IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received November 11, 2021, accepted December 16, 2021, date of publication December 23, 2021,
date of current version January 18, 2022.

Digital Object Identifier 10.1109/ACCESS.2021.3137318

Investigating the Effect of Traffic Sampling on Machine Learning-Based Network Intrusion Detection Approaches

JUMABEK ALIKHANOV[1], (Student Member, IEEE), RHONGHO JANG[2], (Member, IEEE), MOHAMMED ABUHAMAD[3], (Member, IEEE), DAVID MOHAISEN[4], (Senior Member, IEEE), DAEHUN NYANG[5], (Senior Member, IEEE), AND YOUNGTAE NOH[1], (Member, IEEE)

[1]Department of Computer Science and Information Engineering, Inha University, Incheon 402-751, South Korea
[2]Department of Computer Science, Wayne State University, Detroit, MI 48202, USA
[3]Department of Computer Science, Loyola University Chicago, Chicago, IL 60626, USA
[4]Department of Computer Science, University of Central Florida, Orlando, FL 32816, USA
[5]Department of Cyber Security, Ewha Womans University, Seoul 03760, South Korea

Corresponding authors: Youngtae Noh (ytnoh@inha.ac.kr) and Daehun Nyang (nyang@ewha.ac.kr)

This research was supported by Global Research Laboratory (GRL) Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT (NRF-2016K1A1A2912757); National Research Foundation of Korea (NRF) grant funded by the Korean government (MSIT) (NRF-2019R1F1A1059898, NRF-2020R1A2C2009372); and by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2020R1A4A1018774).

ABSTRACT Machine Learning (ML) based Network Intrusion Systems (NIDSs) operate on flow features which are obtained from flow exporting protocols (*i.e.,* NetFlow). Recent success of ML and Deep Learning (DL) based NIDS solutions assume such flow information (*e.g.,* avg. packet size) is obtained from all packets of the flow. However, often in practice flow exporter is deployed on commodity devices where packet sampling is inevitable. As a result, applicability of such ML based NIDS solutions in the presence of sampling (*i.e.,* when flow information is obtained from sampled set of packets instead of full traffic) is an open question. In this study, we explore the impact of packet sampling on the performance and efficiency of ML-based NIDSs. Unlike previous work, our proposed evaluation procedure is immune to different settings of flow export stage. Hence, it can provide a robust evaluation of NIDS even in the presence of sampling. Through sampling experiments we established that malicious flows with shorter size (*i.e.,* number of packets) are likely to go unnoticed even with mild sampling rates such as 1/10 and 1/100. Next, using the proposed evaluation procedure we investigated the impact of various sampling techniques on NIDS detection rate and false alarm rate. Detection rate and false alarm rate is computed for three sampling rates (*i.e.,* 1/10, 1/100, 1/1000), for four different sampling techniques and for three (two tree-based, one deep learning based) classifiers. Experimental results show that systematic linear sampler - SketFlow performs better compared to non-linear samplers such as Sketch Guided and Fast Filtered sampling. We also found that random forest classifier with SketchFlow sampling was a better combination. The combination showed higher detection rate and lower false alarm rate across multiple sampling rates compared to other sampler-classifier combinations. Our results are consistent in multiple sampling rates, exceptional case is observed for Sketch Guided Sampling (SGS) as it caused a drastic performance drop when sampling rate was changed from 1/100 to 1/1000. Our results provide valuable insights for network practitioners and researchers regarding on how packet sampling effects ML-based NIDS performance. In this regard full source code for sampling and ML experiments has been released: github.com/Jumabek/sampledFlowMeter and github.com/Jumabek/nids-with-sampling

INDEX TERMS Flow information export, network traffic sampling, intrusion detection, machine learning, deep learning, CNN.

The associate editor coordinating the review of this manuscript and approving it for publication was Derek Abbott.

I. INTRODUCTION

Network monitoring applications such as flow analysis, intrusion detection, and performance monitoring have become

increasingly popular owing to the continuous increase in the speed and volume of network traffic [1]. Flow-based network monitoring is favored owing to its efficiency over full-packet-monitoring techniques such as deep packet inspection (DPI) [2]. Efficiency is achieved by inspecting the flow records [3] instead of individual packets. The network load owing to flow record collection and export (*e.g.,* NetFlow) is only 0.2% of the packet exporting technologies [2], [4]. However, with the increased volume and speed of internet traffic even flow record export has become a challenge for commodity devices (*i.e.,* switch, router). This is because processing each packet requires a certain bandwidth, memory, and CPU cycles of the measuring device. Therefore, packet sampling is used to reduce the overhead of the flow-information-measuring device [1], [5].

Flow-based Network Intrusion Detection Systems (NIDSs) use flow records as inputs and examine whether the specific flow is normal or malicious [2]. Recent research has proposed a large body of machine learning (ML) and deep learning (DL) solutions for flow-based NIDSs [6]–[13]. These solutions have demonstrated promising results in terms of their robust detection rates (DRs). However, to the best of our knowledge majority of the state-of-the-art solutions assume flow records are computed from full traffic, while in practice they are collected from sampled packets. As a result the success of state-of-the-art ML/DL based methods in practical scenarios is unknown. In this regard we investigate the impact of sampling on ML-based NIDS by considering real-world scenario (*i.e.,* when sampling is inevitable).

Track changes is on 3

Our proposed evaluation framework establishes a foundation for the research on the domain of ML-based NIDSs that consider practical scenario where sampling is inevitable. Proposed procedure for accurate assessment of ML model demonstrates the performance gains achieved by addressing data imbalance and emphasizes the importance of using the right aggregation metric for multi-class classification to avoid positively biased results. To achieve consistency in the ground truth that is not affected by the presence of sampling or configurations of flow export, we proposed the usage of flow level (*e.g.,* TCP connection level) evaluation. Next, by sampling packet capture (PCAP) traces from `CIC-IDS-2018` dataset we investigated the effect of various sampling techniques on the visibility of cyber attacks. Then, using the proposed evaluation framework, the effect of sampling on NIDS is investigated where ML classifiers are evaluated on sampled flow records instead of flow records obtained from full trace. Experimental results show that even moderate sampling rates such as 1/10, 1/100 and 1/1000 results in 20%, 50% and 80% loss of flow visibility respectively. Note, loss of flow visibility implies that NIDS has no chance of analyzing the malicious flow. Insights derived from this study paves the way for researchers in understanding and further studying the effect of sampling on ML-based NIDSs for ensuring the usability of ML-based NIDSs even in the presence of packet sampling. In this regard full source code for sampling and ML experiments has been released: github.com/Jumabek/sampledFlowMeter and github.com/Jumabek/nids-with-sampling. Our contributions in this paper, can be listed as follows:

- *Evaluation framework for flow-level ML-based NIDSs:* (1) in contrast to findings in previous literature [14], the time to train a convolutional neural network CNN) is reduced by 3× using larger batch sizes without sacrificing the performance, (2) a 42% higher detection rate (DR) is achieved by addressing the training-data imbalance, and (3) flow level evaluation framework is proposed that is reliable even when the number of extracted flow records varies owing to configurations of the flow metering & export stage.
- *Effect of sampling technique on malicious flow visibility:* From the perspective of network security, we observed that the linear systematic sampler - SketchFlow sampling (SFS) is most applicable when target malicious attack is formed from a longer flows, and the nonlinear sketch-guided sampling (SGS) is suitable for attacks with shorter flow length.
- *NIDS on sampled data:* The systematic linear SFS sampler provides a higher DR and lower false alarm rate (FAR) on multiple sampling rates. Hence, the most suitable solution for NIDS is an SFS sampler paired with random forest (RF) classifier.
- *Effect of constrained flow cache on the NIDS:* Experiments show that under-dimensioned flow cache (*i.e.,* when switch memory becomes under-dimensioned due to high traffic speed and volume) significantly increases the resource usage for the NIDS. Intuitively, this increase is caused by the increase in the number of extracted flow records. The DR decreased significantly when the flow cache is smaller than the working set of active flows.

