Denial-of-Service Attack Detection using Machine Learning in Network-on-Chip Architectures

Chamika Sudusinghe[‡], Subodha Charles[‡] and Prabhat Mishra[*]
[‡]University of Moratuwa, Colombo, Sri Lanka
[*]University of Florida, Gainesville, Florida, USA

ABSTRACT

State-of-the-art System-on-Chip (SoC) designs consist of many Intellectual Property (IP) cores that interact using a Network-on-Chip (NoC) architecture. SoC designers increasingly rely on global supply chains for obtaining third-party IPs. In addition to inherent vulnerabilities associated with utilizing third-party IPs, NoC based SoCs enable attackers to exploit the distributed nature of NoC and its connectivity with various IPs to launch a plethora of attacks. Specifically, Denial-of-Service (DoS) attacks pose a serious threat in degrading the SoC performance by flooding the NoC with unnecessary packets. In this paper, we present a machine learning-based runtime monitoring mechanism to detect DoS attacks. The models are statically trained and used for runtime attack detection leading to minimum runtime performance overhead. Our approach is capable of detecting DoS attacks with high accuracy, even in the presence of unpredictable NoC traffic patterns caused by various application mappings. We extensively explore machine learning models and features to provide a comprehensive study on how to use machine learning for DoS attack detection in NoC-based SoCs.

1 INTRODUCTION

Network-on-chip (NoC) is widely used for on-chip communication in modern system-on-chips (SoC). NoC has allowed computer architects to fully utilize the computational power in an SoC by facilitating low-latency and high-throughput communication between intellectual property (IP) cores in a many-core SoC. As a result, NoC has become a critical component in state-of-the-art SoC designs [16, 17, 19]. With the increased complexity of SoCs, manufacturers have favored IP licensing and outsourcing where only a subset of IPs are manufactured in-house and the rest is sourced from third-party vendors. There are multiple avenues to introduce malicious implants (e.g., hardware Trojans) in designs during the long supply chain, such as by an untrusted CAD tool, a rouge designer or at the foundry via reverse engineering [10, 17].

The NoC is at an elevated risk of being vulnerable to hardware attacks due to several reasons: i) NoC interconnects IPs manufactured in house and/or sourced from trusted vendors (secure IPs) together with IPs from potentially untrusted vendors (non-secure IPs) allowing Trojan-infected malicious IPs (MIPs) to utilize NoC to launch attacks, ii) the distributed nature of NoC makes it easier to replicate an attack, and iii) the complexity of NoC design allows Trojans to hide without being detected. These vulnerabilities have motivated both industry and academic researchers to develop countermeasures to secure NoC-based SoCs. There are a wide variety of threats from MIPs such as eavesdropping attacks [6, 9], data integrity attacks [8], denial-of-service (DoS) attacks [5], etc. In this paper, we focus on securing the SoC from DoS attacks. The primary objective of a DoS attack is to prevent legitimate users from accessing services and information. In the context of NoC, MIPs sending unnecessary requests to IPs can delay legitimate requests leading to delay of service (e.g., deadline violations in real-time systems) or denial of service (e.g., temporary or permanent service failure). Such "flooding" type of DoS attacks can also cause congestion in the network, further degrading performance and energy efficiency [5, 7].

Previous work on mitigating flooding type of DoS attacks explored traffic latency comparison [15] and packet arrival monitoring [5, 7]. These approaches made an unrealistic assumption, highly predictable NoC traffic patterns, which allowed the construction of linear statistical bounds to detect DoS attacks. Unfortunately, this assumption does not hold during many realistic scenarios that include task migration, task preemption, changing application characteristics due to major input variations, etc. As a potential solution to address such runtime variations, in this paper, we explore the feasibility of using machine learning (ML) for DoS attack detection. While ML has shown promising results for optimizing NoC power consumption [20], to the best of our knowledge, our approach is the first attempt at securing NoC-based SoCs from DoS attacks using machine learning. Major contributions of this paper are as follows:

- We propose an ML-based DoS attack detection method that trains ML models during design time and uses the trained models to classify network traffic behavior as normal or attack during runtime, to detect flooding type of DoS attacks.
- We outline features that can be extracted from NoC traffic as well as engineered features, and experimentally evaluate the most suitable features.
- We perform a comprehensive exploration of 12 different ML models to select the best fit for the given architecture and threat models.
- Our approach achieves high accuracy in DoS attack detection across different NoC traffic patterns caused by various application mappings.
- Our approach can detect DoS attacks in real-time with detection times comparable to previous work [5, 7] without requiring highly predictable traffic patterns.

2 THREAT MODEL AND RELATED WORK

2.1 Threat Model

Figure 1 shows the architecture model considered in this paper that includes a 4×4 mesh NoC connecting 16 IP cores. When a memory request (e.g., memory LOAD or STORE instruction) is initiated

This work was partially supported by National Science Foundation grant SaTC-1936040.

