2022 IEEE Smartworld, Ubiquitous Intelligence & Computing, Scalable Computing & Communications, Digital Twin, Privacy Computing, Metaverse, Autonomous & Trusted Vehicles (SmartWorld/UIC/ScalCom/DigitalTwin/PriComp/Metaverse)

Lightweight Collaborative Inferencing for Real-Time Intrusion Detection in IoT Networks

Gabriel A. Morales, Jingye Xu, Dakai Zhu, and Rocky Slavin
Department of Computer Science
The University of Texas at San Antonio
San Antonio, Texas, USA
{gabriel.morales, jingye.xu}@my.utsa.edu, {dakai.zhu, rocky.slavin}@utsa.edu

***Abstract*—The security in Internet-of-Things (IoT) networks becomes increasingly important with the growing popularity of IoT devices and their wide applications (e.g., critical infrastructure monitoring). However, traditional intrusion detection systems (IDS) are not suitable for IoT networks due to their large resource requirements. Moreover, IoT networks tend to have multiple access points for IoT devices and thus benefit from a distributed framework to enable collaborative prevention of potential attacks. To this end, we propose a lightweight collaborative distributed network IDS (NIDS) based on widely-utilized machine learning (ML) models, which are trained through a federated learning framework with two known datasets. We evaluate the distributed NIDS using the trained ML models on an IoT network testbed under seven types of attacks in comparison with Snort (a state-of-the-art IDS) and a centralized implementation of our proposed NIDS. An offline benchmark is also designed to measure the system's performance with regard to resource usage and response time. Our results show that the proposed distributed NIDS outperforms Snort in identifying malicious traffic and achieves a much lower false positive rate compared to the centralized version in real-time for all seven types of network attacks tested.**

***Index Terms*—Intrusion Detection, Internet of Things, Machine Learning, Network Security**

I. Introduction

The Internet of Things (IoT) connects a multitude of devices with each other and is the backbone of smart homes, smart cities, critical infrastructure monitoring, and countless other applications [24]. With the integration of sensors, lightweight computation, and the proliferation of wireless and wired technologies on IoT platforms, consumers can easily interact with their surrounding physical world thoroughly through IoT networks [2]. This is widely apparent as an estimated 83 billion IoT-compatible devices are predicted to be in use by 2024 [22]. As a result of the widespread adoption of IoT infrastructures throughout various domains, such networks have become attractive to hackers and malicious applications [2]. This problem motivates the development of more IoT-conscious network intrusion detection systems (NIDS) that automatically monitor for potential threats to IoT networks [24], [14]. However, there remain challenges in deploying appropriate NIDSs on IoT networks.

First, traditional NIDSs, deployed on access points, may not be suitable for deployment on IoT networks in terms of network architecture. IoT networks have highly mobile architectures [6], [15] due to the way IoT devices are designed to communicate with a logical center node through network access points [23]. This approach enables the flexibility for IoT devices to switch from one access point to another access point when moving, thus increasing the potential risk for attacks [10]. Considering this architecture, even if one access point locally detects malicious activity, the attacker can switch to another access point to continue the attacks. Therefore, a flexible NIDS in which *collaboration* takes place, is more suitable: nodes, including access points, in a connected IoT system, can share local alerts with one another, creating a stronger deliberation in attack detection.

Second, IoT devices typically have constrained resources [4], [5] and thus have limited computational power to deploy complex and computationally heavy algorithms for security. Traditional NIDSs use traffic rules to detect malicious activities and are usually deployed on servers that are powerful and can detect risks quickly [21]. Moreover, the traffic rules need to be updated frequently and manually, which make traditional NIDSs less feasible for IoTs due to the resources required.

Finally, IoT networks often convey sensitive data [16], including personally identifiable information [3], that the data owners may not want to share outside of the network [12]. For this reason, intrusion detection should be conducted locally.

To address these challenges, we propose a collaborative inference-based NIDS for IoT networks. As a proof of concept and evaluation, we devised an IoT network testbed consisting of multiple access points and IoT devices to simulate the characteristics of a consumer-level IoT network. We then employ a federated machine learning system [17] to train machine learning (ML) models adopted in the proposed NIDS. Due to the distributed nature of the proposed NIDS, the access points can determine locally if traffic flows between devices are malicious with minimal overhead to the IoT network.

We evaluate the performance of the proposed NIDS with five widely-utilized ML models and compare their results to the state-of-the-art NIDS, Snort [20], as well as a centralized (i.e., non-distributed, sequential) version of the proposed NIDS framework to evaluate both the system's ability in detecting malicious traffic and its response time. The eval-

979-8-3503-4655-8/22/$31.00 ©2022 IEEE

DOI 10.1109/SmartWorld-UIC-ATC-ScalCom-DigitalTwin-PriComp-Metaverse56740.2022.00076

uation results show that, for all considered ML models, the distributed NIDS outperforms Snort in identifying malicious traffic. Specifically, Snort could only detect two out of seven attacks in the experiments, while the distributed NIDS detected six out of seven attacks at the node level. Moreover, the distributed nature of the NIDS results in a much lower false positive rate compared to the centralized version for all seven types of network attacks evaluated.

We summarize the contributions of this paper as follows:

1) **A Novel, IoT-Oriented Collaborative IDS Framework**: A lightweight collaborative inference framework is proposed and evaluated on resource-constrained devices such as Raspberry Pis. Real-time detection of potentially malicious traffic yields fast responses as low as one second and lower false positive rates with each model; we also show how the data delegation in the distributed, collaborative system is performed.
2) **Offline Benchmark**: As part of the evaluation of the framework, we present a benchmark application designed to add stress to the models and algorithms. The offline benchmark system measures disk space taken by the models, the amount of time to load them into memory, and the time it takes for each model to predict on a large set of flows. The stress-test for the flow table size is adjustable, along with the packet capture to load for the test.
3) **Experimental Evaluation of the Proposed NIDS**: We evaluate the proposed NIDS powered with five ML models trained by two known datasets including seven attack types on an IoT network testbed. Compared to the state-of-the-art production-level NIDS, Snort, the proposed framework is generally more accurate and acts more rapidly in detecting diverse attacks.