In this paper, we are using a few acronyms and readers are referred to Table 1 for their definitions.

The paper is organized as follows. Section II briefly covers the body of knowledge on traffic sampling, network intrusion detection in the presence of traffic sampling and ML approaches for NIDS. Section III describes the deployment scenario of NIDS within flow monitoring architecture and its components. Section IV prepares a setting to conduct experiments in Section V which evaluates impact of sampling on ML-based NIDS. Finally, Section VI concludes the study with its limitations and future research directions.

II. RELATED WORK

In this section we review the related literature, make comparison with our study and provide a discussion on our novelty and contributions. Our study lies in the intersection of closely related two subdomains: (1) studies that investigated NIDS in the presence of sampling and (2) ML-based NIDS studies. In the following two subsections we compare and discuss novelty of our study in two aspects.

TABLE 1. List of abbreviations.

Domain	Abbreviation	Definition
Networking	PCAP	Packet Capture
Traffic Sampling	SRS	Simple Random Sampling
	SFS	SketchFlow Sampling
	SGS	Sketch Guided Sampling
	FFS	Fast Filtered Sampling
	SR	Sampling Rate
	SI	Sampling Interval
Intrusion Detection	NIDS	Network Intrusion Detection System
	DPI	Deep Packet Inspection
	AD	Anomaly Detection
	MD	Misuse Detection
Machine Learning	ML	Machine Learning
	DL	Deep Learning
	DR	Detection Rate
	FAR	False Alarm Rate
	RF	Random Forest
	DT	Decision Tree
	CNN	Convolutional Neural Networks

A. NIDS IN THE PRESENCE OF SAMPLING

Jun *et al.* [27] employed simple random sampling for detecting distributed denial-of-service (DDoS) attacks. They identified four key criteria for detecting DDoS attacks over the adopted packet-sampling algorithm. These criteria examines 1) if the number of sampled packets is larger than a threshold; 2) if the entropy of the flow-destination Internet protocols is larger than a threshold; 3) if the source port entropy of a flow is larger than a threshold; and 4) if the number of packets per second is larger than a given threshold. Expanding the guidelines to use sampling methods for intrusion detection, Ha *et al.* [28] proposed a sampling strategy to ensure the total amount of sampled traffic will not exceed the processing capacity of an intrusion detection system.

Considering the sampling rate and its effect on the performance of intrusion detection systems, Androulidakis *et al.* [29], investigated if hybrid sampling strategy improves NIDS performance. The authors achieved better detection accuracy by using a strategy which samples small flows with a constant sampling rate and dynamically decreases the sampling rate as the flow size increases.

Some studies [30], [31] have demonstrated that the sampling algorithm might introduce a bias towards the detection of volume and port-scan attacks, resulting in a dramatic decrease in performance. However, factors introduced by the sampling methods on the performance of the intrusion detection system can vary. Such factors can include the type of sampling technique and the underlying design of the intrusion detection system [4]. Therefore, our research focused on investigating the effect of different packet-sampling techniques on intrusion detection.

Notably, Jazi *et al.* [32] investigated the effects of various sampling approaches on the performance of application-layer DoS attack detection. Jazi *et al.* proposed a non-parametric cumulative sum (CUSUM) algorithm for intrusion detection and investigated the effect of sampling. Their results demonstrated that sampling, regardless of the sampling technique, degrades the accuracy of detecting DoS attacks compared with the accuracy achieved without sampling. Among the various sampling methods, selective flow sampling and SGS detected 84.61% of the attacks when a sampling rate of 20% was adopted. By decreasing the sampling rate to 1%, Fast Filtered Sampling (FFS) [22], and SGS achieved the best DRs of 30.76% and 23.07%, respectively. Another significant finding from this study was the effect of sampling rate on the sampler performance. For instance, when flow sampling rate was lower than 20%, SGS enabled a higher DR than the other samplers. In contrast, when flow sampling rate was higher than 20%, selective flow sampling enabled a highest DR.

Unlike previous methods that incorporate sampling for (non ML-based) intrusion detection, this study aimed at building ML-based intrusion detection system from flow records that are obtained from sampled packets. In practice sampled flow records are exported (*e.g.,* using NetFlow/IPFIX) from commodity switches. Focus of this study was packet sampling techniques that are deployed in the flow information export stage. Moreover, our ML experiments considered a practical scenario where available high-speed flow cache for measuring flow statistics is under-dimensioned [1].

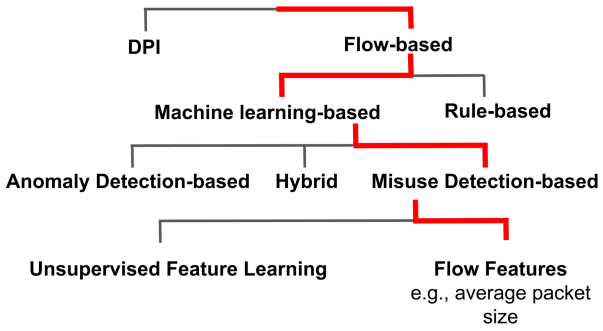

FIGURE 1. Scope of this study within NIDS literature.

B. FLOW-LEVEL ML-BASED NIDS

Domain of network intrusion detection has been extensively researched. Our scope within NIDSs is visualized in Fig. 1. Depending on the granularity of the inspected data, NIDS can be categorized into packet based and flow based groups. Deep Packet Inspection (DPI) is computationally expensive as it examines each packet in detail. Furthermore, its effectiveness is limited when the content of the packet is encrypted. Flow-based NIDS is favoured for its effectiveness and its ability to examine encrypted traffic. Our topic is to investigate effect of packet sampling in flow-based NIDS which employ ML classifiers. Throughout the paper we are referring to flow based NIDS whenever we mention NIDS. Another dimension to consider for the classification of NIDSs is by the approaches they use. Mainly, NIDSs employ machine learning-based or rule-based (*i.e.,*SNORT [57]) solutions. Flow-level NIDS approaches that employ machine learning can further be

categorized in two main groups: Anomaly Detection (AD) based and Misuse Detection (MD) based [33], [34]. AD approaches capture activities deviating from normal profile. Main advantage of AD approaches is the ability to detect unseen attacks. However, they usually suffer from high false alarm rate due to previously unseen but legitimate traffics. On the other hand, MD based approaches aim to identify previously known attacks based on their signature and patterns. They are effective for detecting the known types of attacks but fail to recognize previously unseen attacks.