NOCS '21, October 14–15, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-9083-5/21/10…$15.00
https://doi.org/10.1145/3479876.3481589

by a core during application execution, in case of a cache miss, a memory request is injected into the NoC in the form of NoC packets. Typically, the packets are further broken down into smaller units called *flits* to facilitate flow control mechanisms. The flits are routed in the appropriate virtual network (vnet) that matches the cache coherence request type via routers and links. When the flits arrive at the memory controller, the memory fetch is initiated. Once the operation is completed, the response is routed back to the requestor.

DoS attacks can happen from MIPs intentionally degrading SoC performance by flooding the NoC with packets. MIPs can target a component that is critical to SoC performance, such as a memory controller that provides the interface to off-chip memory, and inject unnecessary requests [7]. As a result, the legitimate requests can experience severe delays. Figure 1 shows a MIP at node 1 that targets its victim at node 7 and injects additional packets. The traffic rate in routers along the routing path is increased causing NoC congestion, which leads to performance degradation and reduced energy efficiency. Since the victim receives a lot more requests than it is designed to handle, responses are delayed and that can lead to violation of task deadlines. Violation of real-time requirements can be catastrophic for safety-critical applications. A similar threat model was also used by previous work that explored DoS attacks in NoC-based SoCs [5, 7, 15].

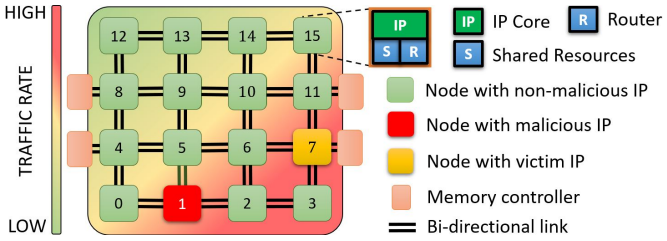

Figure 1: Example DoS attack from a malicious IP to a victim IP. The thermal map shows high traffic near the victim IP.

2.2 Related Work

Previous work that explored defenses against DoS attacks proposed traffic latency comparison [15] and security verification techniques [3]. However, these approaches give suboptimal results due to inherent drawbacks in their methodologies such as injection of additional packets that can further congest the network [15] and being able to reduce the risk of attacks, but unable to detect if an attack happens [3]. Fiorin et al. introduced a countermeasure against DoS attacks that has an architecture similar to our work [12]. However, their method is fundamentally different from ours since they monitor the bandwidth considering the data loaded/stored by an initiator from/to a specific memory block or range of addresses. Charles et al. proposed to statically profile the normal behavior of the SoC and detect DoS attacks during runtime [5, 7]. In their work, each router statically profiled NoC traffic behavior based on packet arrivals at routers and used that as an upper bound to detect attacks. While such methods are efficient when the applications are fixed, they are not suitable when variations can alter the NoC traffic behavior.

ML has been widely adopted in various domains for efficient data processing and fast decision making. For example, ML can analyze encrypted HTTP traffic to differentiate between malicious and benign execution [13, 18]. Cisco encrypted traffic analytics [13] and IBM QRadar security intelligence [14] are two state-of-the-art network security countermeasures developed by the industry. Shekhawat et al. [18] showed that XGBoost, an algorithm based on gradient boosting, can classify encrypted traffic as malicious or benign with an accuracy of 99.15%. To the best of our knowledge, there are no prior efforts that use ML to secure NoC-based SoCs from DoS attacks.

3 ML-BASED DOS ATTACK DETECTION

As a means of achieving high accuracy in detecting DoS attacks in the presence of runtime variations of NoC traffic, we explore the feasibility of using ML for DoS attack detection. An overview of our approach is shown in Figure 2. During design time, NoC traffic is statically analyzed to gather the dataset that is used to train the ML models. Both normal and attack scenarios are emulated during this phase using a few known application mappings. The trained models are stored in a dedicated IP denoted as the *Security Engine* (SE). During runtime, NoC traffic data is gathered at each router using probes attached to routers and the collected data is sent to the SE using a separate physical *Service NoC*. The models at the SE use data collected within a predefined time window to make inferences about the condition of the NoC. In Section 4, we show that our method is capable of classifying data as normal or attack, irrespective of the locations of cores running the applications (active cores) and the locations of MIP(s).

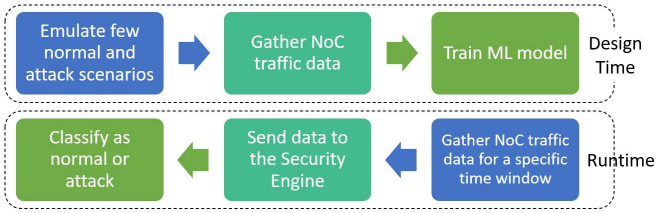

Figure 2: Major steps of the ML-based DoS attack detection.

