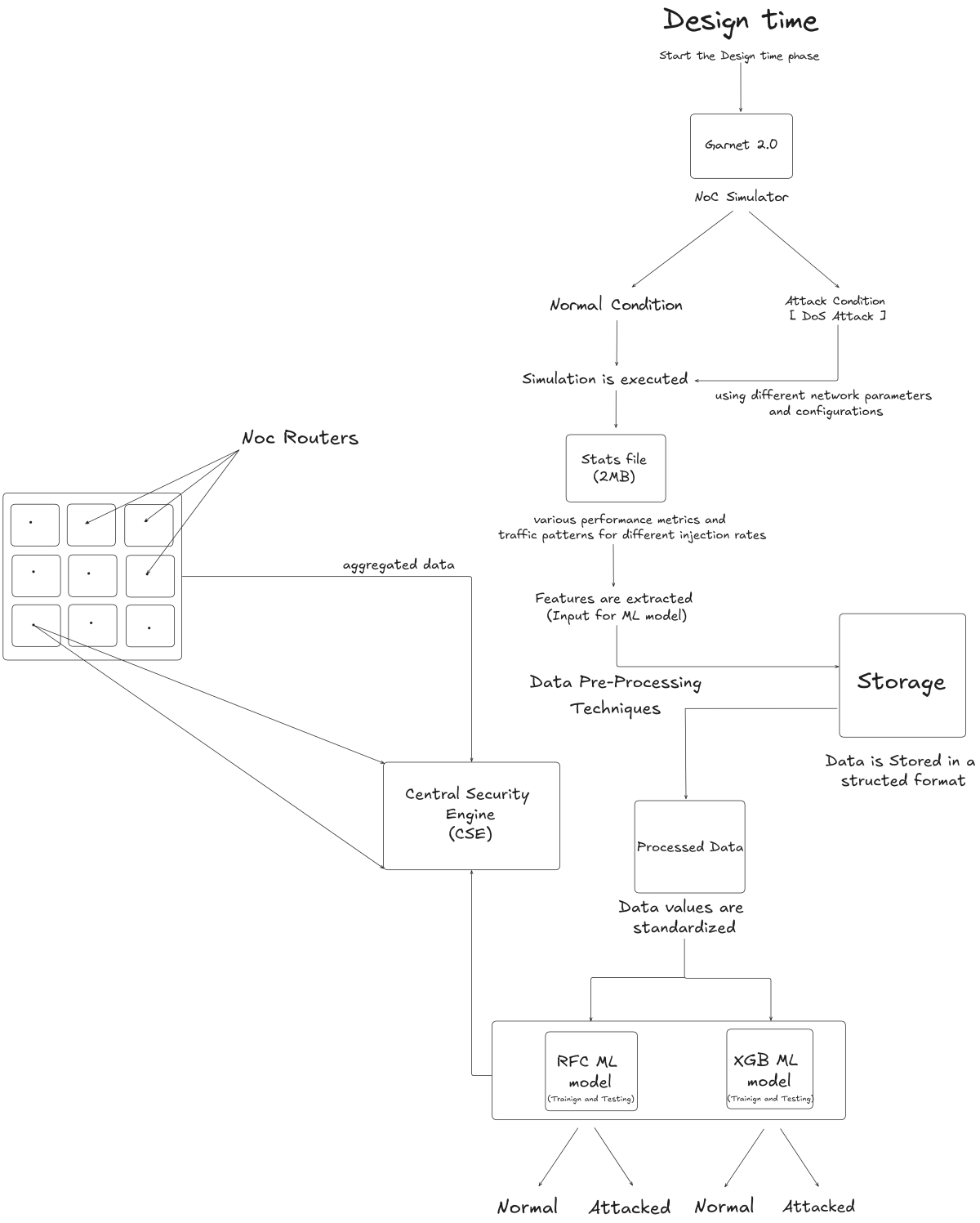

Fig 2. Workflow of the proposed Method

The Figure 2 depicts the workflow of the method that we have proposed.

3.2 Testing Phase

- **Feature Collection at Runtime:** During system operation, traffic data is continuously collected from NoC routers. This data is gathered for a specific time window to capture current network activity.
- **Data Transmission to Security Engine:** The collected runtime data is sent to the security engine, which houses the pre-trained RFC and XGB models.
- **Classification:** The security engine uses the stored models to classify the incoming traffic as either normal or indicative of a DoS attack. Predictions are based on the aggregated data for the given time window, ensuring that both transient and sustained attack patterns can be detected accurately.
- **Model Comparison and Selection:** The detection accuracy of both models is monitored and compared over time. The model with the highest accuracy is selected for real-time classification and alerting.

3.3 Outcome and Model Adaptation

- Once the optimal model is selected, it is used for real-time DoS detection. If required, the model can be re-trained or updated based on new patterns of traffic anomalies observed in the NoC, allowing the system to adapt to evolving attack vectors.

3.4 Working of Random Forest Classifier

The Random Forest algorithm works by following a structured process with various important steps involved:

- **Sampling of data (Bagging):** In the training stage, the algorithm picks random data subsets through bootstrap sampling, which involves sampling with replacement. This procedure guarantees that every decision tree is trained on a different edition of the dataset, promoting variety among the trees. In the scenario of identifying Hardware trojans in NoC architectures, the gathered data could include important performance metrics such as packet latency, hop count, and buffer usage. Examining these characteristics collectively can uncover unique trends that suggest the presence of trojan behavior[7].
- **Building a tree structure:** After sampling the data, the algorithm creates a sequence of decision trees. Every tree is formed by selecting a random subset of attributes, like hop count, latency, or packet size, to determine divisions at different points. This built-in uncertainty guarantees that every tree examines distinct parts of the data, leading to varied predictions. For detecting trojans in NoCs, each tree is able to examine different aspects of network traffic, allowing the Random Forest to detect a diverse range of potential abnormalities[7].
- **Voting Mechanism:** After the trees are constructed, each one produces its own forecast. During classification tasks, like identifying a trojan in the NoC, each tree contributes a vote for its anticipated class. The ultimate categorization relies on the most common choice among all trees. In regression situations, the result is determined by averaging the forecasts made by every tree. This voting technique prevents any one tree's extreme predictions or errors from overly impacting the overall outcome, making Random Forest a dependable and steady method for detecting anomalies[7].

This approach effectively leverages the strengths of machine learning to provide robust and adaptive DoS detection in NoC architectures, offering high accuracy with minimal runtime overhead.

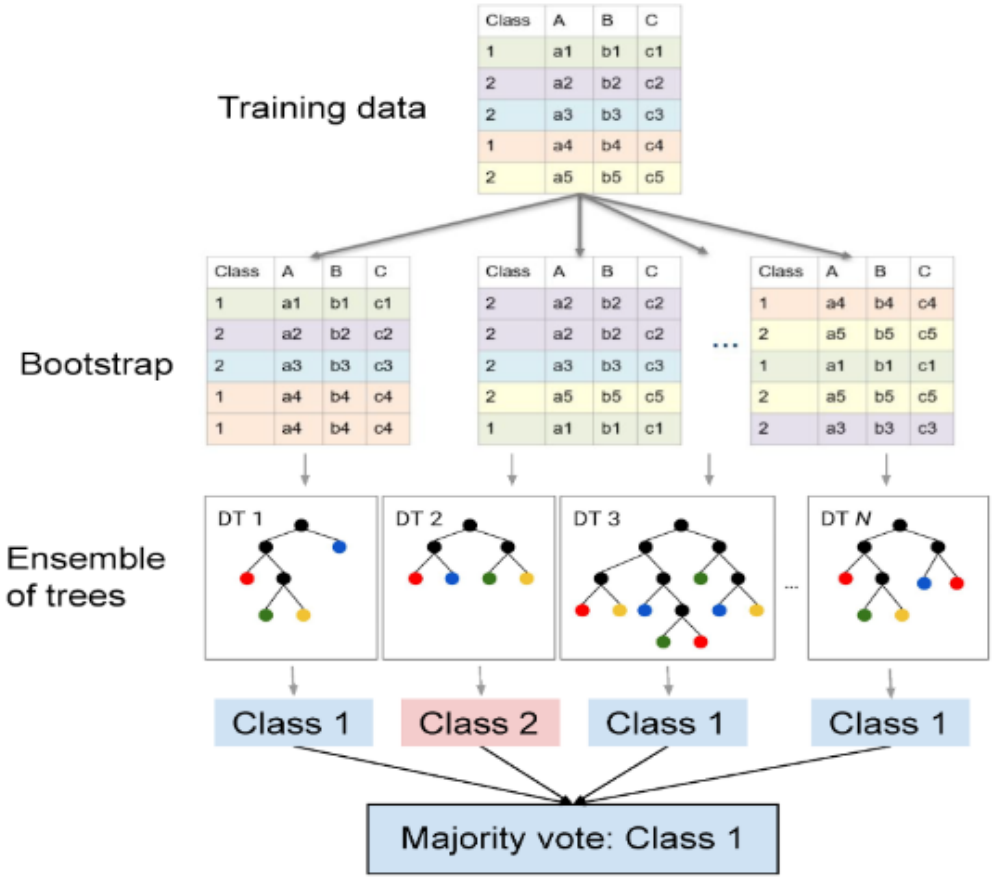

Fig 3. Internal Working of Random Forest Classifier[4]

Figure 3 depicts the Working of the eXtreme Gradient Boosting Algorithm (XGB) and the complete flow of it.