Mirksky *et al.* [35] developed Kitsune - AD based NIDS that is comprised of ensemble of autoencoders. Kitsune can efficiently detect attacks in an online manner while having comparable performance to offline anomaly detectors. Bovenzi *et al.* [34] proposed a lightweight anomaly detection via designing novel multi-modal deep autoencoder. Proposed autoencoder treats each categorical features as a separate modality and achieves good performance while maintaining a lightweight design for IoT scenarios.

This study focuses on MD based NIDS that operates using flow information (records) with the emphasis on the effects of traffic sampling. Although we are the first to investigate the effects of sampling on flow-level ML-based NIDS, in the following we make the comparison of our evaluation procedure with recent works on ML-based NIDS that uses misuse detection approach. Our criteria for the selection was the studies with high number of citation or studies that experimented with the same dataset (*i.e.,* `CIC-IDS-2018`). Overall comparison is shown in Table 2. In the following we briefly review each study.

In order to obtain better feature representation prior research exploited deep auto-encoders for building latent feature representations [10], [11]. Their NIDS pipeline first learns feature representation in a unsupervised manner from unlabeled data. Then on labeled data supervised learning based classifiers such as Random Forest and Softmax employed on learned representations. Another deep learning based approach that learns latent representations for flow is by Yin *et al.* where he proposed Recurrent Neural Networks (RNNs) by considering both binary and multiclass settings. Authors also investigated impacts of the number neurons in RNN and learning rate on model performance. Their results outperformed other ML based classifiers such as J48, ANN, RF and SVM on NSL-KDD benchmark.

One limitation of these studies so far is the usage of outdated traffic (*i.e.,* KDD or NSL-KDD) which is far from modern network traffic patterns. To tackle this issue, Sharafaldin *et al.* [12] collected `CIC-IDS-2017` dataset and extracted biflow features with their CICFLOWMETER [39] tool. In this dataset they evaluated candidate NIDS classifiers such as K-Nearest Neighbors (KNN), Random Forest (RF), ID3 implementation of Decision Tree (DT), Adaboost, Multilayer perceptron (MLP), Naive-Bayes and Quadratic Discrimant Analysis (QDA). Later, same research group released `CIC-IDS-2018` dataset which contains full packet capture (PCAP) trace recorded on each machine. Ferrag *et al.* [13] exploited this recent dataset as a benchmark and compared the performance of seven deep learning algorithms in a binary and multiclass classification settings. Alharbi *et al.* [37] proposed a novel approach for detecting botnet attacks. Their experiments were also evaluated using modern real-world dataset [38]. Authors proposed Local-Global best Bat Algorithm that is used for hyper parameter tuning and weight optimization of Neural Network. Their target task was to classify the traffic as one of the ten different Botnet attacks or as benign.

Next critical issue in the previous studies is the lack of reliable evaluation procedure. Specifically, not addressing the data imbalance in the training phase and misleading performance aggregation in the evaluation. In other words, as shown in Table 2, to the best of knowledge none of the previous studies address data imbalance and used macro averaging in multiclass classification. We experimentally show that addressing data imbalance results in significant performance gains. We also explain how using micro or weighted averaging results in positively biased performance estimation (more details in Section IV) and propose using macro averaging for computing overall performance across multiple categories.

Another crucial aspect of previous evaluation procedure is regarding the granularity in the definition of a flow. Previous research uses the flow records as a unit of evaluation. However, generated number of flow records are inconsistent when any of the components in flow metering stage changes (more details in Section III).This includes the presence of sampling. To solve this issue we first clarify the difference between flow/connection and flow record in Section III.C. We also propose flow/connection level evaluation procedure in Section IV.E2 that is consistent irrespective of the parameters of flow metering stage.

Additionally, we would like to mention that Machine Learning and Deep Learning is widely deployed in domains such as autonomous driving [58]–[61]. As ML/DL solutions are increasing so does the adversarial attacks on such models are being developed. In this regard, Lal *et al.* developed a technique for protecting deep learning based medical models such as diabetic retinopathy recognition from adversarial attacks [62].

III. FROM SAMPLED PACKETS TO NIDS INFERENCE

Flow monitoring stages such as packet observation, flow export, data collection, and data analysis are closely intertwined [1]. This section explains the deployment scenario for NIDS, the effects of traffic sampling on NIDS and the function of sampling within flow monitoring system.

Thus, we describe components and processes in detail. We explain the stages from a packet arrival on the forwarding device (*e.g.,* switch), process for constructing flow record and final NIDS inference on the flow record. A schematic view of the components involved in the NIDS is depicted in Fig. 2. We composed the system by following previous literature [1], [5] where flow monitoring involves three stages: 1) flow metering and exporting, 2) collection, and 3) analysis. The

TABLE 2. Evaluation framework of previous studies are compared. Criteria for inclusion is (1) studies with high citation and/or studies which used same dataset (*i.e.,* CIC-IDS-2018) with our work. We improve previous ML-based NIDS by (1) showing better ML performance through data imbalance handling, (2) employ fair overall metric for comparison by using Macro averaging and (3) most importantly provide evaluation framework for NIDS in the presence of sampling by introducing flow level metric.

No. of citations (as of 04/2021)	Study	Method	Dataset	Recent	Handles Data Imbalance	Aggregation for overall performance	Evaluated unit of data
582	Javaid *et al.* [10]	Self-taught Learning	NSL-KDD-2009			NA	Flow Record
660	Yin *et al.* [36]	RNN	NSL-KDD-2009			NA	Flow Record
487	Shone *et al.* [11]	Deep Auto-Encoder	KDD Cup '99			Micro averaging	Flow Record
804	Sharafaldin *et al.* [12]	Comparison of ML methods	CIC-IDS-2017	✓		Weighted averaging	Flow Record
8	Ferrag *et al.* [13]	Comparison DL methods	CIC-IDS-2018	✓		NA	Flow Record
	Ours	Comparison of CNN, RF, DT	CIC-IDS-2018	✓	✓	**Macro averaging**	**Flow**

flow metering and exporting process is performed on a metering device (*e.g.,* switch). Flow information (*e.g.,* number of packets, average inter-arrival time) is accumulated in the flow cache and exported to the collector. Collector stores the received flow records for further usage. Additionally, it sends flow records to the monitoring applications such as NIDS for real-time analysis. In the following subsections, we provide a detailed description of each stage.

A. FLOW METERING & EXPORT

Flow metering is a process in which flow statistics such as byte volume, packet volume, and inter-arrival time are measured [5]. In a high-speed router, processing each packet requires a certain bandwidth, memory, and CPU cycles of the measuring device. For this reason packet sampling is used to reduce the overhead of flow metering devices. This reduction enables commodity devices to process sampled packets at a line rate [17]. Flow export is the process of exporting accumulated flow information (*i.e.,* flow record) to collector.

1) FLOW METERING

In our experiments, flow metering was performed using CICFlowMeter [39], a flow feature extractor tool. It collects 78 bidirectional flow features as shown in Table 4 by processing the packet capture (PCAP) traces of `CIC-IDS-2018`. The direction of the flow is identified using a 5-tuple (src-IP, dst-IP, protocol, src-Port, dst-Port). Owing to the flow export of the flow cache table, a single flow can be separated into multiple flow records in the collector. Moreover, two distinct flows may have the same flow ID (5-tuple) only separated by time. Therefore, in our experiments, we additionally considered the start/end time of a flow as the flow ID to correctly identify flows.