Our ML-based DoS attack detection mechanism relies on the following features of the architecture model.

- Probes attached to routers can gather data from NoC packets with minor performance and power overhead.
- The SoC architecture comprises two physical NoCs: (i) a *Data NoC* that is used to communicate between IPs for application execution, and (ii) a *Service NoC* which transfers data collected from probes to the SE.

The remainder of this section is organized as follows. Section 3.1 presents the ML models used to make inferences. Section 3.2 discusses the hardware implementation to have probes connected to routers that gather data and send to the SE via the Service NoC.

3.1 Machine Learning Model

As outlined in Section 2.1, NoC packets/flits in our architecture model correspond to memory requests/responses between IPs running the applications and the memory controllers. We extract information when flits are transferred through routers. The features consist of data extracted from NoC packets as well as engineered features (marked with the symbol † in Table I) using the extracted data. A complete list of NoC traffic features used in our exploration is shown in Table 1. However, as elaborated in Section 4.3, we experimentally eliminated some features based on feature importance[1] in an attempt to find

[1]Feature importance gives a score to indicate how important a feature is in the decision making process of an ML model. In a trained model, the more a feature contributes to key decisions, the higher its relative importance.

the optimum trade-off between the least number of features and the highest model accuracy. Feature IDs of the selected features, when running the final model, are marked with a star (*) in Table 1.

We use *Gradient Boosting*, a powerful technique to perform supervised ML classification, as our classifier. It is an ensemble learner that creates the final model based on a collection of weak predictive models, decision trees in most instances, and that results in better overall prediction capabilities due to iterative learning from each model. The key concept of the algorithm is to create new base-learners having a maximum correlation with the negative gradient of the loss function of the entire ensemble. Weaker predictive models in the ensemble are trained gradually, additively, and sequentially, and their shortcomings are identified by the use of gradients in the loss function which indicates the acceptability of the model's coefficients at fitting the underlying data. The decision to use gradient boosting for our classification was made experimentally as outlined in Section 4.2.

3.1.1 Training the ML model. The ML model is trained statically, during design time. We choose a few application mapping scenarios to train the model that includes both normal execution and attack scenarios. A list of all training and testing configurations is outlined in Section 4.1. NoC packet traces are collected during application execution at each router. When flits pass through the routers, a feature vector is constructed including the selected features for each flit. Selected features are transformed using *MinMaxScaler* to fit into the range of 0 to 1, without distorting the shape of the original features. Transformed features are then used to tune the hyperparameters of the model using *Bayesian Optimization*, which outputs the best-optimized list of parameters while learning from previous iterations in each iteration. This process is repeated for all 16 routers separately to train 16 models, one per router.

3.1.2 Attack detection. During runtime, probes attached to the routers gather data and send to the SE for evaluation. The SE aggregates data and constructs feature vectors corresponding to each router, following a process similar to that of during model training. Let $\mathcal{M}_i$ correspond to the model trained for router r_i using gradient boosting. Feature vectors that fall within a predefined time window τ_j is then used as input to each trained model, which gives a probability of an attack as the output. If $\mathcal{V}_{i,j}$ denotes the set of feature vectors constructed at r_i for τ_j, the probability of an attack is denoted by $p_{i,j}$, where $p_{i,j} \leftarrow \mathcal{M}_i(\mathcal{V}_{i,j})$. The probability is calculated as the portion of feature vectors labeled as "attack" during τ_j. If all feature vectors are classified as "attack" by the model, the probability is 1. If all feature vectors are classified as "normal", the probability is 0. The overall attack probability for the time window τ_j is calculated after *pooling* all probabilities as:

$$\mathcal{P}_j = \frac{\sum_{\forall i}(p_{i,j} \cdot |\mathcal{V}_{i,j}|)}{\sum_{\forall i} |\mathcal{V}_{i,j}|} \tag{1}$$

The overall probability for the time window τ_j ($\mathcal{P}_j$) is a weighted average of probabilities from each model where the weights correspond to the number of flits transferred through each router within the given time window. If $\mathcal{P}_j$ is greater than a predefined threshold λ, an attack is flagged. This process is repeated for every τ_j during SoC operation to detect attacks that can be potentially initiated at any point in time.

Weights based on the number of flits indicate that when a model makes a decision based on a lot of data points, it can be trusted to give a more accurate result. The choice was motivated by the fact that we make no assumptions about the placement of the secure and non-secure IPs. However, if more information is available, the weighted average can be adjusted so that some models contribute more to the final decision. For example, if the locations of the non-secure (potentially malicious) IPs are known, the probabilities of models corresponding to those routers can be given more weight and it would result in a better overall performance in distinguishing normal traffic from an attack scenario. How to combine different probabilities to arrive at a single conclusion under various assumptions is well studied in the area of *Opinion Pooling*, which is a part of probability theory, and can be used in our approach based on the assumptions.