The remainder of this paper is organized as follows. Section II presents background and closely related work. The main ideas for the proposed approach are discussed in Section III. Section IV presents an evaluation of the approach and Section V discusses potential threats to validity. Section VI presents our conclusions and outlines our future work.

II. Background and Closely Related Work

An *intrusion detection system* (IDS) automates the process of monitoring for potential threats to a system. These threats may come in the form of malware, adversaries, or any other malicious actor attempting to gain unauthorized access to endpoint devices and the network. A *network intrusion detection system* (NIDS) is one such system designed to facilitate this process and monitor suspicious activities occurring across the network through traffic packets or flows [21]. Traditionally, how well a NIDS performs highly depends on the detection rules that are designed by analyzing the specific characteristics of the attack flows. The rules are highly flexible in that they can be updated to detect new attacks. However, this incremental update method also makes these rules complicated and hard to maintain. Moreover, the rules are usually released after attacks become popular and attract network administrators' notice. Snort is the foremost open source NIDS [20]. It uses a series of rules that help define the malicious network activity and find packets that match against them. In turn, Snort can generate alerts and take actions for users. It can detect most network attacks but requires users to update rules periodically.

A *traffic flow* is defined as containing the common networking attributes of two nodes that communicate with each other. For example, a standard flow contains the Internet Protocol (IP) address, Media Access Control (MAC) address, ports, duration of the communication, package payload, along with statistical information. This information is organized into corresponding forward (source to destination) and backward (destination to source) directions for a single flow. The data structure of many flows together can be viewed as a *flow table*. Based on the aggregate information in a single flow, node network behavior can be collected. As such, they can then be used by a NIDS to differentiate between abnormal and benign traffic flows.

Various novel intelligent NIDS have been proposed demonstrating that traditional machine learning algorithms can be applicable in lightweight applications, without sacrificing detection and model performance [7], [9], [11]. Commonly, collaboration is implemented by *collaborative learning*. For instance, several K-Means variants via distributed computation of alerts from various local IDS systems were proposed by Mokry et al. [18]. In other cases, collaboration is achieved through distribution and coordination of elements within such a system. Igbe et al. [8] modeled a non-centralized NIDS based on artificial immune systems. Dat-Thinh et al. [4] implemented a collaborative mechanism by way of a multi-staged, multi-model approach. This is a non-distributed collaborative process whereby the IoT traffic detection is split up into three pieces: detecting device type, detecting whether or not this particular traffic is anomalous, and finally detecting the specific attack type. Notably, the aforementioned systems implement their collaborative process during *training*. Our system implements a collaborative algorithm during real-time *predictions* .

McMahan et al. [17] introduced a distributed collaborative machine learning approach known as *federated learning*. This approach enables distributed nodes to collaboratively learn and update any shared model while keeping all the training data local. Federated learning can be a valuable approach for IoT NIDSs as it does not require the need to upload the privacy-concerned data to a central node but still can use them to train NIDS models efficiently.

III. Methodology and Implementation

Figure 1 illustrates the overview of the proposed NIDS framework with collaborative inference and machine learning models. From left to right: first, our chosen datasets are preprocessed to sanitize the traffic flows and combine them.

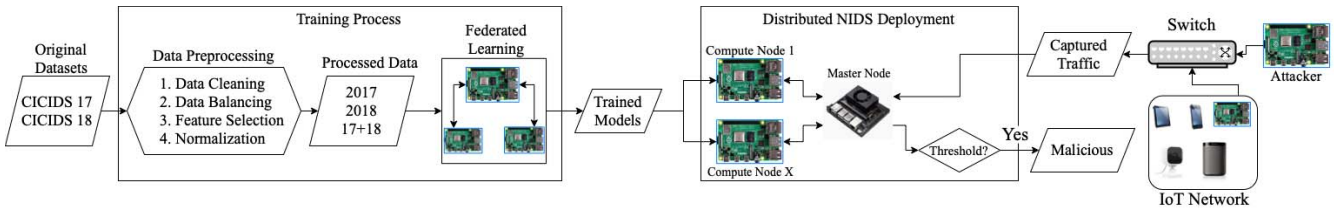

Fig. 1. Overview of the Proposed NIDS Framework with Federated ML models and Collaborative Inference.

Then, these datasets are used to train the adopted ML models through a federated learning system, which are subsequently saved for deployment. In the deployment stage, as shown in the right part of the figure, each trained ML model is loaded into the distributed compute nodes (representing access points) in the IoT network. For ease of processing and evaluation, network traffic is captured from the switch and routed into a master node which will orchestrate the collaborative inference and make the final decisions of benign or malicious network flows. Notably, the master node can also process all the network traffic flows using one of the trained ML models (i.e., a centralized version of the NIDS).

In a distributed IoT network, the task of evaluating a network flow can occur more efficiently in real-time since subdivided captures are delegated to each computing node by the master node (as shown in Figure 1). Each computing node predicts on its own sets of data, which necessitates a collaborative scheme for deliberation as traditional voting is not sufficient and difficult to scale.

A. Distributed Collaborative Framework Design

Fig. 2. Concurrent Distributed NIDS Data Flow

The proposed lightweight collaborative framework maintains fairness and operates in real-time by yielding a fast response. The system encapsulates a single responsibility in each compute node, which reports to the master node. Each compute node contains separate evidence buffers to which local predictions are mapped. Predictions made raise the count of benign or malicious alerts within the particular node's buffer. Once any of the benign or malicious thresholds are reached within a compute node, the evidence is then sent out to the master node, which maintains its own local buffer and thresholds. Once enough compute nodes have reported the same judgement over time, the master node makes the final determination. This careful deliberation allows the system to provide additional fairness so as to limit false positive rates within the system, regardless of the model used.