2) SAMPLING

Packet sampling is primarily used for reducing the workload in flow metering stage. As a result only sampled packets undergo flow metering process. In our experiments sampling module is integrated inside CICFlowMeter. Each (incoming) packet is examined by sampling module where only sampled packets are further considered for flow metering. For our experiment purposes we implemented four different samplers: Simple Random Sampling (SRS), SFS, FFS and SGS.

3) FLOW CACHE AND FLOW EXPORT

Flow cache is a table which allocates an entry for each active flow. Flow cache usually resides in the high speed TCAM memory, in order to maintain millions of lookups per second during the flow metering process. However, this type of memory cannot be large because of the high price tag and large power consumption. Thus, flow export is crucial for the continuous operation of the cache table. Flow export stage evicts the records of the expired flows and sends them to collector. Additionally, if flow cache utilization reaches to specified threshold, some portion of flow records are prematurely exported.

In NetFlow [40], flow is considered expired based on the following rules. First, a flow record is naturally exported when the *FIN* or *RST* flag packet is observed. Second, the flow record is exported if it is inactive for a certain time (*i.e.,idle timeout*) to save memory. Third, the *hard timeout* of a flow also triggers an export event for the timely analysis of elephant flows. Our implementation followed these rules.

It is noteworthy to mention that the presence of sampling has impact on the flow expiration. For instance, natural flow expiration with *FIN* or *RST* flags becomes unreliable if those flag packets are not sampled. Similarly, idle timeout-based expiration becomes inaccurate since sampling does not guarantee the two consecutive packets will be sampled. In our experiments, exported flow records were saved into CSV files and subsequently used to build and evaluate ML-based NIDS.

B. FLOW COLLECTION AND NETWORK INTRUSION DETECTION SYSTEM

Here we describe the procedure of collecting exported flow records and their analysis by NIDS.

1) FLOW COLLECTION

When flow exporting process expires a flow, its record is sent to collector. Collector then in turn stores it for logging purposes and passes it to real-time monitoring tools such as NIDS. It is important to distinguish the difference between the flow collector and flow feature extractor in the

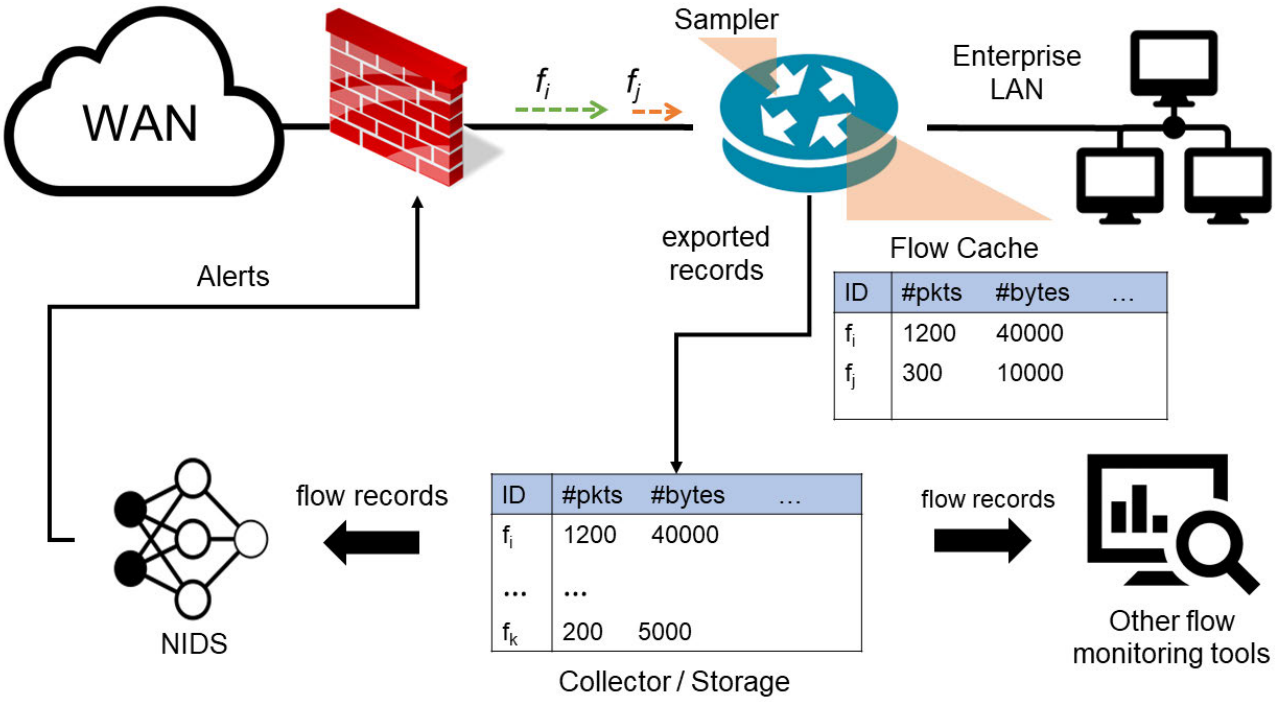

FIGURE 2. Schematic view of flow monitoring architecture which is located between WAN and enterprise LAN. While packets are passing through, flow information is accumulated in the switch from a sampled set of packets. Once flow finishes or the timer expires, an accumulated record of the flow is exported to Collector. Collectors can store the records for the future and/or pass them to real-time monitoring systems such as NIDS. NIDS analyzes the flow record and informs the firewall or network administrator if a record for that particular flow is found malicious. The key question here is how the type and rate of the sampling technique affect malicious flow detection performance.

metering stage. The flow collector is similar to an archiving database that stores flow records while the feature extractor (*i.e.,* CICFlowMeter in our setting) collects features for the individual flow from incoming packets in the flow metering stage. Readers might want to refer to [1] for more detailed information on flow collectors.

2) NETWORK-INTRUSION-DETECTION SYSTEM

The NIDS receives a flow record from the collector and examines whether the corresponding flow is malicious. The scope of our study was a flow-level ML-based NIDS, which consists of classifying each flow (*i.e.,* record) as specific malicious category or benign. Although some NIDSs can inspect deterministic rules (*i.e.,* signature-based) such as the number of flows/packets passing through the observation point) for port-scan attacks, they are out of the scope of this paper.

In our experiment, the ML-based NIDS was trained and evaluated on flow records that were extracted with CICFlowMeter. For sampling experiments, flow records were collected from sampled packets. Thus, the presence and type of sampling has a direct and crucial effect on the overall performance of the NIDS.

C. FLOW AND FLOW RECORD

Understanding the difference between flow (*i.e.,* connection/-conversation object) and flow records (*i.e.,* traffic object) is crucial for accurate evaluation of flow based NIDS. Unfortunately, this difference is often overlooked, perhaps not well understood. Understanding the underlying issues and proposing a solution allows us to accurately assess the performance of ML-based NIDS even in the presence of sampling.

In networking community a flow has multiple definitions. RFC 2722 [41] defines traffic flow as ''an artificial logical equivalent to a call or connection.'' According to RFC 3679, A flow is a sequence of packets sent from a particular source to a particular unicast, anycast, or multicast destination that the source desires to label as a flow. A flow could consist of all packets in a specific transport connection or a media stream. However, a flow is not necessarily 1:1 mapped to a transport connection. Flow is also defined in RFC 3917 [42] as ''a set of IP packets passing an observation point in the network during a certain time interval.''