It is important to note that all the features we have used in our method can be extracted from the packet header or by counting flits or as a combination of header and count information. Observing the packet payload (e.g., memory data block in case of a memory data fetch packet) is not required. Therefore, our approach can be used together with other NoC security mechanisms such as encryption and authentication.

Table 1: NoC traffic features used in our ML model

Feature ID	Feature Name	Feature Description
A	outport*	port used by the flit to exit the router (0-local,1-north,2-east,3-south,4-west)
B	inport	port used by the flit to enter the router (0-local,1-north,2-east,3-south,4-west)
C	cc type†	cache coherence type of the packet corresponding to the flit
D	flit id	identifier used to denote each flit
E	flit type	type of the flit (head, tail, body)
F	vnet	virtual network used by the flit
G	vc*	virtual channel used by the flit
H	traversal id*†	identifier used to group all packet transfers related to one NoC traversal
J	hop count*†	number of hops from the source to the destination
K	current hop†	number of hops from the source to the current router
L	hop percentage†	ratio between the current hop and the hop count
M	enqueue time*	time spent inside the router by the flit
N	packet count decr.*†	cumulative no. of flit arrivals within time window τ (decremented as packets arrive)
O	packet count incr.*†	cumulative no. of flit arrivals within time window τ (incremented as packets arrive)
P	max packet count*†	maximum no. of flits transferred through the router within a given time window τ
Q	packet count index*†	packet count incr × packet count decr
R	port index†	outport × inport
S	traversal index*†	cache coherence type × flit id × flit type × traversal id
T	cc vnet index†	cache coherence type × vnet
U	vnet vc cc index†	cache coherence vnet index × vc

3.2 Implementation of Hardware Components

Our approach relies on collecting features at routers using probes and sending the data via a separate physical NoC (Service NoC) to the SE

to make inferences. In this section, we discuss the implementation of these hardware components.

3.2.1 Multiple physical NoCs. We identify two main types of packets to be transferred through the NoC to facilitate our ML-based DoS attack detection method: i) packets related to application execution as introduced in Section 2.1, and ii) packets related to extracted NoC features transferred from probes at routers to the SE. Instead of using different virtual networks to carry the different packet types, we propose to use two separate physical NoCs (Data NoC and Service NoC) to carry the two main types of packets. The choice is motivated by state-of-the-art commercial NoC-based SoC architectures that follow the same practice of carrying different types of packets over multiple physical NoCs [19, 21]. There is a trade-off between area and performance when considering one versus multiple NoCs. When different packet types are facilitated through the same NoC, header fields must be added to distinguish between the packet types. Furthermore, the buffer space must be shared between virtual networks. This can lead to performance degradation, specially when scaling to many-core processors. On the other hand, separate physical NoCs contribute to the area overhead. However, due to advances in manufacturing technologies, additional wiring to facilitate the NoCs incurs minimal overhead as long as the wires stay on-chip. On-chip buffer area has become the more scarce resource. If virtual networks are used, the increased buffer space due to sharing and the logic complexity to handle virtual networks can closely resemble to having a separate physical NoC. Intel and Tilera opted for separate physical NoCs for the same reasons. Yoon et al's work provides a comprehensive trade-off analysis [23]. When we apply the analysis from [23] to fit the parameters in our work, the power and area overhead of having two physical NoCs versus one NoC are 7% and 6%, respectively.

3.2.2 Probes at routers and security engine. Hardware implementations for probes collecting data at routers and the SE have been explored in several prior work [11, 12]. Fiorin et al. [12] utilized probes attached to the network interfaces to collect data and send to a central processor to detect DoS attacks. The runtime NoC monitoring and debugging framework proposed in [11] also used a similar setup where event related information is gathered at NoC routers and sent to a central unit for processing. Our security mechanism is built using a similar architecture. In our framework, the probes are event triggered on flit arrival. The probes consist of a sniffer, an event generator and an interface to the Service NoC. The sniffer extracts the features from flits and sends to the event generator to create the timestamped messages. The network interface then packetizes the messages and sends to the SE via the Service NoC. The SE completes feature engineering and combines the engineered and extracted features to construct the final feature vectors. Previous work performed detailed overhead analysis and reported minimal area overhead, for example, the probes consumed $0.05mm^2$ compared to a $0.26mm^2$ router area when synthesized with 0.13 micron technology [11]. Our overhead analysis is consistent with the analysis done in [11].