We choose to use the Dask [19] distribution framework due to its plug-and-play functionality: any computing device with Dask installed may become a component in the cluster via a single command-line invocation. Furthermore, Dask provides straightforward APIs to submit computing tasks to the cluster. The NIDS is divided into two primary computation components: the master node and the compute nodes. To use Dask, the master node will be the server running the `dask-scheduler` service, responsible for transparently managing the cluster and submitting computation tasks. Any device can become a compute node by running the `dask-worker` service. The compute nodes may come and go on-demand without interrupting the cluster, making the system easily scalable. In order to develop a NIDS which is compatible with our models, live traffic must be converted into the flow formats the models were trained on. Live data is captured from our test network, which may either be benign or malicious depending on if we are conducting attacks. For ease of implementation, these captures are saved using a `tempfile` system call to create flows using NFStream [1] from each capture. Furthermore, all models have their weights, along with their respective feature scalers, saved. In deployment, both are loaded. In what follows, the algorithmic steps for each component is described in more detail.

1) Master Node: The master node computation is divided into four concurrent tasks. Outside of the Dask scheduler, which is initialized on startup, three threads are in operation. As shown in Figure 2, these three threads are labelled numerically with four algorithmic sub-tasks each. Thread 1 captures data from the Network Switch plugged into the Ethernet

interface using the Python networking package, Scapy (step 1A). These data packets are then saved to random access memory (RAM) and processed into individual flows via the NFStream package, then turned into flow tables (steps 1B & 1C). To perform effective data delegation, historical flows are used by appending each new flow table to the table generated in the prior iteration. The resulting aggregate flow table is split into smaller fragments amongst the flow table queue for thread 2, and flushed for the next iteration (step 1D).

Thread 2 first takes the next available flow table from a thread-safe queue, and then passes it as a parameter to be scheduled for a task on the cluster (steps 2A & 2B). Each submitted task is handled by the Dask scheduler service running in the background. When a task is submitted, a Dask future object, an asynchronous object which allows non-blocking computations, is returned immediately for the next step (step 2C). This future, representing the cluster task, is then added to the `DASK_FUTURE_QUEUE` on the system for thread 3 (step 2D).

For the steps in thread 3, Algorithm 1 is also referenced. Thread 3 obtains the next available future from the `DASK_FUTURE_QUEUE` (step 3A, line 27) and then the `result` method is called on the future. This is a blocking call, so instead we check if the task is pending and requeue it if so (step 3B, lines 28-31), otherwise the result can be obtained. These results are encoded from the evidence buffers in the compute nodes. Results are then submitted to the master evidence buffer (step 3C, lines 31-35). The final step is observing the master node's thresholds and checking if either is exceeded. If the benign threshold, `MN_BENIGN_THRESH`, is exceeded, enough evidence has been accrued to judge that activity is benign. If the malicious threshold, `MN_MAL_THRESH`, is exceeded, the activity is judged as malicious. Finally, the respective entry is flushed (step 3D, lines 36-42).

2) Compute Node(s): In Figure 2 Compute Node X denotes one or more devices joining as a `dask-worker` to the cluster. Algorithm 1 is also referenced. Compute nodes have two steps: predict on the received flow table (lines 3-4) and map the predictions to the local evidence buffer. For each prediction, the MAC addresses are mapped to the buffer and their judgment count is increased (lines 6-10). If any of the thresholds are exceeded, the MAC address is encoded with the node's judgment of benign or malicious and the count (lines 11-14 & 15-19). If the benign threshold, `CN_BENIGN_THRESH`, is exceeded, it is reset to zero for future flows of this device. If the malicious threshold, `CN_MAL_THRESH`, is exceeded, its counts are gradually reset by half to maintain a level of suspicion over time, thus ensuring real malicious activity is not ignored.

The source code for Algorithm 1 and the proposed NIDS system is open source and available on GitHub[1].

[1]https://github.com/jingye-xu/DistriLearn

Algorithm 1 Distributed Collaborative Flow Detection NIDS

```
1:  function RUN_INFERENCE(dataframe)
2:      model.get_instance() // load once and reference from mem-
    ory
3:      predictions ← model.predict(dataframe)
4:      result ← 0
5:      mac_index ← 0
6:      while mac_index ≤ dataframe.length do
7:          mac ← dataframe[mac_column][mac_index]
8:          prediction ← predictions[mac_index]
9:          buffer ← {benign: 0, malicious: 0}
10:         buffer[mac][prediction] += 1
11:         if buffer[mac].benign ≥ CN_BENIGN_THRESH then
12:             result ← {mac : buffer[mac].benign_count}
13:             buffer.flush_benign(FULL)
14:         end if
15:         if buffer[mac].malicious ≥ CN_MAL_THRESH then
16:             result ← {mac : buffer[mac].malicious_count}
17:             buffer.flush_benign(ONE_HALF)
18:             buffer.flush_malicious(ONE_HALF)
19:         end if
20:         mac_index += 1
21:     end while
22:     return result
23: end function
24:
25: function RESULTS
26:     while not shutdown do
27:         dask_future ← DASK_FUTURE_QUEUE.dequeue()
28:         if dask_futue.status == PENDING then
29:             DASK_FUTURE_QUEUE.enqueue(dask_future)
30:         end if
31:         result ← dask_future.result()
32:         mac ← result[0]
33:         prediction ← result.prediction_type
34:         type_count ← result[prediction].count
35:         master_buffer[mac][prediction] += type_count
36:         if buffer[mac].benign ≥ MN_BENIGN_THRESH then
37:             // Report benign judgment
38:             buffer.flush_benign(FULL)
39:         end if
40:         if buffer[mac].malicious ≥ MN_MAL_THRESH then
41:             // Report malicious judgement
42:             buffer.flush_malicious(FULL)
43:         end if
44:     end while
45: end function
```

B. Datasets and Feature Engineering

To achieve differentiation between malicious and benign traffic, we selected two datasets provided by the Canadian Institute for Cyber Security (CIC): CIC-IDS 2017[2] and CSE-CIC-IDS 2018[3]. These contain organized network attacks commonly used by adversaries. We restrict our scope to these datasets for simplicity and defer external datasets to future work. Since many IoT devices are connected on the Wi-Fi network, all attacks conductible on non-IoT devices are applicable to IoT devices as well.