3.5 Working of the eXtreme Gradient Boosting Algorithm (XGB)

The Boosting algorithm generates new weak learners (models) and combines their predictions sequentially to enhance overall model performance. For instances where predictions are incorrect, the algorithm assigns greater weights to misclassified samples while assigning lower weights to those that are correctly classified. In the final ensemble model, weak learner models that demonstrate better performance receive higher weights. Importantly, Boosting does not alter previous predictors; instead, it focuses on correcting errors made by earlier predictors by learning from their mistakes[5].

Since Boosting follows a greedy approach, it is advisable to establish a stopping criterion—such as monitoring model performance (early stopping) or limiting the number of stages (e.g., tree depth in tree-based learners)—to avoid overfitting the training

data. The first implementation of the Boosting technique was called AdaBoost (Adaptive Boosting).

This structured approach allows for continuous improvement in predictive accuracy, making Boosting a powerful tool in machine learning applications.

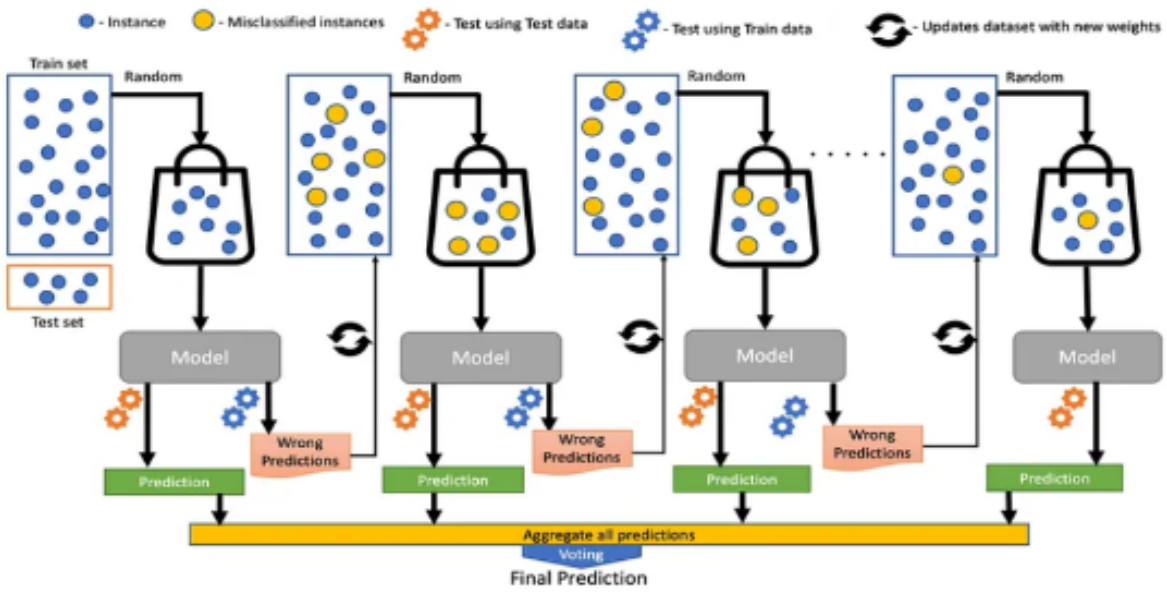

Fig 4. Internal Working of Boosting Algorithm[5]

As shown in Figure 4, how does boosting algorithm works internally

- Gradient Boosting
  Gradient boosting is a specific type of boosting algorithm that minimizes errors using a gradient descent approach, resulting in a model composed of weak prediction models, such as decision trees.

  The key distinction between traditional boosting and gradient boosting lies in how each algorithm updates the weak learners based on incorrect predictions. In gradient boosting, weights are adjusted according to the gradient, which indicates the direction of improvement in the loss function. This process employs an algorithm known as Gradient Descent, which iteratively refines the model by updating the weights to minimize loss. Here, "loss" typically refers to the discrepancy between predicted and actual values. For regression tasks, the Mean Squared Error (MSE) is commonly used as the loss metric, whereas for classification tasks, logarithmic loss is often employed[3].

- Gradient Boosting Process

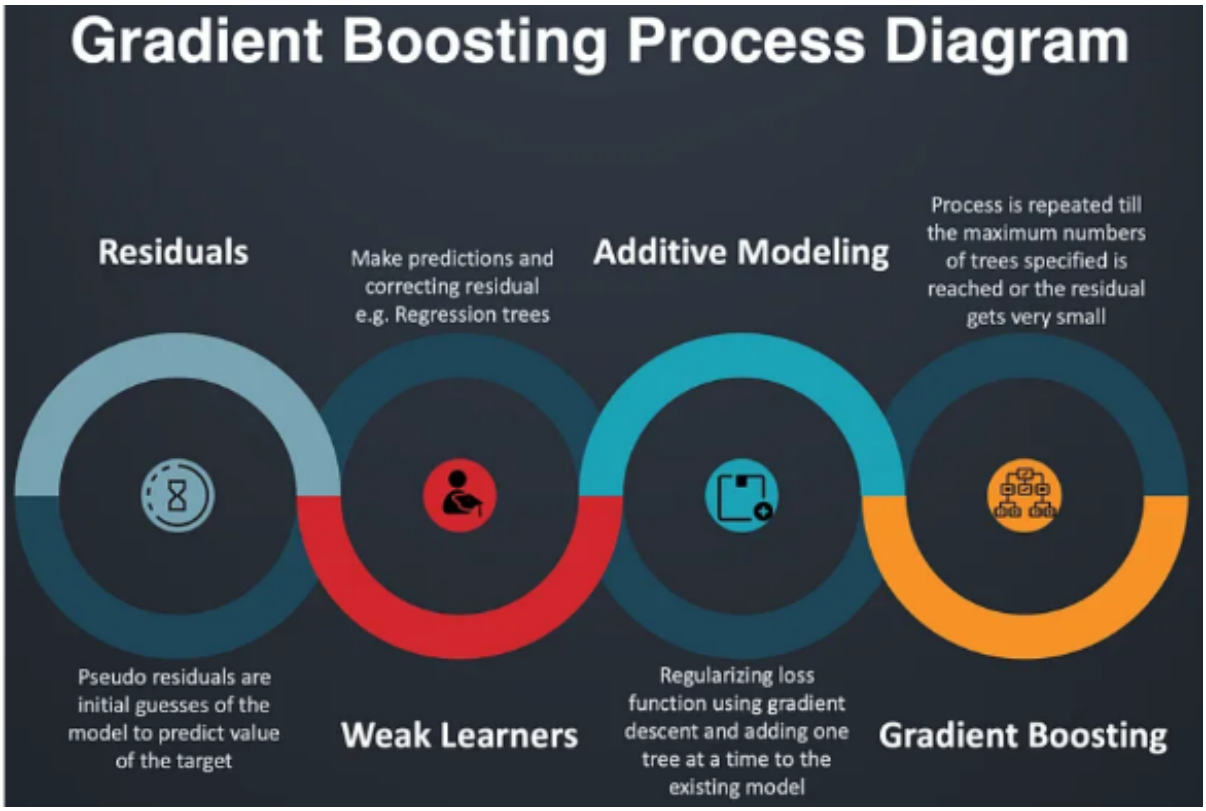

Fig 5. Gradient Boosting Process[5]

Figure 5 depicts the complete process of gradient boosting and what happens at each step.