1) DEFINITIONS OF FLOW AND FLOW RECORD IN THE LITERATURE

As a result based on the above definition it is not clear if five-tuple (*i.e.,* source IP, destination IP, source port, destination port, transport protocol) is sufficient to identify a flow. It is also not clear if the records extracted from flow export tools (*e.g.,* NetFlow) are 1:1 mapped to flow. In a different but closely related domain of traffic classification Dainotti *et al.* [43] highlighted the same issue by explaining the different granularities of traffic flows. The granularity

of traffic flows reflect the portion of the packet headers analyzed to construct flow objects (*i.e.,* records). As shown by Dainotti *et al.* [43], single TCP connection may consists of multiple flow/biflows. Authors define these flow objects as follows:

- *TCP connections:* Heuristics based on the observation of some TCP flags or TCP state machines are used to identify the start and the end of each connection.
- *Flows:* A typical flow definition uses the 5-tuple which consists of source IP, source port, destination IP, destination port and transport-level protocol. Some tools also use a flow timeout (60 s or 90 s of idle time to delineate the end of a flow) or periodic reset (*e.g.,* timeout all flows on a 5-min boundary).
- *Bidirectional flows (biflows):* Same as above, but includes both directions of traffic, assuming both directions of flows can be observed.

2) REDEFINING FLOW AND FLOW RECORD ON CIC-IDS-2018

Following Dainotti *et al.* [43], we differentiate the connection and flow object (*i.e.,* flow record). However, instead of flow/biflow object we use the NetFlow's terminology of flow record (objects). We also expanded the ''TCP connection'' [43] flow object type to ''flow'' which includes both TCP and UDP conversation. CICFlowMeter extracts bidirectional flow features (*i.e.,* biflow records). Thus, unless otherwise mentioned throughout the paper, we are using the record and flow record terms to refer to bidirectional flow records.

Generally, a flow (connection level flow ID) is defined with 5-tuples. However, since `CIC-IDS-2018` PCAP trace is captured for the whole day it is possible to have more than one connection for the same 5-tuples. For instance, earlier during the day a TCP connection *A* starts and lasts ten minutes, then later during the day a new connection *B* with same 5-tuple starts and lasts some time. In such cases it is difficult to distinguish two different connection/flow with 5-tuple. Therefore, after thorough explorative data analysis, we verified that a 5-tuple on a given day's PCAP trace is unique. During this verification, for TCP, our heuristic was TCP flags such as SYN, FIN, RST where single connection should have single handshake. For UDP, we considered all packets from 5-tuples as a single connection as long as there is no large time interval between the two consecutive bursts. During flow export stage, flow can be split into multiple sub-records due to active/idle timeout or premature eviction that is caused by under-dimensioned flow cache. In other words, for a single connection multiple flow records could be exported. Therefore, it is essential to distinguish the difference between flow and flow record. In this regard we define the flow and flow record as follows:

- `Flow:` A transport layer connection (TCP or UDP) that is uniquely identified with 5-tuples with heuristics such as TCP flags and sufficiently large gap between two bursts for UDP. Our definition of flow maps 1:1 to transport layer connection.
- `Flow Record:` a biflow object that consists of flow properties for the given time interval. Properties such as average packet size, total number of bytes is accumulated with CICFlowMeter and exported in the form of a record when one of the following condition occurs: (1) active flow timeout - when flow was active for a long time period and its partial record that is accumulated so far should be exported for timely analysis; 2) idle timeout - when flow was inactive for certain period and it should be exported for timely analysis; and 3) eviction - when flow cache is under-dimensioned and last recently updated flow records are kicked out.

IV. EXPERIMENTAL SETTING

In this section, we describe our experimental dataset, settings used for flow cache and configuration for flow expiration in the exporting stage. We explain the parameters of the samplers and their settings for obtaining desired sampling rate. Hyper-parameters and architecture of machine learning classifiers also explained. We also introduce and justify our metrics for assessing the classifier performance and aggregation technique for obtaining overall representative score. Finally, we explain our flow level evaluation procedure to evaluate NIDS in the presence of sampling.

A. DATASET

Most commonly used dataset for evaluating NIDS performance is `KDD` or its updated variant `NSL-KDD` [44]. However, these dataset contain outdated attack scenarios with small scale. To address this issue recently new datasets `CIC-IDS-2017`, `AWS CIC-IDS-2018`, and `DDoS 2019` were released [12]. Among them `AWS CIC-IDS-2018` contains the complete PCAP traces of both benign and malicious network traffic. It also includes schedules for malicious attacks. Hence, we adopted the `AWS CIC-IDS-2018` dataset for our experiments.

1) AWS CIC-IDS-2018

The dataset was constructed to convey the network traffic of an organization that operates on 30 servers and 420 machines. Malicious attacks were performed by an attacker network that consisted of 50 machines. Normal user behavior (traffic) was simulated using ML techniques such as clustering. Thirteen types of malicious attacks were captured within 10 days. During the period incoming and outgoing traffic was recorded in each host machine. Consequently, a total of $450 \times 10 = 4500$ PCAP files were collected. In order to study the effect of sampling, we need the PCAP traces to be recorded from forwarding devices such as switch and router. Therefore, we merge each 450 machine's traces into single trace to obtain equivalent of PCAP trace that is recorded in forwarding device. As a result, we obtained ten merged PCAP traces for each day.

TABLE 3. Per category distribution of network flows. Brute Force-Web* attacks have equally short and long flows. In this context, short attack is an attack where its flows have mostly small number of packets based on the 3.

Short Attacks	# Flows	Long Attacks	# Flows
DDoS-HOIC	163,750	Brute Force Web	260
DDoS LOIC-HTTP	163,439	Brute Force XSS	117
DoS-Golden Eye	14,116	SSH-Brute Force	14,146
DoS-Slow HTTP Test	14,116	DoS Hulk	14,116
FTP-BruteForce	14,116	DDoS-LOIC-UDP	1763
DoS-Slow loris	7,247	Infiltration	5
SQL Injection	50		
Benign	40,568,049		
Total Flows	40,975,291		

a: FLOW SIZE DISTRIBUTION

Flow size is the number of packets that belong to specified flow. As explained in Section IV.C, probability of the packet being sampled may differ depending on the (accumulated-online) size of its flow. Therefore, the effect of sampling on NIDS affected by the underlying flow size distribution of the traffic and type of deployed sampling technique. In this regard flow size distribution is shown in Fig. 3 where each category is color coded. Graph conveys that size of the flows for different cyber attacks vary where some flows are short size while some are long. Interestingly, DDoS-LOIC-UDP has both long and short flows.

b: CLASS DISTRIBUTION

As it is common in network-intrusion-detection datasets, `CIC-IDS-2018` is severely imbalanced (Table 3). As Table 3 shows, attacks such as *i.e.,* Brute-Force-Web, Brute-Force-XSS, Infiltration and Sql-Injection have very small numbers of flows compared with the DoS and DDoS attacks (*e.g.,* DoS Hulk with flow counts of 14,116).

2) DISTRIBUTION OF FLOW FEATURES

For extracting flow features we use CICFlowMeter [39] which collects and extracts 78 features for the given flow. Complete list of features with their distribution (*i.e.,* mean and standard deviation) is shown in Table 4. For more information on the computation procedure and other details we refer the readers to CICFlowMeter [39].