We implement a preprocessing phase which ensures the

[2]https://www.unb.ca/cic/datasets/ids-2017.html
[3]https://registry.opendata.aws/cse-cic-ids2018

TABLE I
FEATURES SELECTED FROM CIC DATASETS

Feature Name	Description
Destination Port	Communication Protocol Port.
Flow Duration	Duration of a flow.
Total Fwd Packets	Total forward packets.
Total Backward Packets	Total backward packets.
Total Length of Fwd Packets	Total length of forward packets.
Total Length of Bwd Packets	Total length of backward packets.
Fwd Packet Length Max	Maximum length of forward packets.
Fwd Packet Length Min	Minimum length of forward packets.
Fwd Packet Length Mean	Mean length of forward packets.
Fwd Packet Length Std	Standard deviation length of forward packets.
Bwd Packet Length Max	Maximum length of backward packets.
Bwd Packet Length Min	Minimum length of backward packets.
Bwd Packet Length Mean	Mean length of backward packets.
Bwd Packet Length Std	Standard deviation length of backward packets.
Flow Bytes per second	Number of bytes transmitted per second.
Flow Packets per second	Number of packets transmitted per second.
Flow IAT Mean	Mean inter-arrival time in the flow.
Flow IAT Max	Maximum inter-arrival time in the flow.
Flow IAT Min	Minimum inter-arrival time in the flow.
Fwd IAT Total	Total inter-arrival time of forward packets.
Fwd IAT Mean	Mean inter-arrival time of forward packets.
Fwd IAT Std	Standard deviation inter-arrival time of forward packets.
Fwd IAT Max	Maximum inter-arrival time of forward packets.
Fwd IAT Min	Minimum inter-arrival time of forward packets.
Bwd IAT Total	Total inter-arrival time of backward packets.
Bwd IAT Mean	Mean inter-arrival time of backward packets.
Bwd IAT Std	Standard deviation inter-arrival time of backward packets.
Bwd IAT Max	Maximum inter-arrival time of backward packets.
Bwd IAT Min	Minimum inter-arrival time of backward packets.
Fwd Packets per second	Number of forward packets transmitted per second.
Bwd Packets per second	Number of backward packets transmitted per second.
Min Packet Length	Minimum packet length for a flow.
Max Packet Length	Maximum packet length for a flow.
Packet Length Mean	Mean length of packets in a flow.
Packet Length Std	Standard deviation length of Packets in a flow.
Packet Length Variance	Variance in the length of packets in a flow.
RST Flag Count	Number of RST flags transmitted for a flow.
Average Packet Size	Average packet size transmitted in a flow.

quality and performance of the models. In Figure 1, this phase is summarized in the training process' data preprocessing box. Our feature set is listed in Table I. The original datasets are also merged to create a new, larger one, resulting in organic upsampling. Table II shows each of the datasets' class distributions after processing.

The NIDS is designed to differentiate between benign and potentially malicious traffic. However, each of the flows in the datasets is labelled as one of seven network-based attack categories: Brute Force (FTP/SSH), DoS/DDoS, Heartbleed, Web Attacks, Infiltraion, Botnet, Portscan. Hence, each attack flow is relabeled as `malicious` while flows labeled `benign` are unchanged. This strategy allows the NIDS to easily inter-operate between datasets and aggregate them without accommodating change in class sizes. Additionally, this allows the datasets to be used for binary classification.

TABLE II
DATASET CLASS DISTRIBUTIONS

	CIC-IDS17	CIC-IDS18	CIC17+18
Total	851,482	2,702,734	3,554,216
Malicious	425,741	1,351,367	1,777,108
Benign	425,741	1,351,367	1,777,108

C. Machine Learning Models

Five widely-utilized machine learning models [9], [13] are chosen for the proposed NIDS, which are trained using the three processed datasets as discussed in the last section. Implemented in Sklearn, we use: The Support Vector Machine (SVM), Logistic Regression (LR), K-Nearest Neighbors (KNN), and Random Forest (RF). We implemented the Neural Network (NN) using PyTorch. Each of the models are trained using federated learning with the average strategy.

D. Network Architecture of the Testbed

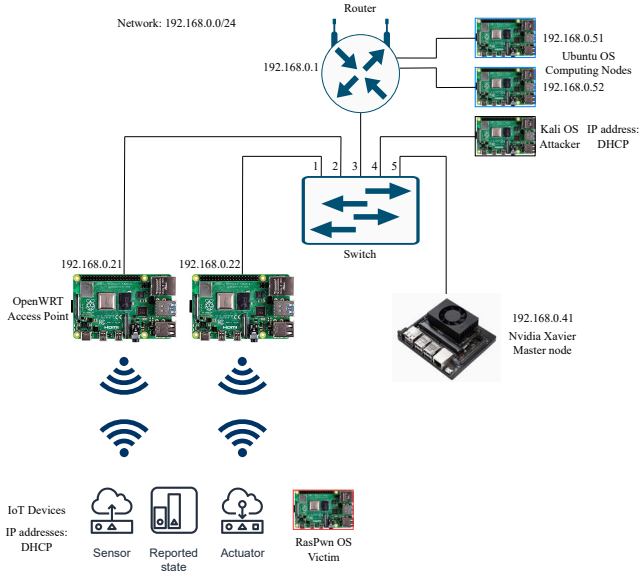

Fig. 3. Network architecture of the IoT Network Testbed.