Gradient boosting employs Additive Modeling, where a new decision tree is incrementally added to the existing model to minimize loss through gradient descent. The previously established trees remain unchanged, which helps mitigate the risk of overfitting. The output from the newly added tree is combined with the outputs of the existing trees until the loss is reduced below a certain threshold or a specified number of trees is reached. In mathematical terms, Additive Modeling involves decomposing a function into the sum of multiple subfunctions. From a statistical perspective, it can be viewed as a regression model where the response variable y is expressed as the total of the individual effects contributed by predictor variables x[5].

4 Simulation and Results

4.1 Simulation

In this section, we detail the process of simulating the system using the Garnet simulator to collect essential data. The simulator enabled us to replicate network-on-chip (NoC) scenarios and generate data reflective of real-world conditions. From this data, key features were extracted to capture patterns and characteristics necessary for analysis. These features were then utilized to train machine learning models, forming the basis for detecting and mitigating potential issues in the system.

4.1.1 Data Collection of Normal Scenario

The command to simulate a 8 x 8 2D mesh NoC network with XY routing and uniform random traffic and injection rate=0.05

```
anugrah@Anugrah-VirtualBox:~/Downloads/gem5-fa70478413e4650d0058cbfe81fd5ce362101994$
 ./build/NULL/gem5.debug configs/example/garnet_synth_traffic.py --num-cpus=64 --num-
dirs=64 --network=garnet2.0 --topology=Mesh_XY --mesh-rows=8 --sim-cycles=10000 --syn
thetic=uniform_random --injectionrate=0.05
```

```
anugrah@Anugrah-VirtualBox:~/Downloads/gem5-fa70478413e4650d0058cbfe81fd5ce362101994$
 ./build/NULL/gem5.debug configs/example/garnet_synth_traffic.py --num-cpus=64 --num-
dirs=64 --network=garnet2.0 --topology=Mesh_XY --mesh-rows=8 --sim-cycles=10000 --syn
thetic=uniform_random --injectionrate=0.05
warn: CheckedInt already exists in allParams. This may be caused by the Python 2.7 co
mpatibility layer.
warn: Enum already exists in allParams. This may be caused by the Python 2.7 compatib
ility layer.
warn: ScopedEnum already exists in allParams. This may be caused by the Python 2.7 co
mpatibility layer.
gem5 Simulator System.  http://gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 version 20.0.0.3
gem5 compiled Sep 20 2023 15:38:15
gem5 started Sep 30 2024 12:09:36
gem5 executing on Anugrah-VirtualBox, pid 3914
command line: ./build/NULL/gem5.debug configs/example/garnet_synth_traffic.py --num-c
pus=64 --num-dirs=64 --network=garnet2.0 --topology=Mesh_XY --mesh-rows=8 --sim-cycle
s=10000 --synthetic=uniform_random --injectionrate=0.05

Global frequency set at 1000000000 ticks per second
warn: rounding error > tolerance
```

```
info: Entering event queue @ 0.  Starting simulation...
warn: Replacement policy updates recently became the responsibility of SLICC state ma
chines. Make sure to setMRU() near callbacks in .sm files!
Exiting @ tick 10000 because Network Tester completed simCycles
```

Orchestration of Garnet 2.0

The **stats.txt** file, located in the gem5 directory, contains comprehensive details of the most recent simulation. The features, averaged across the network, will be extracted from this file for input into the machine learning model.

```
system.ruby.network.average_flit_latency  1091.616979
system.ruby.network.average_flit_network_latency   541.944139
system.ruby.network.average_flit_queueing_latency   549.672840
system.ruby.network.average_flit_vnet_latency |  431.519394
system.ruby.network.average_flit_vqueue_latency |  346.071924
system.ruby.network.average_hops             2.837721
system.ruby.network.average_packet_latency   847.357746
system.ruby.network.average_packet_network_latency   457.359216
system.ruby.network.average_packet_queueing_latency   389.998530
system.ruby.network.average_packet_vnet_latency |  431.519394
system.ruby.network.average_packet_vqueue_latency |  346.071924
system.ruby.network.avg_link_utilization     5.327400
```

```
system.ruby.network.avg_vc_load          |    1.048700     19.69%     19.69% |
   1.029800     19.33%     59.62% |    0.517200      9.71%     69.33% |    0.329700
   5.24%     92.81% |    0.215700      4.05%     96.85% |    0.167600      3.15%
system.ruby.network.avg_vc_load::total       5.327400
system.ruby.network.ext_in_link_utilization        12359
system.ruby.network.ext_out_link_utilization         9990
system.ruby.network.flit_network_latency |     1679905                          |
system.ruby.network.flit_queueing_latency |     1347258                           |
system.ruby.network.flits_injected       |        4310     34.35%     34.35% |
system.ruby.network.flits_injected::total        12547
system.ruby.network.flits_received       |        3893     38.97%     38.97% |
system.ruby.network.flits_received::total         9989
system.ruby.network.int_link_utilization         30925
system.ruby.network.packet_network_latency |     1679905                          |
system.ruby.network.packet_queueing_latency |     1347258                           |
system.ruby.network.packets_injected     |        4310     46.11%     46.11% |
system.ruby.network.packets_injected::total         9347
system.ruby.network.packets_received     |        3893     47.68%     47.68% |
system.ruby.network.packets_received::total         8165
```

Images of Stats file

4.1.2 Simulation of DDoS Attack

Network Configuration in Garnet2.0

The image represents the router labeling scheme for an 8×8 mesh network topology used in the Garnet gem5 simulator, a widely used structure in network-on-chip (NoC) systems. This mesh network consists of 64 routers arranged in a grid-like format with 8 rows and 8 columns, where each router is uniquely identified by a number from 0 to 63. The labeling follows a row-major order, starting from the bottom-left corner, with router 0 in the first row and first column, and router 63 in the last row and last column. Numbers increment sequentially from left to right within each row, and then continue to the next row above. For example, the bottom row is labeled 0 to 7, the

second row is labeled 8 to 15, and the top row is labeled 56 to 63. Each router connects to its neighboring routers in the north, south, east, and west directions, except for edge and corner routers, which only connect to their existing neighbors. This systematic labeling facilitates traffic mapping, performance analysis, and attack simulation, such as directing malicious traffic to specific routers (e.g., routers 37 and 44 targeting router 36).

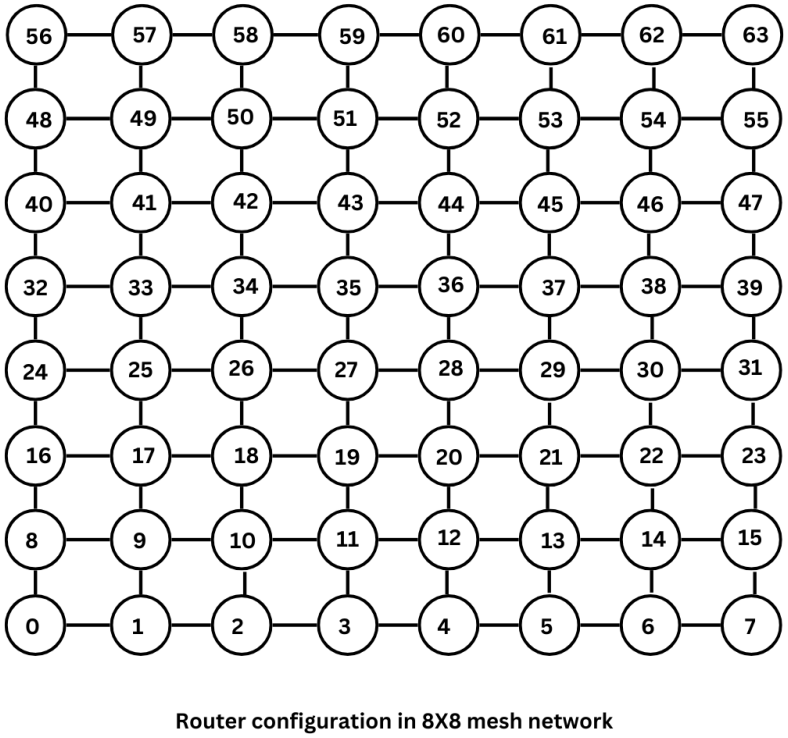

Fig 6. Network Configuration in 8X8 mesh network

Figure 6 depicts the network configuration and the attack simulation targets the Network-on-Chip (NoC) by introducing malicious behavior in specific routers. Routers with IDs **37** and **44** have been programmed to act as Trojan-infected nodes, with router **36** designated as the victim. These Trojan routers inject excessive traffic and specifically direct it towards the victim router, simulating a Denial-of-Service (DoS) attack in the NoC environment.