4 EXPERIMENTS

In this section, we experimentally evaluate our approach. First, we describe our experimental setup (Section 4.1). Next, we explore several machine learning models to identify the best performing one and justify the choice of gradient boosting (Section 4.2). Then, we rank feature importance according to the selected model and eliminate low priority features in an attempt to find the optimum trade-off between the number of features and model accuracy (Section 4.3). Finally, we show how our ML-based DoS attack detection mechanism performs across several training and testing configurations by exploring model accuracy for all the test cases in Table 2 (Section 4.4).

Table 2: Train and test configurations

Iteration ID (IID)	Train		Test
	Normal	Attack	Attack
1	N-0-15	N-0-15-A-1	N-0-15-A-7
			N-0-15-A-11
			N-0-15-A-12
2	N-0-15	N-0-15-A-1	N-0-15-A-7
	N-0-15	N-0-15-A-11	N-0-15-A-12
	N-0-9	N-0-9-A-1	N-0-9-A-7
	N-0-9	N-0-9-A-11	N-0-9-A-12
	N-0-6	N-0-6-A-1	N-0-6-A-7
	N-0-6	N-0-6-A-11	N-0-6-A-12
	N-0-4	N-0-4-A-1	N-0-4-A-7
	N-0-4	N-0-4-A-11	N-0-4-A-12
3	N-0-6-9-15	N-0-6-9-15-A-1-11	N-0-6-9-15-A-1-7
			N-0-6-9-15-A-7-11
			N-0-6-9-15-A-11-12
			N-0-6-9-15-A-7-12

4.1 Experimental Setup

Following the realistic architecture model proposed in [4], the 4×4 mesh NoC was modeled using the "GARNET2.0" framework [1] that is integrated with the gem5 [2] cycle-accurate full-system simulator. The NoC model was implemented using X-Y routing with wormhole switching, 3-stage router pipeline (buffer write, route compute + virtual channel allocation + switch allocation, and link traversal) and 4 virtual channel buffers per input port. Each IP was modeled as a processor core executing a given task at 1 GHz with a private L1 cache. Processor cores used the NoC for memory operations as outlined in Section 2.1. The four memory controllers attached to four boundary nodes of the NoC provided the interface to off-chip memory. The address space was shared equally between the memory controllers. FFT benchmark from the SPLASH-2 benchmark suite [22] was used for application instances. The same benchmark has been used in [5, 7] that explored DoS attacks in NoC-based SoCs. During normal operation, n IPs out of the 16 IPs in the 4×4 mesh, were chosen at random to run an instance of the benchmark (active IPs). To model the DoS attack scenario, an IP that did not run an instance of the benchmark injects memory request packets to the four memory controllers increasing the overall network traffic by 50%. A complete set of training and testing configurations are listed in Table 2. Iteration ID (IID) 1 indicates that the model has been trained with two datasets: i) normal execution scenario with applications running on IPs 0 and 15 (N-0-15), and ii) attack scenario with an attacker at IP 1 launching a DoS attack while applications are running on IPs 0 and 15 (N-0-15-A-1). The trained model has been tested with three attack scenarios: i) N-0-15-A-7, ii) N-0-15-A-11, and iii) N-0-15-A-12. The IP numbers correspond to the node numbers given in Figure 1.

4.2 Machine Learning Model Comparison

To identify which ML model performs the best for our given architecture and threat models, we compared the performance of 12 ML models - Naive Bayes Classifer (NBC), Logistic Regression (LRN),

2-Layer Neural Network (2NN), 3-Layer Neural Network (3NN), 4-Layer Neural Network (4NN), 5-Layer Neural Network (5NN), 6-Layer Neural Network (6NN), K-Neighbors Classifier (KNN), LightGBM Classifier (LGB), Decision Tree Classifier (DCT), Random Forest Classifier (RFC), and XGBoost Classifier (XGB). Each model was trained using the training dataset of IID 2. Figure 3 shows training accuracy and validation accuracy measured using an 80:20 training:validation split from the dataset at router 0 (r_0). The model comparison results at other routers manifested a similar trend (omitted from Figure 3 for clarity). We can observe that non-linear ML models perform better than linear models with XGB showing the best results. XGBoost is an algorithm based on gradient boosting machines.

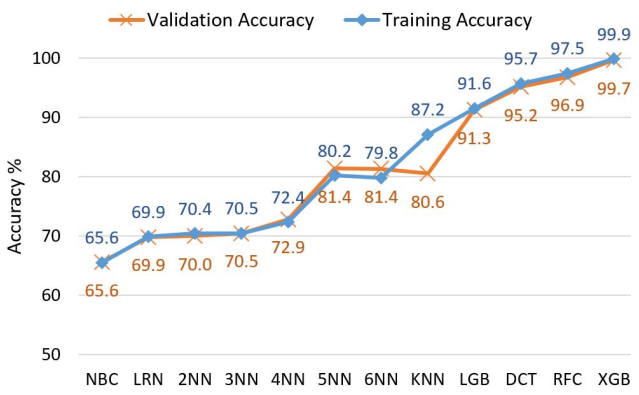

Figure 3: ML model performance comparison using IID 2 training dataset.