Figure 3 shows the basic network topology for the IoT network testbed designed to simulate a real-world smart home environment upon which the NIDS system is implemented. For the router, the testbed utilizes an ASUS RT-AC1200GE disconnected from the Internet to prevent unwanted traffic. An NVIDIA Jetson Xavier with a 6-core 64-bit CPU and 8 GB Memory running Ubuntu 18.04 serves as the master node for our cluster. Two Raspberry Pi (RPI) devices act as access

TABLE III
DISTRIBUTED SYSTEM REACTION TIMES (S)

	Network Discovery		DoS		Brute Force		Vuln. Discovery		Port Scan		Medusa Brute		Fuzzing	
	Relaxed	Load	Relaxed	Load	Relaxed	Load	Relaxed	Load	Relaxed	Load	Relaxed	Load	Relaxed	Load
LR (2017)	F	F	2	4	4	4	6	7	6	11	1	1	1	1
SVM (2017)	F	F	1	1	2	2	12	12	4	4	1	1	1	1
KNN (2017)	F	F	2	3	4	4	7	7	F	F	3	2	2	3
NN (2017)	F	F	1	2	2	2	2	2	14	17	2	2	1	2
RF (2017)	F	F	F	F	30	50	F	F	F	F	1	20	F	F
LR (2018)	F	F	2	3	1	2	11	12	1	2	1	2	1	2
SVM (2018)	1	1	1	1	1	1	12	12	1	1	1	1	1	1
KNN (2018)	F	F	4	7	3	4	13	14	6	6	2	3	2	5
NN (2018)	F	F	2	3	3	3	12	13	1	2	1	2	1	2
RF (2018)	F	F	1	2	2	5	13	15	3	3	1	1	1	2
LR (17+18)	F	F	1	2	2	2	12	13	1	1	1	2	1	1
SVM (17+18)	F	F	1	2	2	2	12	12	1	2	2	2	1	2
KNN (17+18)	F	F	3	4	4	7	13	14	6	10	3	3	2	8
NN (17+18)	F	F	1	4	2	2	13	13	2	2	2	2	1	1
RF (17+18)	F	F	2	2	2	3	13	16	2	2	1	2	1	1
Snort	F	F	F	F	2	F	11	42	F	F	F	F	F	F

points running OpenWRT, an open-source router OS. Various IoT devices such as smart phones, speakers, and cameras are connected to these RPI access points. Two other RPI 4B devices act as compute nodes in our IDS: 4-core CPU, with 2GB and 4GB of memory. Finally, we have two RPI devices for pen-testing. One RPI runs Kali Linux as the attacker, the other runs a RasPwn OS web server as the victim.

A Netgear (GS305E) switch is placed between the router and the rest of the network, including the NIDS nodes. There are five ports to which the router, all RPI devices, and the master node are connected and have been assigned static IP addresses as shown in Figure 3. All traffic flowing to and from port 3 (router) is also mirrored to port 5 (master node). We assume that the attacker has no knowledge of the IDS system to prevent unwanted interference as discussed in Section V.

IV. EVALUATION AND RESULTS

In this section, we address the research questions (RQ) below by evaluating the performance of the proposed NIDS.

- **RQ1:** Is a collaborative, inference-based NIDS viable compared to the state of the art?
- **RQ2:** Is the distributed, collaborative design of the NIDS system beneficial?
- **RQ3:** How well do different machine learning algorithms balance performance and resource consumption when deployed in the NIDS?

1) Baseline: Snort was chosen as the baseline to evaluate the performance of the proposed, as it supports multiple platforms [20]. The GPLv2 Community rules were used for the deployment of Snort. To detect all the traffic flows, Snort was configured to alert when it detects malicious activity instead of blocking it. To evaluate **RQ2**, a centralized version of the proposed NIDS was also designed which runs data collection and performs model inferences solely on the master node (i.e., no parallelism or distribution is exploited). This centralized NIDS acts on all generated flows immediately.

2) Time Calibration: NTP settings were configured on all nodes in the system such that their local time is synchronized. This enables the approximate calculation of response times by obtaining the difference from the time of starting attacks and the time of detecting malicious flows.

3) Attack Design: We conducted seven different types of attacks to test successful detection and how long it will take to detect, described in Section III-B. The attack types are identical to those present in the CIC datasets. All attacks were launched using the Kali RPI device. We list them below along with the tools and commands we used to launch them:

- Network Discovery: `nmap`
- Denial of Service (DoS): `hping3`
- Brute Force: `nmap`
- Vulnerability Discovery: `nmap`
- Port Scan: `nmap`
- SSH/FTP Brute Force: `medusa` with SecLists[4]
- Web Fuzzing: `ffuf` with SecLists

Each of these attacks were conducted five times to measure the best response time overall. We divide our measurements into a `Relaxed` state, where the NIDS is running under normal conditions, and a `Load` state where the NIDS is running under a highly-loaded CPU using the `stress` command-line tool. We measure response times as $(detection\_time - attack\_start\_time)$ using NTP synchronization. Entries in Table III and Table IV contain a response time if the attack was successfully detected at *any point* in the experiment. If Snort, the centralized, or distributed systems were unable to successfully detect the attacks by their completion, the entries are marked as `F` (failed detection).