The attack disrupts the normal flow of traffic, creating localized hotspots and impacting overall network performance. This serves as a practical scenario for developing and validating machine learning-based Trojan detection mechanisms.

To simulate the Trojan attack, two specific code changes were implemented in the Garnet gem5 traffic generation logic:

1. **Forced Traffic Injection for Trojan Routers :** The first modification alters the traffic injection logic to ensure that routers with IDs **37** and **44**, designated as Trojan routers, always inject packets into the network. This is achieved by bypassing the normal traffic injection control, which is typically governed by a probabilistic condition based on the injection rate. The modified logic allows these routers to send packets unconditionally, irrespective of the configured injection rate:

```
    if (id == 37 || id == 44) {
    sendAllowedThisCycle = true;  // Always allow sending for attack routers
} else {
    // Normal injection rate control for other routers
    sendAllowedThisCycle = (trySending < injRate * injRange);
}
```

Code changes in GarnetSyntheticTraffic.cc

This change ensures that the Trojan routers can flood the network with traffic, creating a high load that disrupts normal operations.

2. **Targeted Destination Selection for Trojan Routers** The second modification ensures that all packets originating from routers 37 and 44 are directed to a single victim router, router 36. Normally, the destination for packets is determined dynamically based on the chosen traffic pattern, such as Tornado or Uniform Random. However, for Trojan routers, the logic was explicitly altered to override this behavior and target router 36:

```
if(source==37 || source==44){
    destination=36;
}
```

Code changes in GarnetSyntheticTraffic.cc

This ensures that all traffic generated by Trojan routers is concentrated on router 36, creating a hotspot at the victim router. This targeted traffic redirection simulates a Denial-of-Service (DoS) attack by overwhelming the victim router.

Impact of the Modifications These code changes simulate a twofold attack:

- **Traffic Overload:** Trojan routers (37 and 44) bypass normal injection rate controls, generating traffic at an abnormally high rate, which increases overall network load.
- **Localized Congestion:** By directing all packets to a single destination (router 36), the attack causes a hotspot, resulting in congestion, increased latency, and potential packet loss at the victim router.

These modifications provide a practical framework for simulating and studying Trojan attacks in NoC environments, offering insights into their impact and aiding in the development of detection mechanisms.

4.2 Data Exploration:

Data extraction:

average_flit_latency	average_packet_latency	average_packet_queueing_latency	average_flit_queueing_latency	packets_injected::total
1489.133595	2347.888915	2323.83061	1463.419285	193358
1208.42967	1524.120265	1485.939419	1147.696413	333782
1677.248187	1401.218356	1360.299591	1603.961215	437722
3523.406303	1811.388369	1768.158168	3437.494846	535695
4292.155057	1862.259725	1820.571014	4206.09561	632679
5047.986114	1903.571262	1862.529082	4961.539236	726424
5082.629383	1800.687543	1755.177698	4995.362501	817700
5598.117292	2246.96858	2185.954836	5490.759579	870680

Fig. 7 Data in Excel File

packets_received::total	flits_injected::total	flits_received::total	average_hops	avg_link_utilization	r
193258	405138	404918	4.762888	54.78344	
333505	702774	701813	4.927187	97.27756	
437299	818934	817411	4.9236	113.25736	
535189	902283	900441	4.921365	124.72348	
632033	980091	977912	4.942961	135.89306	
725740	1038304	1036160	4.974407	144.63322	
816846	1094176	1091864	5.006412	153.111	
869561	1119336	1116761	4.995722	156.3726	

Fig. 8 Data in Excel File

The Figure 7 and 8 represents performance metrics extracted from the Garnet gem5 simulator's stats file and converted into a structured .csv format for easier analysis. This tabular format will be used as input to our machine learning model for training and detection of anomalies or Trojan attacks in the network.

Dataset Description:

- **average_flit_latency**: The average time taken by a single flit (flow control unit) to traverse the network from the source to the destination, measured in cycles or seconds.
- **average_packet_latency**: The average time taken by a complete data packet to travel from its source to its destination, including transmission and queuing delays.
- **average_packet_queueing_latency**: The average time packets spend waiting in queues at the source or intermediate nodes before being transmitted to the next node.

- **average_flit_queueing_latency**: The average time individual flits spend waiting in queues within the network before being transmitted to the next node.
- **packets_injected_total**: The total number of data packets injected into the network by all sources during the simulation or operation.
- **packets_received_total**: The total number of data packets successfully delivered to their respective destinations.
- **flits_injected_total**: The total number of flits generated and injected into the network by all sources.
- **flits_received_total**: The total number of flits that reached their respective destinations successfully.
- **average_hops**: The average number of hops (network nodes or routers) a packet takes to travel from its source to the destination.
- **avg_link_utilization**: The average utilization percentage of the network links, indicating how efficiently the links are used during the operation.
- **routersXX.crossbar_activity** (e.g., routers60.crossbar_activity): Represents the activity level of the crossbars (switching elements) in the network's routers. Each router's crossbar activity indicates its workload and the data traffic it processes. The dataset includes crossbar activity for multiple routers, which collectively contribute to identifying network behavior under normal and attacked scenarios.
- **injection_rate**: The rate at which packets are injected into the network, usually expressed as a fraction of the network's maximum capacity.
- **victim_location**: The identifier or address of the router or node targeted as the victim in a simulated or real attack scenario.
- **no_of_trojan**: The number of malicious entities (Trojans) embedded in the network for simulating security attacks.
- **trojan_location_1**: The identifier or address of the first malicious Trojan node in the network.
- **trojan_location_2**: The identifier or address of the second malicious Trojan node in the network, if applicable.

Analyzing Simulation Results:

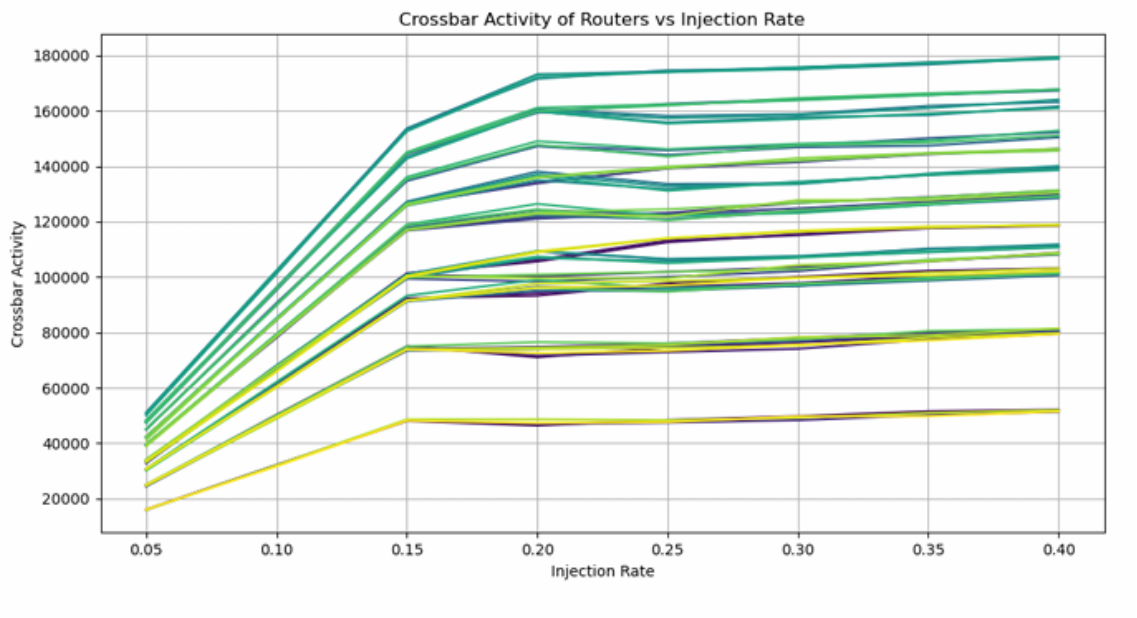

Fig 9. Crossbar Activity of Routers in normal scenario

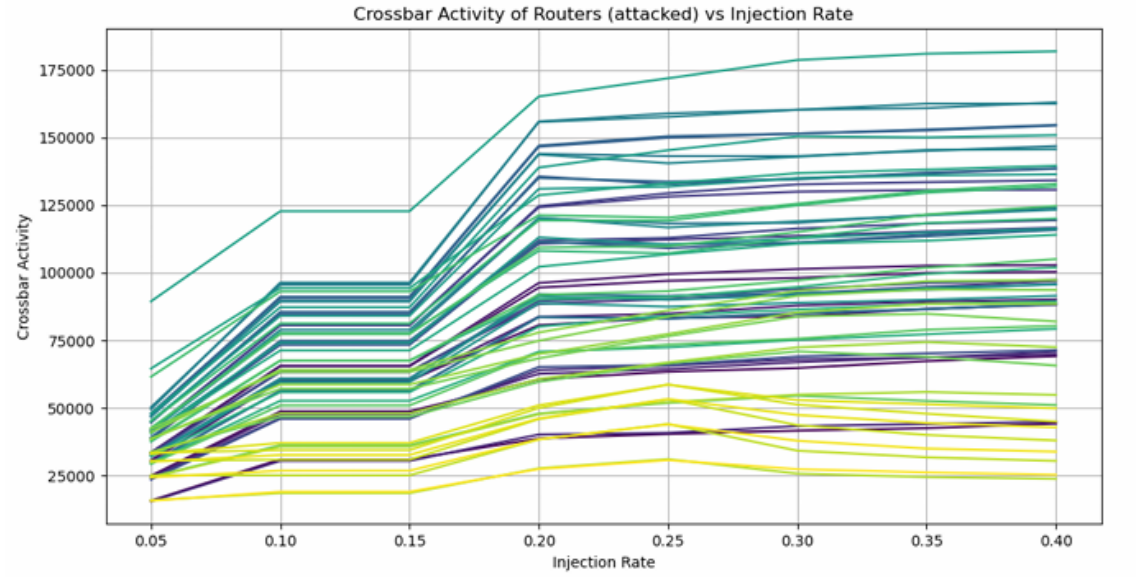

Fig 10. Crossbar Activity of Routers after attack

The Figure 9 represents the baseline crossbar activity under normal conditions, while Figure 10 captures crossbar activity in the presence of an attack. The comparison of these scenarios reveals distinct patterns that emerge during an attack. By studying these patterns, the crossbar activity of individual routers can serve as a potential feature for predicting the attacker router within the network. This approach emphasizes

the feasibility of using router-level crossbar metrics for proactive attack detection and enhanced network security.

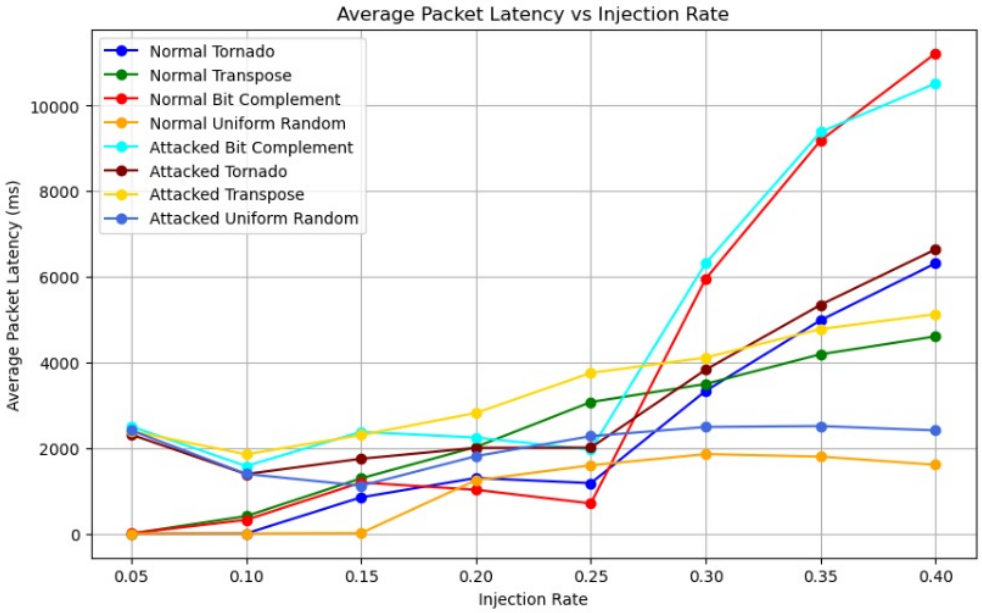

Fig 11. Average Packet Latency vs Injection Rate

In Figure 11, the average packet latency increases gradually as the injection rate rises, maintaining reasonable performance until higher rates when congestion starts to emerge. Under attack, the latency increases significantly at moderate-to-high injection rates, indicating severe performance degradation.

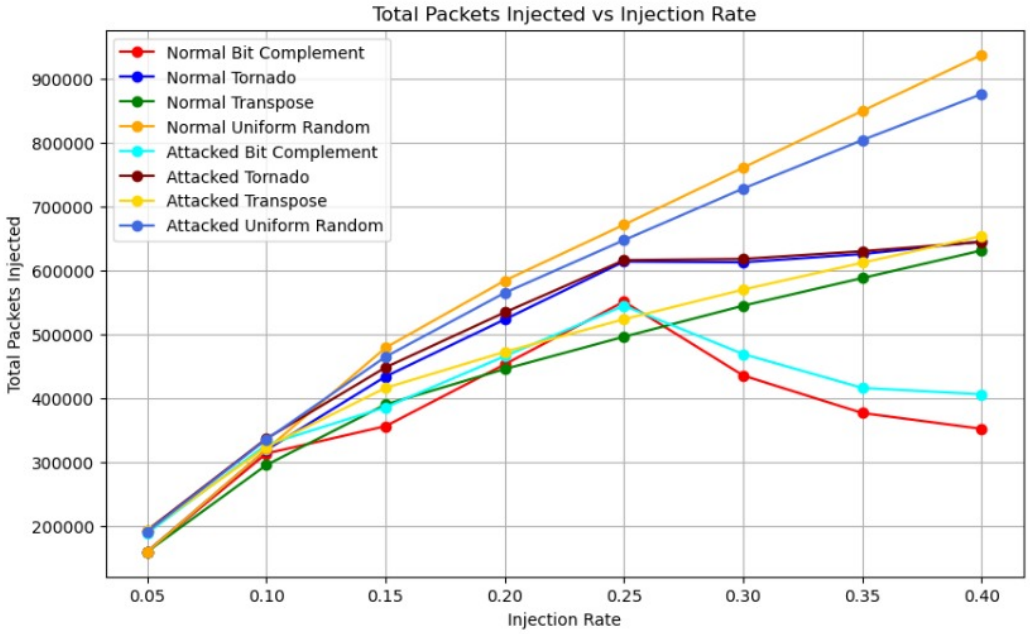

Fig 12. Total Packets Injected vs Injection Rate

In the Figure 12, the total number of packets injected into the network increases proportionally with the injection rate, showing efficient handling of traffic. However, under attack, the injected packets plateau or decline at higher injection rates, reflecting reduced network capacity and performance.

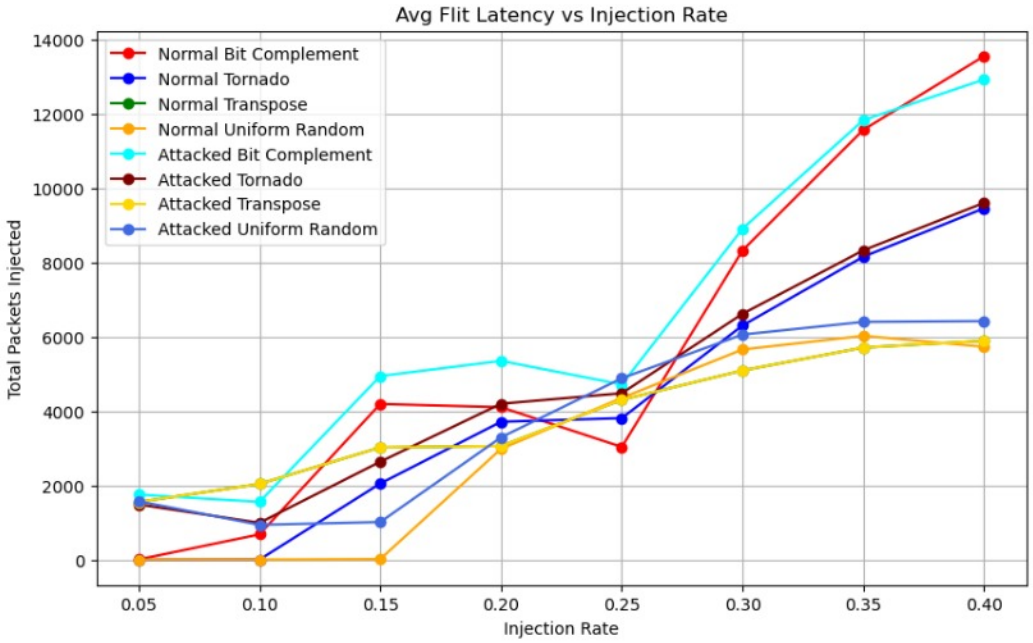

Fig 13. Average Flit Latency vs Injection Rate

In the absence of attacks as shown in Figure 13, the average flit latency rises steadily with the injection rate, showing predictable performance. When under attack, flit latency grows much faster, especially at higher injection rates, revealing the network's vulnerability to increased congestion.