To evaluate the selected XGB model further, we use cross validation, which is a resampling process used to evaluate the performance of a trained ML model. We use StratifiedKFold[2] cross validation since it gives a better representation over the entire dataset. Results for 10 folds of StratifiedKFold cross validation are shown in Figure 4. *The results generated by cross validation confirm that the model is less biased, performing well in unseen data and not overfitting. Since our exploration indicated that XGB performs best in the given scenario, we use XGB as the ML model for our DoS attack detection method.*

fold	1	2	3	4	5	6	7	8	9	10
accuracy %	96.84	96.86	96.84	96.88	96.80	96.92	96.84	96.54	96.71	96.90

Figure 4: Validation results of the model using StratifiedKFold cross validation.

4.3 Feature Importance

While using more features can certainly increase model accuracy, extracting redundant features from NoC traffic can lead to unnecessary performance and power overhead. Therefore, we eliminate features that show the least importance for the decision making process of the ML model-XGB and experimentally evaluate the optimum number of features to have the best trade-off with the accuracy of the model. Figure 5 shows the feature importance rank of each feature. Since each router runs a model trained from the data extracted at that particular router, the feature importance rank slightly changes from router to router. However, the overall trend remains consistent where the highlighted features are the least used. Elimination of features *flit id, flit type and cc type* from the decision making process can be understood because during the modeled attack, no header information is changed. Since flits are routed via *vnets* based on their *cc type*, the rank of the feature *vnet* can be perceived. It can be said that *outport* is outperforming *inport* because of the influence of the attacker that injects memory request packets, and *current hop and hop percentage* are relative features when compared to *hop count*. Therefore, for the rest of the exploration, we eliminate the highlighted features when training and testing the accuracy of our approach.

[2] *StratifiedKFold* cross validation uses a subset from each class in the test set emulating a representation of the entire dataset in each fold.

	Router															
Feature	r_0	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}	r_{11}	r_{12}	r_{13}	r_{14}	r_{15}
outport	12	8	8	13	11	9	8	10	11	9	9	10	14	9	9	14
inport	14	12	11	12	10	11	10	14	13	12	11	12	15	11	10	11
cc type	13	15	15	14	15	13	12	16	15	13	13	15	13	14	13	13
flit id	18	18	18	18	20	20	20	19	19	20	20	20	18	18	18	19
flit type	17	17	16	16	19	18	18	18	18	19	18	19	17	17	17	16
vnet	16	19	20	20	18	19	19	20	20	18	19	18	20	20	20	20
vc	10	13	14	10	9	17	15	8	8	16	15	8	10	15	15	10
traversal id	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
hop count	6	7	7	7	7	7	7	7	7	7	7	7	7	7	7	6
current hop	9	16	17	17	14	16	16	12	10	14	16	11	12	16	16	12
hop percentage	15	9	10	11	16	10	11	13	14	10	12	17	11	10	12	15
enqueue time	8	10	9	9	8	8	9	9	9	8	8	9	8	8	8	8
packet count decr	3	4	4	5	4	5	4	4	4	4	4	4	5	5	5	5
packet count incr	5	5	6	6	5	6	6	6	5	6	6	5	6	6	6	7
max packet count	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
packet count index	4	3	3	4	3	3	3	3	3	3	3	3	4	4	4	4
port index	20	14	13	15	13	12	14	11	12	11	10	14	16	12	11	18
traversal index	7	6	5	3	6	4	5	5	6	5	5	6	3	3	3	3
cc vnet index	19	20	19	19	17	15	17	17	17	17	17	16	19	19	19	17
vnet vc cc index	11	11	12	8	12	14	13	15	16	15	14	13	9	13	14	9

Figure 5: Feature importance rank for features at each router for IID 2 dataset with least important features highlighted.