4) Offline Benchmark Design: To stress-test the models and simulate a network under load, we processed a 7.73 GB packet capture with the trained models loaded. As with our real-time design, we use NFStream to read the capture from disk quickly and generate the flowtable data structure. An adjustable limit of 5,001 entries was placed on the flowtable, which takes 1.4 MB of memory. As shown in Table V, a growth of disk space, prediction time, and load time is present

[4]https://github.com/danielmiessler/SecLists

TABLE IV
CENTRALIZED SYSTEM REACTION TIMES (S)

	Network Discovery		DoS		Brute Force		Vuln. Discovery		Port Scan		Medusa Brute		Fuzzing	
	Relaxed	Load	Relaxed	Load	Relaxed	Load	Relaxed	Load	Relaxed	Load	Relaxed	Load	Relaxed	Load
LR (2017)	1	2	1	1	4	3	11	11	F	F	1	1	1	1
SVM (2017)	2	2	1	1	2	1	12	12	1	1	1	1	1	1
KNN (2017)	1	2	1	1	4	3	11	11	F	F	1	1	1	1
NN (2017)	1	1	1	1	1	2	11	12	1	1	1	1	1	1
RF (2017)	F	F	F	F	6	8	F	F	F	F	1	1	F	F
LR (2018)	1	1	1	1	1	2	11	11	1	1	1	1	1	1
SVM (2018)	1	1	1	1	2	2	12	11	1	1	1	1	1	1
KNN (2018)	1	1	1	1	2	2	11	11	1	1	1	1	1	1
NN (2018)	1	1	1	1	2	2	11	11	1	1	1	1	1	1
RF (2018)	1	1	1	1	1	2	14	11	1	1	1	1	1	1
LR (17+18)	2	1	1	1	1	1	11	11	1	1	1	1	1	1
SVM (17+18)	1	1	1	1	2	2	12	12	1	1	1	1	1	1
KNN (17+18)	1	1	2	1	1	1	11	12	1	1	1	1	1	1
NN (17+18)	1	1	1	1	1	1	11	12	1	1	1	1	1	1
RF (17+18)	1	1	1	1	2	1	12	11	1	1	1	1	1	1
Snort	F	F	F	F	2	F	11	42	F	F	F	F	F	F

TABLE V
OFFLINE BENCHMARK TESTS

	Disk Space	Model Init. Time (s)	Prediction Time (s)
LR (2017)	1.02 KB	0.114	0.0008
SVM (2017)	926 Bytes	0.00087	0.0003
KNN (2017)	164.68 KB	0.355	6.34
NN (2017)	9.02 KB	0.007	0.06
RF (2017)	162.19 KB	0.355	0.016
LR (2018)	1.02 KB	0.08	0.0006
SVM (2018)	926 Bytes	0.001	0.0013
KNN (2018)	522.72 MB	0.76	19.58
NN (2018)	9.022 KB	0.007	0.06
RF (2018)	185.11 KB	0.31	0.02
LR (17+18)	1.02 KB	0.07	0.001
SVM (17+18)	926 Bytes	0.0011	0.00039
KNN (17+18)	685.01 MB	0.9	24.69
NN (17+18)	9.022 KB	0.01	0.046
RF (17+18)	185.81 KB	0.21	0.016

TABLE VI
FALSE POSITIVE RATE COMPARISON

	Centralized (%)	Distributed (%)
LR (2017)	15	**8**
SVM (2017)	9.3	**3**
K-NN (2017)	3	**2**
NN (2017)	4	**3**
RF (2017)	0	0
LR (2018)	32	**16**
SVM (2018)	38	**28**
K-NN (2018)	2	**0.9**
NN (2018)	6	**4**
RF (2018)	2	**0.3**
LR (17+18)	13	**4**
SVM (17+18)	20	**12**
K-NN (17+18)	5	**3**
NN (17+18)	27	**4**
RF (17+18)	1.4	**0.2**

for some models. This gives a primary indication of how the model will scale in real-time.

5) False Positive Rate Comparisons: To measure the false positive rates, one computing device generated passive benign traffic. During this time, no attacks, malicious activity, or external interventions took place. We ran all models for 30 minutes each to obtain how much of the benign traffic was flagged as malicious for an extended period of time. The rates were calculated relative to the number of individual flows generated for that system (distributed system and centralized system): $(system\_malicious\_flags \div system\_total\_flags) * 100$

Next we discuss our research questions.

RQ1: We compared our distributed collaborative NIDS to Snort using the same attacks from Section IV-3 and the standard community rules. In the normal state, Snort detected two of the seven attack types: brute force in two seconds and vulnerability discovery in 11 seconds, while the other types were undetected. When under stress, Snort only detected one attack: vulnerability discovery in 42 seconds. Tables III and IV display Snort's performance for each attack under these states. In contrast, the distributed NIDS system, shown in Table III, detected all of the attack types with faster response times when relaxed and stressed. For instance, the SVM (2018) model detected all of these attacks in one second, and only 12 seconds for vulnerability discovery. This shows that, in comparison to state of the art, the proposed NIDS is viable.

RQ2: We compared our distributed collaborative NIDS to its centralized counterpart in addition to measuring the distributed system's overhead times.

The centralized system was designed with no parallelism or distribution, thus it infers on incoming flows immediately. As shown in Table IV, the slowest reacting models were trained on the 2017 dataset: KNN detected six out of the seven attacks and RF only detected two attack types. These response times were also the highest in difference of the three datasets for the brute force attack. For the 2018 dataset and the combined dataset, all of the attacks were detected, with a maximum of a one second shift in response times between the two sets. Furthermore, load factors for each experiment added no overhead resulting in $\pm 1s$ differences.