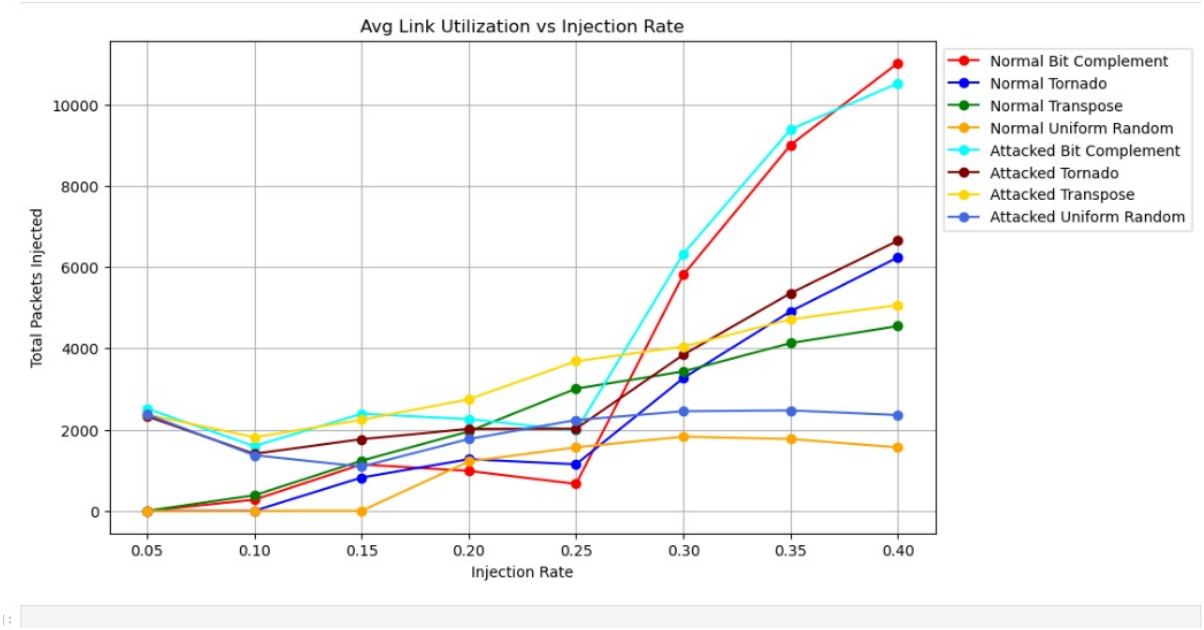

Fig 14. Average Link Utilization vs Injection Rate

Under normal conditions as shown in Figure 14, link utilization increases consistently with injection rate, indicating balanced and efficient resource usage. In the attacked scenario, utilization shows irregular behavior, peaking and then dropping at higher injection rates, signifying disrupted and inefficient network operations.

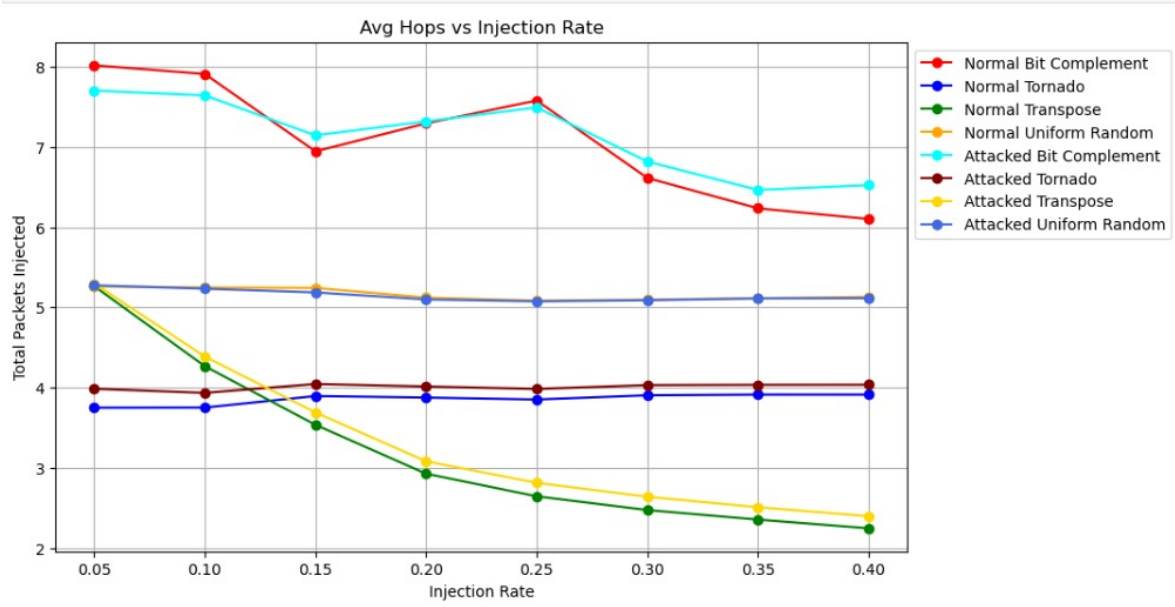

Fig 15. Average Hops vs Injection Rate

In the normal scenario as depicted in Figure 15, the average number of hops remains relatively stable or decreases slightly with increasing injection rate, reflecting efficient routing in the network. Under attack, the number of hops for certain patterns, es-

pecially "Attacked Bit Complement" and "Attacked Uniform Random," fluctuates and decreases more significantly at higher injection rates, indicating disrupted routing and reduced network efficiency due to the attack.

4.3 Implementation of Machine Learning Models

1. **Preparation Phase:** The simulation revolves around detecting trojan attacks within a network. The dataset, generated using the GarNET simulator, was constructed to represent diverse scenarios, including regular and attack traffic. Data from multiple excel files was integrated into a single unified dataset to ensure comprehensive coverage. There were a total of 7 attacks performed, and each file captured distinct network behaviors, enriching the analysis with varied conditions.

   The merging process created a consolidated dataset that eliminated redundancy and preserved the integrity of the data. This unified view was instrumental in simplifying downstream processing, allowing for seamless exploration, feature engineering, and model training. By leveraging descriptive statistics, we ensured the dataset met the quality standards required for machine learning tasks. This step also provided initial insights into data distribution, trends, and anomalies, which are essential for preparing robust predictive models.

2. **Data Processing and Analysis:** The first step in analyzing the dataset involved labeling each instance with a binary indicator denoting the presence or absence of a trojan attack. This classification, achieved by adding a new column (trojan_present), a binary field, provided a clear differentiation between normal and abnormal data. This labeling was critical for supervised machine learning, enabling the models to distinguish between benign and malicious activities effectively.

   Exploratory Data Analysis (EDA) played a significant role in understanding the dataset. Using statistical summaries such as mean, median, and standard deviation, we identified trends and potential outliers. For instance, attack scenarios exhibited significant deviations in certain network parameters compared to normal traffic, reinforcing the need for precise detection methods. We utilized a combination of libraries such as NumPy for numerical computations, Matplotlib for data visualization, and specific machine learning frameworks that were relevant to our analysis.

3. **Model Training and Testing for Binary Classification:**
   The model training and testing process is a crucial part of this simulation, designed to ensure that the predictive models generalize well to unseen data. This

phase involves transforming raw data into training and testing sets, selecting appropriate models, fine-tuning hyperparameters, and analyzing performance metrics to validate the model's effectiveness.