3) FLOW LABELING PROCESS

Since we are building misuse detection based NIDS our task is a classification task. Classification is a supervised learning problem which requires target (*i.e.,* label) variable. Here we discuss the procedure of obtaining target variable for our dataset of choice. Note that our target parameter is the label of each records.

`CIC-IDS-2018` provides necessary metadata comprised of flow IDs of malicious attack and their schedules. Such information is sufficient to label each flow record with its corresponding category (*e.g.,* DDoS-HOIC). However, after exploratory data analysis we found that in some instances same Flow ID was used to execute more than one type of cyber attacks.

For instance, assume that flow with ID 18.218.115.60-172.31.69.28-53373-80-6 is first used to conduct the Brute-Force-Web attack at 10:23:15 AM and Brute-Force-XSS attack at 02:10:36 PM. As a result, without the attack schedules, it is not possible to distinguish two different labels. Therefore, for labeling a flow record, we considered attack schedules in addition to the flow IDs.

Inaccurate attack schedules introduce noise to the data. Consequently, derived analysis from noisy data would become unreliable. Therefore, we carefully examined the PCAP traces of `CIC-IDS-2018` using Wireshark to see if attack schedules corresponds to the timeline of malicious-flow IDs. We found that slight inconsistency where certain malicious flows lasted longer than provided schedules [45]. Thus, we provided our own updated attack schedules of `CIC-IDS-2018` in Table 5. Our goal here is to contribute for the reliability of the future experiments on `CIC-IDS-2018` dataset. It is noteworthy to mention that PCAP trace on Friday-02-03-2018 that is responsible for Botnet attack was excluded from analysis for having too many mismatch between the malicious flow ID and provided attack schedules.

B. FLOW EXPIRATION AND FLOW CACHE SETTINGS

Flow features for active flows are accumulated as packets in the flow cache until the conditions of flow expiration are satisfied (Section III.A). Active timeout values range from 120 s to 30 min, and idle timeout values range from 15 s to 5 min [1]. In line with CICFlowMeter, for the timeout values we used active timeout threshold of 120 s and the idle timeout threshold of 15 s. Original implementation of CICFlowMeter assumes enough flow cache is available so that no record is kicked out prematurely. We follow the same pattern and assume flow cache is unlimited. However, for under-dimensioned flow cache experiment in Section V.D fixed size is assigned to the flow cache. The size is computed with respect to the ideal memory size. Ideal memory for each PCAP trace is a flow cache memory that is enough to encompass Working Set of Active Flows [46]. Ideal flow cache memory for each of nine PCAP traces computed separately. When flow cache becomes under-dimensioned, CICFlowMeter kicks out (*i.e.,* prematurely exports) the record from least recently updated entry.

C. SAMPLERS

Here, describe and compare the samplers adopted in our experiments. Additionally, their parameter settings for obtaining the desired sampling rates is also explained. We used 1 Mb memory for each sampler, as suggested in the original studies [18], [47]. Comparison of samplers in multiple dimensions is shown in Table 6. In terms of complexity SRS is preferable as it does not need any extra memory resources for per-flow counting. Additionally, SRS

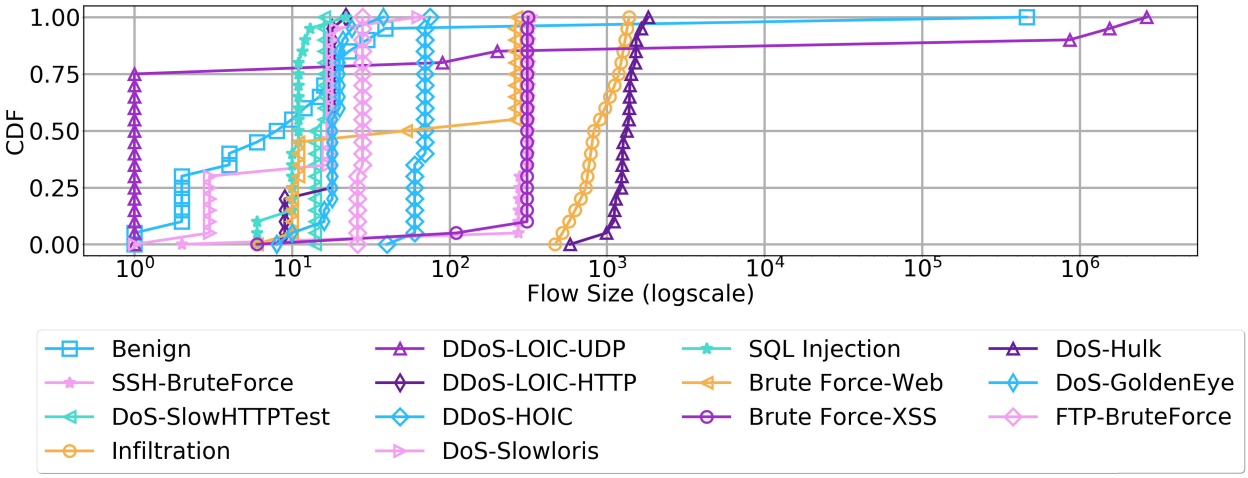

FIGURE 3. Flow size distribution of `CIC-IDS-2018` per category whereas each category is distinguished by the color and marker. Flow size is the number of packets within the flow. 95% of benign traffic flows have length less than 100. Majority of DoS-Hulk and Infiltration attacks are of length more than 1000 which implies high flow visibility even with sampling rates of 1/1000.

guarantees constant sampling rate for any flow distribution and does not require manual parameter tuning. In contrast, other three samplers FFS, SFS and SGS require user to initialize some parameters before sampling. Assuming traffic distribution is known, this involves experimentally trying multiple values for each of the parameters. This procedure is repeated until desired sampling rate for the given distribution is obtained. For instance, in the case of SGS, we tried multiple values for error bound ϵ parameter until we achieved desired sampling rate. In the following we describe of each individual sampler and their selected parameter values that provided desired sampling rate. Note, if flow size distribution of the traffic changes, then overall sampling rate also changes. This implies that in practice, it is difficult to guarantee constant sampling rate across traffic for samplers other than SRS.

1) SIMPLE RANDOM SAMPLING (SRS)

SRS [15] is the most representative linear sampler that guarantees a constant sampling rate for any flow distribution. SRS requires almost no memory as it needs a single counter that counts between two sampled packets. SRS is included in our experiments as a baseline sampler. The SRS strategy involves counting from 0 until n. When counter reaches to n, the n-th packet is sampled and counter is reset. After each sampling, n is set to a random number. In our settings, we used sampling rates of 1/10, 1/100, and 1/1000.

2) SketchFlow SAMPLING (SFS)

SFS is included as a candidate per-flow sampler that provides approximately stable sampling rate for each flow. SFS achieves this by approximately per-flow counting using the novel compact data structure (*i.e.,* sketch) [18]. Owing to per-flow counting, SFS outperformed SRS in terms of accuracy [18]. However, due to the nature of sketching (approximate counting), SFS cannot obtain a precise sampling rate as SRS. In our settings, we varied the number of layers and number of saturating bit parameters until desired sampling rate approximated. Specifically, sampling rates of 1/10 is obtained using one-layer sketch with five non-saturating bits), 1/100 is obtained using two-layer sketch with five non-saturating bits), and 1/1000 is obtained using four-layer sketch with four non-saturating bits).