4.4 DoS Attack Detection Accuracy

With the selected model and features, in this section, we evaluate the accuracy of our DoS attack detection method. As outlined in Section 3, each model outputs the attack probability independently for a given time window τ_j. The overall attack probability during τ_j ($\mathcal{P}_j$) is calculated according to Equation 1. Figure 6 and Figure 7 show excerpts from results generated during an attack (IID 2 and test case N-0-15-A-12) and a normal (IID 2 and test case N-0-15) scenario, respectively. The threshold for inferring attacks from $\mathcal{P}_j$ is set to 0.5 ($\lambda = 0.5$) since an attack scenario should give probabilities close to 1 whereas in a normal scenario, the probabilities should be close to 0. Columns "r_0" through "r_{15}" in Figure 6 and Figure 7 show the probabilities outputted by models corresponding to each router. Column "$\mathcal{P}_j$" shows the overall probability for time window τ_j calculated using Equation 1 and the "Status" column indicates the final decision of the ML model for each τ_j. The two excerpts show 100% accuracy since all the time windows are classified accurately. However, each test case consists of more than 3000 time windows (3280 in the complete table corresponding to Figure 6), which is related to the application execution time. DoS attack detection accuracy is calculated as the portion of accurately classified time windows.

Figure 8 shows DoS attack detection accuracy for all test cases shown in Table 2. In IID 1, the model is trained with two datasets (N-0-15 and N-0-15-A-1) and tested with varying MIP locations (7, 11 and 12). Even though the number of training datasets is low, the ML model still achieves an accuracy of ~90%. As the number of training datasets is increased, the model achieves very high accuracy (~99%),

τ_j	Attack Probability																P_j	Status
	r_0	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}	r_{11}	r_{12}	r_{13}	r_{14}	r_{15}		
τ_1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	ATTACK
τ_2	1	1	1	0.3	1	1	1	1	1	1	1	1	1	1	1	1	1	ATTACK
τ_3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	ATTACK
τ_4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	ATTACK
τ_5	1	1	1	1	1	1	0.7	1	1	1	1	1	1	1	1	1	1	ATTACK
τ_6	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	ATTACK
τ_7	0.2	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0.9	ATTACK
τ_8	1	1	0.8	1	1	1	0.5	0	1	1	1	1	1	1	1	1	1	ATTACK
τ_9	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	ATTACK
τ_{10}	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	ATTACK

Figure 6: Results for attack scenario for IID 2 and test N-0-15-A-12 (τ_j = 1000).

τ_j	Attack Probability																P_j	Status
	r_0	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}	r_{11}	r_{12}	r_{13}	r_{14}	r_{15}		
τ_1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.0	NORMAL
τ_2	0	0	0	0.7	0	0	0	0	0	0	0	0	0	0	0	0	0.0	NORMAL
τ_3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.0	NORMAL
τ_4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.0	NORMAL
τ_5	0	0	0	0	0	0	0	0	0	0	0	0	0.8	0	0	0	0.0	NORMAL
τ_6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.0	NORMAL
τ_7	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0.0	NORMAL
τ_8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.0	NORMAL
τ_9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.0	NORMAL
τ_{10}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.0	NORMAL

Figure 7: Normal scenario IID 2 and test N-0-15 (τ_j = 1000).

even when tested with MIP locations which the model was not trained on. Since a decision is made at the end of each time window, the time taken to detect an attack is τ_j, which is experimentally set to 1000 cycles (1 μs). Attack detection times of previous work that addressed DoS attack detection in real-time systems fall in the same range (3-8 μs) [5, 7].

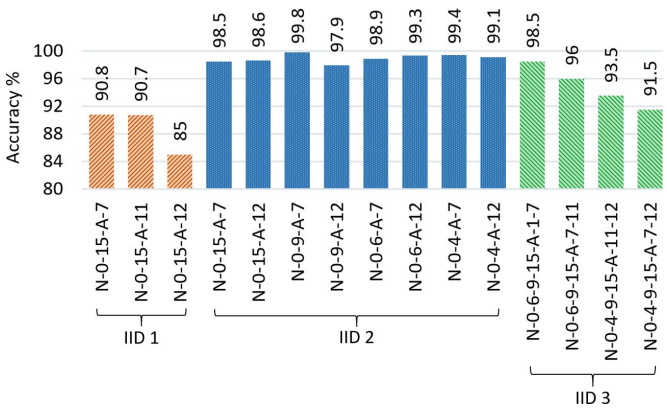

Figure 8: DoS attack detection accuracy for all tests in Table 2.

Figure 9 shows a comparison of common evaluation metrics used to evaluate ML models. The false positive rate ranges from 2.08% to 12.51% (on average), and the true negative rate ranges from 89.98% to 99.35% (on average), for all 3 iterations of test cases. Further, having precision, recall and f1 score in the range of 0.89 or above gives evidence to the high precision of the model. *Hence, results show that our approach is capable of detecting DoS attacks with high accuracy and in real time, irrespective of the number or the placement of MIPs and the number of applications running on the SoC. High attack detection accuracy is achieved not only if active and malicious IP placements match the training configurations, but also in new MIP placements, which the model has not been trained on.*

	Accuracy	False Positive Rate	True Negative Rate	Precision	Recall	F1 Score
IID1	88.83%	12.51%	89.98%	0.900	0.893	0.896
IID2	98.93%	2.08%	99.35%	0.993	0.991	0.992
IID3	94.95%	5.64%	95.29%	0.953	0.955	0.954

Figure 9: Comparison of evaluation metrics for test cases.