- **Data Splitting**
  To prepare the dataset for training and testing, the feature columns (denoted as the X vector) and the target column (y vector) were defined. The X vector included all input features relevant for the prediction, while the y vector contained the binary target column, trojan_present, indicating whether a trojan attack was present (1) or not (0).

  The dataset was split into training and testing subsets using an 80:20 ratio**:**

  - **80% Training Data**: Used to fit the machine learning models and enable them to learn patterns in the data.
  - **20% Testing Data**: Held out for evaluating model performance on unseen data, ensuring unbiased assessment of their generalization ability.

  The splitting process ensured a balanced distribution of attack and normal traffic instances in both subsets by employing stratified sampling. This approach preserved the original class proportions, critical for models to learn and predict both classes effectively.

- **Training and Evaluation of the XGBClassifier Model**
  The XGBClassifier, a gradient-boosting algorithm, was selected for its exceptional performance and ability to handle complex datasets effectively. The model was trained using 5-fold cross-validation, ensuring robust evaluation and generalizability. The key hyperparameters employed during training were:

  - **N_estimators**: 1000
  - **eval_metric**: "logloss"
  - **Learning Rate**: 0.01
  - **Max Depth**: 5
  - **Number of Estimators**: 150
  - **Subsample**: 0.8
  - **Colsample_bytree**: 0.8

  These parameters optimized the learning process, balancing accuracy and computational efficiency. Below is the snippet showing the params of XGB Classifier:

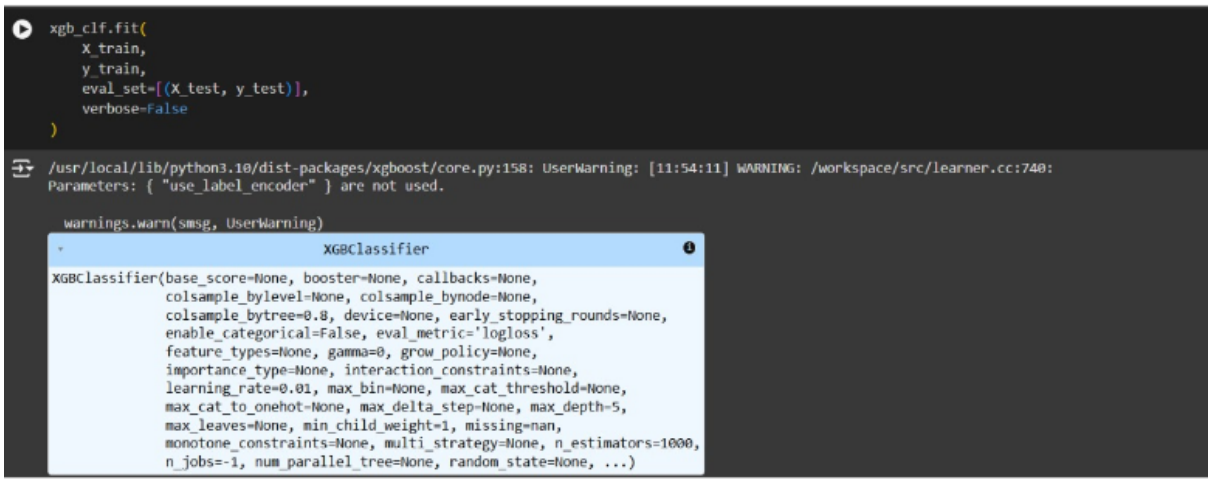

Fig 16. XGB Model Training

Figure 16 shows XGB Model Training and the training process was conducted on the X_train, X_test, Y_train and Y_test subsets for each fold. The individual fold accuracies were **98.05%**, **98.05%**, **98.05%**, **99.51%**, and **99.02%**, resulting in an average accuracy of **98.54%**.

These results underscore the XGBClassifier's ability to predict the presence of trojans with remarkable reliability. The high precision indicated minimal false positives, while the high recall demonstrated its capability to detect nearly all trojan attacks. The balanced F1-score further validated the overall effectiveness of the model for this task.

- **Training and Evaluation of the Random Forest Classifier**
  The Random Forest Classifier (RFC), an ensemble-based algorithm, was trained on the dataset using 5-fold cross-validation to ensure robust evaluation. This model combines predictions from multiple decision trees, making it resilient to noise and overfitting. The key hyperparameters used during training were:
  - **N_estimators =** 1000
  - **Max_depth =** None
  - **Min_samples_split =** 2
  - **Min_samples_leaf =** 1
  - **Max_features =** "sqrt"

  Below is the snippet showing the params of RFC Classifier:

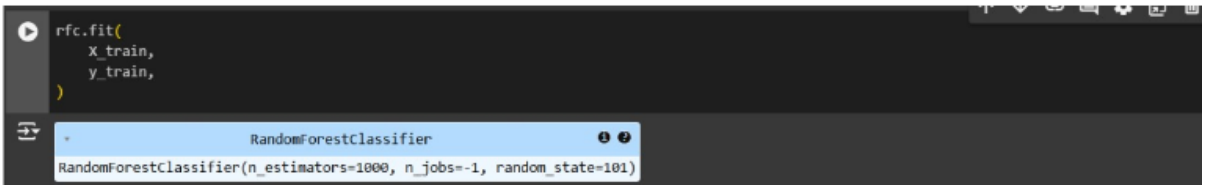

Fig 17. RFC Model Training

Figure 17 shows the RFC parameters and the model was trained using X_train and y_train subsets for each fold, and its performance was evaluated across all folds. The individual accuracies for the 5 folds were **97.07%**, **98.05%**, **94.63%**, **94.63%**, and **97.55%**, resulting in an overall average accuracy of **96.38%**. This consistent performance across folds highlights the reliability of the RFC classifier for this task.

5-Fold Cross-Validation Accuracy for RFC and XGBClassifier

Fold	**RFC Accuracy (%)**	**XGB Accuracy (%)**
Fold 1	97.07	98.05
Fold 2	98.05	98.05
Fold 3	94.63	98.05
Fold 4	94.63	99.51
Fold 5	97.55	99.02
Average	**96.38**	**98.54**

4. **Multilabel Classification for Affected Routers:**
Now we delve into the multilabel classification task aimed at identifying the specific routers affected by trojan activity. Unlike binary classification, where the goal was to detect the presence or absence of a trojan, this task required identifying the exact locations of trojans across multiple routers. This process involved careful data preparation, transformation into a multilabel format, and training of a multi-output classifier.

- **Data Preparation and Feature Selection**
For this classification task:

- **X vector (Features)**: Included only the crossbar activity data for all 64 routers. The crossbar activity data was selected as it directly reflects the operational behavior of the routers, which is crucial for identifying anomalies caused by trojans.
- **Y vector (Targets)**: Consisted of the columns trojan_location_1 and trojan_location_2. These columns represent the routers potentially affected by trojan activity.

Before proceeding with training, we retained only the data points where a trojan was confirmed to be present. This ensured that the models were trained exclusively on relevant instances, eliminating noise from normal router traffic that could otherwise hinder model performance.

- **Transformatoin into Multilabel Format**
  Since the target involved multiple router locations, the Y vector required transformation into a format suitable for multilabel classification. To achieve this:
  - **MultiClassClassifier**:
    * The MultilabelBinarizer from the sklearn library was applied to transform the target columns (trojan_location_1 and trojan_location_2) into a multilabel binary format.
    * Each router ID was represented as a separate binary column, with a value of 1 indicating an affected router and 0 for unaffected routers.

  The above transformation enabled the model to predict multiple outputs simultaneously, each corresponding to a specific router.

- **Model Training: Multi-Output Classifier**
  The multilabel problem was approached using a MultiOutputClassifier:
  - The MultiOutput Classifier is a wrapper that trains separate binary classifiers for each target label, enabling the simultaneous prediction of multiple outputs.
  - The base estimator for this task was the XGBClassifier, chosen for its robust performance and adaptability to complex datasets.