3) SKETCH-GUIDED SAMPLING (SGS)

SGS is a nonlinear sampler that reduces the packet-sampling rate as the online size of a particular flow increases. The rationale behind the SGS design is that for elephant flows, even small sampling rates sample sufficient number of packets to estimate flow information. Thus SGS strategy assigns low sampling rate for large flows and large sampling rate for small flows. As a nonlinear sampler, SGS cannot guarantee constant sampling rate across distributions. Hence, we manually attempted multiple values for the error bound parameter to obtain desired sampling rates. We observed that $\epsilon = 0.05$, $\epsilon = 1$, and $\epsilon = 11.5$ results in sampling rates of 1/10, 1/100, and 1/1000, respectively. It is noteworthy to mention that this parameters apply only for `CIC-IDS-2018` traffic distribution.

4) FAST FILTERED SAMPLING (FFS)

FFS assumes that small flows are the sources of anomalous traffic. Hence, a filtering mechanism is introduced to thin large flows in which packets are sent to the sampling module only if they pass through the filtering module. The filtering module counts flow sizes online and has two threshold values, s and l, where $s < l$. If the flow size of the arriving packet is smaller than s, the filtering module simply passes the arriving packet to the sampling module. This allows higher sampling probability for flows smaller than size s. If the flow size is larger than s, the next arriving packets from the s-th to the

TABLE 4. List of features extracted by CICFlowMeter for `CIC-IDS-2018`. For each feature, mean and standard deviation is shown. Time based features such as IAT, active/idle statistics are shown in seconds.

index	feature	mean	standard deviation	index	feature	mean	standard deviation
1	Dst Port	NA	NA	40	Pkt Len Min	11.61	33.75
2	Protocol	NA	NA	41	Pkt Len Max	323.05	488.34
3	Flow Duration	4.39	18.82	42	Pkt Len Mean	69.65	103.4
4	Tot Fwd Pkts	6.75	648.46	43	Pkt Len Std	99.68	153.38
5	Tot Bwd Pkts	6.53	161.36	44	Pkt Len Var	33460.92	177900.94
6	TotLen Fwd Pkts	381.84	20892.43	45	FIN Flag Cnt	0.34	0.48
7	TotLen Bwd Pkts	4136.83	229662.87	46	SYN Flag Cnt	0.68	1.27
8	Fwd Pkt Len Max	137.4	252.16	47	RST Flag Cnt	0.22	0.45
9	Fwd Pkt Len Min	1.73	9.43	48	PSH Flag Cnt	2.63	11.83
10	Fwd Pkt Len Mean	34.49	57.84	49	ACK Flag Cnt	9.05	205.02
11	Fwd Pkt Len Std	50.59	89.49	50	URG Flag Cnt	0	0.01
12	Bwd Pkt Len Max	315.1	484.69	51	CWE Flag Count	0.18	0.42
13	Bwd Pkt Len Min	15	37.38	52	ECE Flag Cnt	0.24	0.58
14	Bwd Pkt Len Mean	90.44	149.92	53	Down/Up Ratio	0.73	1.75
15	Bwd Pkt Len Std	118.69	187.7	54	Pkt Size Avg	82.33	117.01
16	Flow Byts/s	260610.32	4010679.43	55	Fwd Seg Size Avg	34.49	57.84
17	Flow Pkts/s	30501.69	215424.79	56	Bwd Seg Size Avg	90.44	149.92
18	Flow IAT Mean	0.22	0.76	57	Fwd Byts/b Avg	0	0
19	Flow IAT Std	0.33	0.95	58	Fwd Pkts/b Avg	0	0
20	Flow IAT Max	0.93	2.41	59	Fwd Blk Rate Avg	0	0
21	Flow IAT Min	0.04	0.44	60	Bwd Byts/b Avg	0	0
22	Fwd IAT Tot	4.05	18.67	61	Bwd Pkts/b Avg	6.3	649.54
23	Fwd IAT Mean	0.34	1.32	62	Bwd Blk Rate Avg	40607.99	7266310.65
24	Fwd IAT Std	0.28	0.95	63	Subflow Fwd Pkts	0	0
25	Fwd IAT Max	0.73	2.24	64	Subflow Fwd Byts	14.86	25.72
26	Fwd IAT Min	0.05	0.58	65	Subflow Bwd Pkts	0.44	0.5
27	Bwd IAT Tot	4.14	18.28	66	Subflow Bwd Byts	59.07	96.13
28	Bwd IAT Mean	0.34	1.19	67	Init Fwd Win Byts	-1	0
29	Bwd IAT Std	0.35	1.04	68	Init Bwd Win Byts	11095.03	22769.41
30	Bwd IAT Max	0.84	2.35	69	Fwd Act Data Pkts	4.81	646.36
31	Bwd IAT Min	0.02	0.37	70	Fwd Seg Size Min	0	0
32	Fwd PSH Flags	0	0	71	Active Mean	0.28	1.37
33	Bwd PSH Flags	0.1	0.3	72	Active Std	0.03	0.45
34	Fwd URG Flags	0	0	73	Active Max	0.33	1.63
35	Bwd URG Flags	0	0	74	Active Min	0.26	1.32
36	Fwd Header Len	98.49	5300.35	75	Idle Mean	1264920562	401278422.2
37	Bwd Header Len	136.27	3227.22	76	Idle Std	301398939	470089028.9
38	Fwd Pkts/s	2168.21	15482.57	77	Idle Max	1519233628	425716.26
39	Bwd Pkts/s	28333.65	214027.2	78	Idle Min	1059663895	697857883

l-th values are dropped by the filtering module. When the online counter of a flow reaches l, it is reset to zero to pass the next s packets to the sampling module.

Although, other values for filtering thresholds could potentially result in better flow visibility, we fixed threshold as $s = 8$ and $l = 16$ by following the original work [22]. Otherwise, it would result in large number of trials while grid searching the best possible combination of filters. After fixing the filters, we set the sampling intervals for the SRS sampler to $si = 4$, $si = 40$, and $si = 400$, which resulted in 1/10, 1/100, and 1/1000 sampling rates, respectively.

D. CLASSIFIERS

We selected three machine learning classifiers, namely, decision tree (DT), Random Forest (RF), and CNN, as these are widely studied in NIDS because of their reliable DRs [12], [13], [48], [49]. Unless otherwise mentioned we used default hyper-parameters for all three methods.

For DT, we used scikit-learn [50] implementation with default hyper-parameter settings. According to default setting each test split considers random $\sqrt{d}$ features where $d = 78$ is the dimension of the flow-record features. Similarly, for random forest (RF) we used scikit-learn [50] implementation with default settings for hyper-parameters. Default setting

TABLE 5. Updated attack schedules for CIC-IDS-2018.