5 CONCLUSION

In this paper, we introduced a machine learning based DoS attack detection mechanism for NoC-based SoCs. We consider a widely explored threat model where a malicious IP floods the NoC with a large number of packets causing deadline violations, performance degradation or reduced energy efficiency. Unlike existing DoS attack detection methods that rely on highly predictable NoC traffic patterns and specific use cases, our approach is capable of detecting DoS attacks with high accuracy in real-time, in the presence of unpredictable NoC traffic patterns caused by diverse applications with input variations and application mappings. Experimental results demonstrated that non-linear models, such as gradient boosting, produce the best results for the given architecture and threat models. Our observations reveal that the key to achieving high accuracy is to carefully craft features out of the data extracted from NoC traffic. Our approach is capable of detecting DoS attacks with high accuracy in a wide variety of scenarios.

REFERENCES

[1] N. Agarwal et al. 2009. GARNET: A detailed on-chip network model inside a full-system simulator. In *ISPASS*. 33–42.
[2] N. Binkert et al. 2011. The Gem5 Simulator. *SIGARCH Comput. Archit. News* 39, 2 (2011), 1–7.
[3] T. Boraten et al. 2016. Secure model checkers for Network-on-Chip (NoC) architectures. In *GLSVLSI*. 45–50.
[4] S. Charles et al. 2018. Exploration of Memory and Cluster Modes in Directory-Based Many-Core CMPs. In *International Symposium on Networks-on-Chip (NOCS)*.
[5] S. Charles et al. 2019. Real-time Detection and Localization of DoS Attacks in NoC based SoCs. In *Design Automation and Test in Europe (DATE)*. 1160–1165.
[6] S. Charles et al. 2020. Lightweight Anonymous Routing in NoC based SoCs. In *Design Automation and Test in Europe (DATE)*. 334–337.
[7] S. Charles et al. 2020. Real-time Detection and Localization of Distributed DoS Attacks in NoC based SoCs. *IEEE Transactions on CAD (TCAD)* 39(12) (2020).
[8] S. Charles and P. Mishra. 2020. Lightweight and trust-aware routing in NoC-based SoCs. In *IEEE Annual Symposium on VLSI (ISVLSI)*. 160–167.
[9] S. Charles and P. Mishra. 2020. Securing network-on-chip using incremental cryptography. In *IEEE Annual Symposium on VLSI (ISVLSI)*. IEEE, 168–175.
[10] S Charles and P. Mishra. 2021. A Survey of Network-on-Chip Security Attacks and Countermeasures. *ACM Comput. Surv.* 54, 5, Article 101 (May 2021).
[11] C. Ciordas et al. 2004. An event-based network-on-chip monitoring service. In *HLDVT*. 149–154.
[12] L. Fiorin et al. 2008. A security monitoring service for NoCs. In *CODES+ISSS*. 197–202.
[13] M. Geller and P. Nair. 2018. 5G Security Innovation with Cisco. *Whitepaper Cisco Public* (2018), 1–29.
[14] S. Gupta et al. 2016. Vulnerable network analysis using war driving and security intelligence. In *ICICT*, Vol. 3. 1–5.
[15] Rajesh JS et al. 2015. Runtime Detection of a Bandwidth Denial Attack from a Rogue Network-on-Chip. In *NOCS*.
[16] Hyung Gyu Lee et al. 2008. On-chip communication architecture exploration: A quantitative evaluation of point-to-point, bus, and network-on-chip approaches. *ACM Trans. on Design Automation of Electronic Sys. (TODAES)* 12, 3 (2008), 1–20.
[17] P. Mishra and S. Charles. 2021. *Network-on-Chip Security and Privacy*. Springer.
[18] A. S. Shekhawat. 2018. Analysis of Encrypted Malicious Traffic. (2018).
[19] A. Sodani et al. 2016. Knights Landing: Second-Generation Intel Xeon Phi Product. *IEEE Micro* 36, 2 (2016), 34–46.
[20] K. Wang et al. 2019. High-performance, energy-efficient, fault-tolerant network-on-chip design using reinforcement learning. In *DATE*.
[21] D. Wentzlaff et al. 2007. On-Chip Interconnection Architecture of the Tile Processor. *IEEE Micro* 27, 5 (2007), 15–31.
[22] S. C. Woo et al. 1995. The SPLASH-2 programs: characterization and methodological considerations. In *ISCA*. 24–36.
[23] Y. J. Yoon et al. 2013. Virtual Channels and Multiple Physical Networks: Two Alternatives to Improve NoC Performance. *TCAD* 32 (12) (2013).