  Below is the snippet showing the params of Multi-Output Classifier:

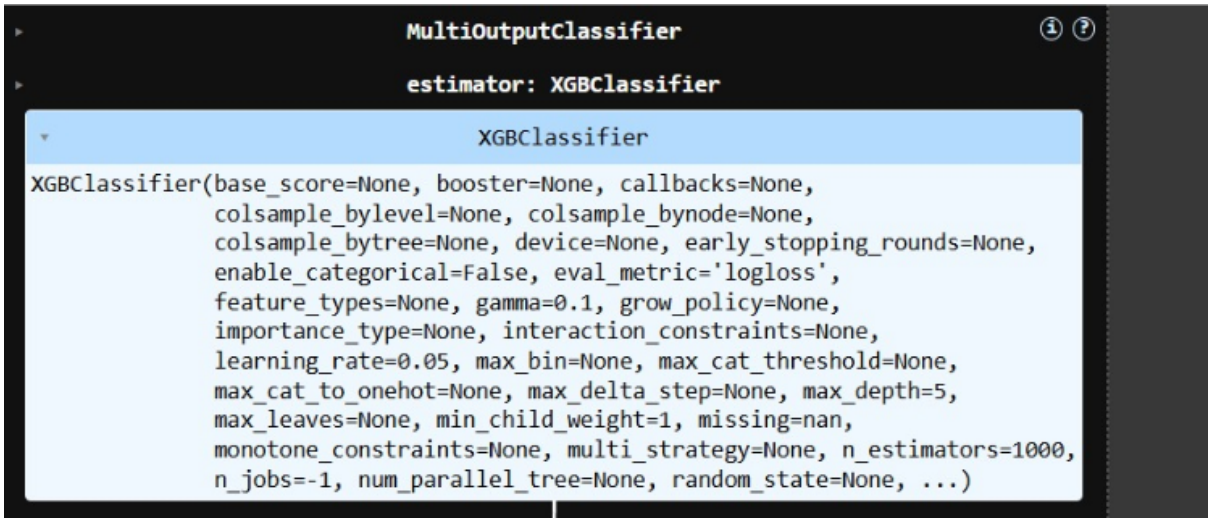

Fig 18. MultiOuput Classifier Training and Params

The Figure 18 shows the MultiOuput Classifier Training and its parameters. The training process involved:

(a) Feeding the transformed X vector (crossbar activity) and multilabel Y vector (binary-encoded trojan locations) into the multi-output classifier.

(b) Utilizing the same hyperparameter tuning strategies as in previous tasks, ensuring the model was well-optimized for this multilabel classification problem.

- **Model Evaluation and Results**

  After training, the model was tested on the reserved testing data to assess its performance. The evaluation revealed:

  Accuracy: The model achieved an accuracy of **75.55%**, indicating its ability to correctly predict the affected routers in most cases.

  The moderate accuracy reflects the complexity of the task, as accurately pinpointing specific affected routers.

  To interpret the model's predictions:

  (a) The binary outputs generated by the MultilabelBinarizer were converted back to readable router IDs.

  (b) This step ensured that the results were human-readable, with affected routers identified by their unique IDs.

5. **Conversion of trained ML model to C code:**

   As a final step, we translated the Python codebase of the model into C using m2cgen library to optimize performance and enable seamless integration into network hardware simulations. This transition significantly improved execution speed and memory efficiency, making the solution more practical for real-time

applications. The C implementation retains all the functionality of the original Python model while catering to the stringent performance requirements of network-on-chip environments.

5 Conclusion and Future Work

The objectives outlined have been systematically addressed, contributing to the advancement of hardware Trojan detection and mitigation in Networks-on-Chip (NoC) environments. This comprehensive approach combines analysis, simulation, dataset generation, and machine learning to enhance NoC security. A thorough investigation of current methodologies for hardware Trojan detection has uncovered key strengths and limitations, providing the foundation for innovative solutions that address gaps in scalability, detection accuracy, and adaptability specific to NoC architectures. By simulating Distributed Denial of Service (DDoS) attacks in a 2D mesh NoC employing XY routing, we have demonstrated how such attacks can exploit NoC vulnerabilities, serving as both a validation of the threat and a benchmark for developing robust detection mechanisms.

The generation of a high-quality, diverse dataset tailored to NoC environments is a cornerstone of this research, providing a critical resource for training, validating, and benchmarking machine learning models, enabling reproducible and scalable research. Additionally, a machine learning-based detection model has been developed, demonstrating exceptional performance in identifying hardware Trojans. This model leverages cutting-edge algorithms to ensure adaptability to diverse NoC configurations, addressing the dynamic nature of hardware threats. Beyond detection, the ability to localize compromised routers enhances the precision of response strategies. The localization model not only identifies affected routers but also provides actionable insights for mitigating the impact of Trojans in real time.

5.1 Future Work

- Enhance the accuracy of attack localization, which currently stands at 75%, by:
  - Refining the feature selection process.
  - Optimizing the prediction model.
- Expand the scope of research to include:
  - Modeling and analysis of another type of delay attack.
  - Studying the impact of the delay attack on crossbar activity.
  - Incorporating this analysis into the predictive framework to improve robustness in detecting and localizing diverse attack scenarios.

- Convert the optimized C code into Verilog for deployment on a security engine within a hardware environment:
  - Enable real-time attack detection and localization directly on the chip.
  - Leverage high-speed and parallel processing capabilities of hardware implementations.
  - Integrate the predictive framework into hardware for enhanced efficiency and practical applicability in securing network-on-chip architectures.

References

[1] Niket Agarwal et al. "GARNET: A detailed on-chip network model inside a full-system simulator". In: *2009 IEEE International Symposium on Performance Analysis of Systems and Software*. 2009, pp. 33–42. DOI: 10.1109/ISPASS.2009.4919636.

[2] Luca Benini and Giovanni Micheli. "Networks on Chips". In: Dec. 2005, pp. 49–80. ISBN: 9780123852519. DOI: 10.1016/B978-012385251-9/50017-7.

[3] Humboldt University of Berlin. *Image Classification (Part 2)*. 2023. URL: https://pages.cms.hu-berlin.de/EOL/geo_rs/S08_Image_classification2.html.

[4] Antony Christopher. *Random Forest*. 2021. URL: https://medium.com/analytics-vidhya/random-forest-4a2981aab4f7.

[5] SFU CSPMP. *XGBoost: A Deep Dive into Boosting*. 2023. URL: https://medium.com/sfu-cspmp/xgboost-a-deep-dive-into-boosting-f06c9c41349.

[6] W.J. Dally and B. Towles. "Route packets, not wires: on-chip interconnection networks". In: *Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232)*. 2001, pp. 684–689.

[7] GeeksforGeeks. *Random Forest Algorithm in Machine Learning*. 2023. URL: https://www.geeksforgeeks.org/random-forest-algorithm-in-machine-learning/.

[8] gem5. *gem5 Documentation*. 2023. URL: https://www.gem5.org/documentation/.

[9] Ruchika Gupta et al. "Securing On-chip Interconnect against Delay Trojan using Dynamic Adaptive Caging". In: *Proceedings of the Great Lakes Symposium on VLSI 2022*. GLSVLSI '22. Irvine, CA, USA: Association for Computing Machinery, 2022, pp. 411–416. ISBN: 9781450393225. DOI: 10.1145/3526241.3530333. URL: https://doi.org/10.1145/3526241.3530333.

[10] Spot Intelligence. *Random Forest Classifier*. 2023. URL: https://spotintelligence.com/2023/08/31/random-forest-classifier/.

[11] Chawade Mahendra, Mahendra Gaikwad, and Rajendra Patrikar. "Review of XY Routing Algorithm for Network-on-Chip Architecture". In: *International Journal of Computer and Communication Technology* (Oct. 2016). DOI: 10.47893/IJCCT.2016.1384.

[12] Aakash Parmar, Rakesh Katariya, and Vatsal Patel. "A Review on Random Forest: An Ensemble Classifier". In: *International Conference on Intelligent Data Communication Technologies and Internet of Things (ICICI) 2018*. Ed. by Jude Hemanth et al. Cham: Springer International Publishing, 2019, pp. 758–763. ISBN: 978-3-030-03146-6.

[13] Vishal Singh. *Brief Notes on SoC and NoC*. 2023. URL: https://www.linkedin.com/pulse/brief-notes-soc-noc-vishal-%D1%95ingh/.

[14] Chamika Sudusinghe, Subodha Charles, and Prabhat Mishra. "Denial-of-Service Attack Detection using Machine Learning in Network-on-Chip Architectures". In: *2021 15th IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*. 2021, pp. 35–40.

[15] Wen-Chung Tsai et al. "Networks on Chips: Structure and Design Methodologies". In: *Journal of Electrical and Computer Engineering* 2012 (2012), pp. 1–15. DOI: 10.1155/2012/509465. URL: https://onlinelibrary.wiley.com/doi/10.1155/2012/509465.