The thresholds set for the master and compute nodes control the strength of deliberation prior to flags being raised. As a result, the thresholds will also affect false positive

TABLE VII
REAL-TIME METRICS

	RAM Reference (ms)	Map (ms)	Average Inference (ms)	Memory (MB)	CPU (%)
LR (2017)	2E-3	2E-1	8.0E-1	112	4
SVM (2017)	2E-3	4E-1	9.0E-1	123	5
KNN (2017)	2E-3	2E-1	9.1E1	649	20
NN (2017)	2E-3	2E-1	1E0	197	4
RF (2017)	2E-3	2E-1	4.9E0	123	9
LR (2018)	1E-3	1E-1	7E-1	132	4
SVM (2018)	6E-3	4E-1	1E0	121	5
KNN (2018)	2E-3	4E-1	6.2E2	631	39
NN (2018)	2E-3	2E-1	1.9E0	198	6
RF (2018)	2E-3	3E-1	7E0	138	6
LR (17+18)	2E-3	3E-1	9E-1	135	5
SVM (17+18)	2E-3	6E-1	9E-1	134	5
KNN (17+18)	2E-3	2E-1	7.1E2	794	54
NN (17+18)	1E-3	3E-1	1.6E0	198	4
RF (17+18)	1E-3	3E-1	9.7E0	139	6

TABLE VIII
TRAINING METRICS AND RESULTS

	Accuracy (%)	F1	Precision (%)	Recall (%)
LR (2017)	91.62	0.91	88.28	95.84
SVM (2017)	91.66	0.92	87.91	96.64
KNN (2017)	99.05	0.99	98.63	99.47
NN (2017)	90.95	0.91	87.69	95.29
RF (2017)	98.52	0.98	99.17	97.86
LR (2018)	86.35	0.86	83.43	90.69
SVM (2018)	81.70	0.82	80.21	84.11
KNN (2018)	94.97	0.94	97	92
NN (2018)	88.75	0.88	86.75	90.28
RF (2018)	93.33	0.93	97	89.42
LR (17+18)	85.72	0.86	82.60	90.60
SVM (17+18)	82.11	0.8	79.67	86.38
KNN (17+18)	95.9	0.95	97.24	94.52
NN (17+18)	84.56	0.85	81.23	90.06
RF (17+18)	93.81	0.93	96.199	91.24

rates. In practice, these thresholds can be adjusted to suit the desired sensitivity of the NIDS. We seek to find a balance between rapid detection and strong deliberation. We used two compute nodes with the malicious threshold set to 10 and the benign threshold set to 26. Our master node's malicious threshold was set to two and 20 for benign. Similar to the centralized system, the slowest reacting models are present in the 2017 dataset: RF only detected one attack of seven, and the KNN detected five, shown in Table III. None of the models detected network discovery, which is expected since this attack was not present in the original datasets; this particularly differentiates the false positive rates present in Table VI. Our distributed collaborative system exhibited a large *reduction* in false positive rates due to the stronger deliberation the framework imposes. As displayed in Table III, reaction times remained low and rarely had a shift larger than $\pm 2s$ as compared to centralized results. This implies that our system remains efficient and responsive. Furthermore, our distributed system provides a reduction in false positive rates through the stronger deliberation.

Finally, we limited our captures to 90 packets per round, which averaged 0.704s on steady traffic. Importantly, this is unavoidable overhead. The total overhead added from writing to, and reading captures from, RAM was 0.0824s on average, with the maximum capture being 105KB. The longest amount of time taken is converting these flows into a flow table data structure (dataframe), at an average of 0.13s. Overall, the expected overhead is 0.2214s to submit real-time flow tables to our cluster.

RQ3: To evaluate the different models, we compared the metrics obtained by the models during training on each dataset in tandem with how each of them perform in real-time. Further, we observed their offline stress test performance.

Our real-time metrics (Table VII) represent the metrics that correspond to each component of the NIDS taken by a task on the compute nodes: RAM indicates the time it takes to reference the model from memory each time, Map indicates the time the algorithm takes to map from predictions to the evidence buffer, Average Inference is the amount of time on average for a model to predict on the incoming flows, Memory usage and CPU represent the resource utilization of the compute node. For the Sklearn models, a 35% train/test split is applied. For the neural network, a train/test split of 25% is applied.

CIC-IDS 2017 is the smallest dataset when balanced, which is well-indicated by the performance of the models. Shown in Table VIII, each of these models achieved high accuracy and F1-Score; precision remained high with the lowest being 87% and recall reaching 99%. When referenced with Table VII, we see that each of the metrics took less than a millisecond. KNN used the highest amount of resources and time, with RF being second highest, exceeding a millisecond.

For the 2018 dataset and the combined dataset, the RAM and Map metrics were unchanging with the best at one-thousandth of a millisecond each. The impact of the datasets on the models was such that accuracy fell below 90% with the worst reaching 81%. F1 fell as low as 0.82, with precision reaching 79% and recall reaching 84%. In general, increasing dataset size slightly degrades overall training performance; however, this is also indicative of less potential to overfit. KNN did not scale well in our system: as the dataset increases, so too did the resource consumption. In addition, this is further evident by our offline benchmark where this particular model takes the most disk size, initialization time, and prediction times reaching 25 seconds as the dataset size increases. The most efficient model with the highest prediction performance was Random Forest, also yielding the lowest false positive rates.

V. THREATS TO VALIDITY

In our framework, there are two separate thresholds for the master node and the compute nodes. Depending on what these thresholds are, detection rates and false positives may vary. With a higher threshold, false positives are fewer and detection rates are slower. With a lower threshold, the false positives are more frequent and detection rates are quicker. We do not reset the malicious buffer entries in the compute

nodes at first since it ensures a level of suspicion remains for when attacks do occur. Therefore, it is important to find a balance for these numbers to maintain efficiency and fairness in the system.

Flows are non-deterministic in real-time, causing predictions to vary. This is not an issue with our implementation, but rather an important observation. This is exhibited when we launch attacks multiple times (as seen in the tables in Section IV). Often malicious flows are detected and some others can be missed. This is based on when the flows start forming, and how closely the patterns resemble malicious or benign activity.

There are three categories of memory that can be measured: process, managed, and unmanaged (old & recent). Memory use may increase to a point of high consumption. This is due to unused memory being out of our manual control: unused memory in Dask may not be released back to the operating system immediately. We seek to improve on this constraint in future work.

Finally, when attacks begin to target the NIDS itself, Dask notices incoming connections and reports them. Transmission of information on the cluster may be interrupted, so we make the assumption that the adversary is unaware of the NIDS, as this is beyond the scope of the current work.

VI. Conclusions and Future Work

In this paper, we propose a lightweight collaborative distributed NIDS based on machine learning algorithms specifically for IoT networks. To evaluate the feasibility of applying the proposed NIDS, we designed an offline benchmark to stress-test the models and assess how they scale under load, then measure their performance in real-time. The results show that the proposed NIDS uses less memory and CPU, while not sacrificing response time. In addition, we evaluate the attack detection ability of the distributed NIDS compared to the state-of-the-art system (Snort) and a centralized version of the NIDS. For the seven types of attacks we considered, both the distributed NIDS and centralized version outperform Snort. Snort detected two out of seven attacks while the distributed NIDS detected more attacks on average. The centralized version could also detect all attacks but it used more resources on a single node. Furthermore, it could not perform collaborative detection with other nodes, which limits its flexibility. The distributed NIDS is shown to largely reduce false positives while maintaining fair judgments and faster response times. Considering the resource usage, false positive rates, response times, and deployment structure, the proposed distributed NIDS is more applicable to IoT networks.

Additional facets are open to future research. We seek to study a malicious node within the NIDS itself, where attacks can interrupt the system. We will expand the deliberations to more complex models using ensemble approaches and tailor response actions towards specific attacks. Attacks constantly evolve and all possible cases are not covered, even with machine learning. Solutions include addressing concept drifts and aggregating larger datasets along with our own. Finally, we plan to reduce overhead and memory usage even further than we currently have, and unifying the NIDS directly onto network devices (e.g., router, smartphone, etc) for better responses.

References

[1] Z. Aouini and A. Pekar. Nfstream: A flexible network data analysis framework. *Computer Networks*, 204:108719, 2022.

[2] L. Babun et al. A survey on IoT platforms: Communication, security, and privacy perspectives. *Computer Networks*, 192:108040, 2021.

[3] E. Bertino. Data privacy for IoT systems: Concepts, approaches, and research directions. In *IEEE International Conference on Big Data (Big Data)*, pages 3645–3647. IEEE, 2016.

[4] N. Dat-Thinh, H. Xuan-Ninh, and L. Kim-Hung. MidSiot: A multistage intrusion detection system for internet of things. *Wirel. Commun. Mob. Comput.*, 2022:1–15, Feb. 2022.

[5] M. Eskandari, Z. H. Janjua, M. Vecchio, and F. Antonelli. Passban ids: An intelligent anomaly-based intrusion detection system for iot edge devices. *IEEE Internet of Things Journal*, 7(8):6882–6897, 2020.

[6] S. M. Ghaleb, S. Subramaniam, Z. A. Zukarnain, and A. Muhammed. Mobility management for iot: a survey. *EURASIP Journal on Wireless Communications and Networking*, 2016(1):1–25, 2016.

[7] V. Hnamte, G. Balram, et al. Implementation of Naive Bayes Classifier for Reducing DDoS Attacks in IoT Networks. *JOURNAL OF ALGEBRAIC STATISTICS*, 13(2):2749–2757, 2022.

[8] O. Igbe et al. Distributed network intrusion detection systems: An artificial immune system approach. In *IEEE First International Conference on Connected Health: Applications, Systems and Engineering Technologies (CHASE)*, pages 101–106, 2016.

[9] S. U. Jan et al. Toward a lightweight intrusion detection system for the internet of things. *IEEE Access*, 7:42450–42471, 2019.

[10] H. Jradi et al. Overview of the mobility related security challenges in lpwans. *Computer Networks*, 186:107761, 2021.

[11] Y. Kayode Saheed et al. A machine learning-based intrusion detection for detecting internet of things network attacks. *Alexandria Engineering Journal*, 61(12):9395–9409, 2022.

[12] M. A. Khan and K. Salah. Iot security: Review, blockchain solutions, and open challenges. *Future Generation Computer Systems*, 82:395–411, 2018.

[13] A. Khraisat et al. Survey of intrusion detection systems: techniques, datasets and challenges. *Cybersecurity*, 2(1), Dec. 2019.

[14] H.-J. Liao et al. Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, 36(1):16–24, 2013.

[15] J. E. Luzuriaga et al. Handling mobility in iot applications using the mqtt protocol. In *Internet Technologies and Applications (ITA)*, pages 245–250, 2015.

[16] M. Marjani et al. Big iot data analytics: Architecture, opportunities, and open research challenges. *IEEE Access*, 5:5247–5261, 2017.

[17] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.

[18] L. Mokry et al. Efficient and privacy-preserving collaborative intrusion detection using additive secret sharing and differential privacy. In *IEEE International Conference on Big Data (Big Data)*, pages 3324–3333, 2021.

[19] M. Rocklin. Dask: Parallel computation with blocked algorithms and task scheduling. In *Proceedings of the 14th python in science conference*, volume 130, page 136. Citeseer, 2015.

[20] M. Roesch et al. Snort: Lightweight intrusion detection for networks. In *Lisa*, volume 99, pages 229–238, 1999.

[21] K. Scarfone, P. Mell, et al. Guide to intrusion detection and prevention systems (idps). *NIST special publication*, 800(2007):94, 2007.

[22] S. Smith. Iot connections to reach 83 billion by 2024, driven by maturing industrial use cases. *Accessed: Apr*, 10:2021, 2020.

[23] S. Tomovic et al. Software-defined fog network architecture for iot. *Wireless Personal Communications*, 92(1):181–196, 2017.

[24] Y. Zhang et al. Efficient and Intelligent Attack Detection in Software Defined IoT Networks. *IEEE International Conference on Embedded Software and Systems, ICESS 2020*, dec 2020.