Date	Attack Name	Start Time	Finish Time
Wed-14-02-2018	FTP-BruteForce	10:32:00 AM	12:10:31 PM
	SSH-Bruteforce	02:01:00 PM	03:32:30 PM
Thu-15-02-2018	DoS-GoldenEye	09:27:42 AM	10:11:45 AM
	DoS-Slowloris	11:00:12 AM	11:41:34 AM
Fri-16-02-2018	DoS-SlowHTTPTest	10:12:00 AM	11:08:59 AM
	DoS-Hulk	01:45:00 PM	02:19:59 PM
Tues-20-02-2018	DDoS-LOIC-HTTP	10:12:00 AM	11:17:59 AM
	DDOS-LOIC-UDP	01:13:00 PM	01:32:59 PM
Wed-21-02-2018	DDOS-LOIC-UDP	10:08:50 AM	10:43:59 AM
	DDOS-HOIC	02:05:00 PM	03:05:59 PM
Wed-28-02-2018	Infiltration	10:50:00 AM	12:08:55 PM
	Infiltration	01:42:00 PM	02:40:59 PM
Thu-22-02-2018	Brute Force-Web	10:17:00 AM	11:24:59 AM
	Brute Force-XSS	01:50:00 PM	02:29:59 PM
	SQL Injection	04:15:00 PM	04:29:59 PM
Fri-23-02-2018	Brute Force-Web	10:03:00 AM	11:03:59 AM
	Brute Force-XSS	01:00:00 PM	02:10:59 PM
	SQL Injection	03:05:00 PM	03:18:59 PM
Thu-01-03-2018	Infiltration	09:57:00 AM	10:55:59 AM
	Infiltration	02:00:00 PM	03:37:59 PM

TABLE 6. Comparison of sampling techniques.

Sampling technique	Flow size preference	Memory usage	User defined parameters		
			Constant sampling rate	Parameter name	Requires parameter tuning
Simple Random Sampling	No preference	No	Yes	- sampling rate	- No
Fast Filtered Sampling	Small	Large	No	- small flow threshold - large flow threshold - sampling rate of sampling module	- Yes - Yes - Yes
SketFlow Sampling	Large	Small	No	- number of sketch layers - number of non-saturating bits	- Yes - Yes
Sketch Guided Sampling	Small	Large	No	- error bound	- Yes

uses $n = 100$ tree estimators with bootstrapping. Hence, each tree was constructed using 1% of the training data.

For the CNN model, we exploited the architecture proposed for NIDS by Kumar [6]. Performance of DL model can be improved through exhaustive hyper-parameter tuning or searching for the optimal CNN architecture. Designing another CNN architecture involves searching for the optimal number of layers to use, kernel size of convolution and pooling layers, usage of dropout, batch normalization, and number of units in dense layers. Such process exhausts time and computing resources. Additionally, it increases the carbon footprint of this research [51]. Hence, we saved resources and reduced carbon footprint by exploiting previously well establish CNN architecture [6]. The CNN architecture consists of two 1D convolution, one pooling, and two dense layers. The convolution layers consisted of 64 filters with 1D kernel of size 3. The max pooling layer used a kernel of size 2. The first and second dense layers had 2496×256 and 256×14 neuron units. We adjusted last layer to have 256×14 units because our dataset has 14 different categories (13 malicious and one benign). Vinaya *et al.* used default batch size provided in Keras library [52] that is 32. However, as shown in the Section V.B we found that larger batch sizes result in faster training time without degrading model performance. After experimentally evaluating multiple batch sizes we found that the batch size of 4096 provided fastest training speed while having small FAR with comparable DR. For the rest of the hyper parameters, we use default settings such as adam optimizer, learning rate of 0.003, and weight decay of 0.

E. PERFORMANCE METRICS FOR FLOW-LEVEL ML-BASED NIDS

In the following, we first explain the metrics used to evaluate sampler in the context of network intrusion detection. Then we explain and justify our evaluation setup for NIDS in the presence of sampling.

1) SAMPLER EVALUATION

Sampling affects NIDS in two main ways: (1) loss of flow visibility and (2) loss of accuracy in estimated flow records. For our definition of flow please refer to Section III.C.

a: FLOW VISIBILITY

We introduce a flow visibility metric to measure the percentage of flows that are remained visible after sampling. If no packet from the flow is sampled, then no flow record is exported. As a result, flow is not analyzed by the NIDS. Flow visibility is a binary metric where it is either observed or not. Visibility is 100% when at least one packet is sampled which ensures a flow record is generated for a given flow. It is 0% if no packet is sampled from that flow and consequently no flow record is exported.

b: FLOW-RECORD QUALITY

When estimating the flow information from sampled packets, quality of a record is degraded due to loss of data. This in turn leads to decreased ML classifier performance when learning to distinguish between the different types of attacks or benign traffic. Therefore, if certain sampler provides relatively higher quality flow record, then it contributes to better NIDS performance. In other words, if same NIDS classifier was build on two different flow records that correspond to two different sampling techniques, then the sampler that resulted better NIDS performance implies that sampler is more suitable for NIDS purpose than its counterpart. Hence, we do not have exclusive metric for flow-record quality evaluation, but infer it from NIDS performance.

2) NIDS EVALUATION IN THE PRESENCE OF SAMPLING

a: CROSS VALIDATION

In ML community, Cross Validation (CV) is used for reducing the bias introduced when splitting small dataset. Although `CIC-IDS-2018` is a large dataset, we still need CV for two reasons. First is the presence of small categories. As shown in Table 3, attacks such as Infiltration and SQL injection have only 5 and 50 flows. Second is the loss of flow visibility after sampling. In other words, in the presence of sampling number of flows per category decreases even more. Another important consideration when conducting CV is to ensure flow records of the same flow does not end up across multiple splits. Therefore, standard CV implementations of available tools such as scikit-learn [50] is inapplicable in this scenario. Thus, we used our custom CV implementation that ensures flows with multiple records stay in the same split. Our

implementation also ensures equal class distribution across folds as an equivalent of Stratified KFold.

b: MACRO AVERAGING

In multiclass classification, a single evaluation score needs to be computed in order to evaluate the overall performance of a classifier. When performing such aggregation, a care must be taken as there are multiple different ways to aggregate. Especially, when class imbalance is present, performance aggregation becomes even more trickier. There are three main way to average the scores of multiple classes:

- global/micro averaging where class information is disregarded and score is computed for each instance then averaged.
- weighted averaging considers frequent categories as more important. First, scores for each class is computed, then weighted average is taken. This type of averaging assigns weight to each class in proportion to their frequency.
- macro averaging treats each class equally important. First, scores for each class is computed then averaged considering each class/category as equally important.

Example usages on global averaging [11] and weighted averaging [12] can be found in the related literature. However, we could not find any study that highlighted the data imbalance issue and how it can be the source of many pitfalls with wrong type of averaging. We argue that when data is imbalanced, global or weighted averaging provides a false impression that the classifier is accurate. In other words, such averaging provides high overall score even if the only frequent categories are well distinguished and many small categories are poorly classified. This is especially the case if train data is also imbalanced which naturally leads to classifier that is biased towards frequent categories when making a decision. We propose to considering all categories as equally important. Hence, we suggest macro averaging when obtaining overall performance of the classifier by aggregating its performance across multiple classes.

c: CLASSIFIER EVALUATION METRIC

Multiclass attack classification can be evaluated with different metrics such as accuracy, precision, recall, F1-score, true positive rate, true negative rate and false alarm rate [37]. Our evaluation derives flow level metrics from record level inference. Hence, we resort to simplest yet meaningful metrics such as detection rate and false alarm rate. Detection rate for the specific malicious category is the percentage of the correctly predicted flows over the total number of flows. Overall detection rate is a metric that is aggregated across multiple malicious categories with macro averaging. False alarm rate is the percentage of benign (*i.e.,* normal) traffic that is confused with malicious categories.

When sampling is present, only percentage of flows will be observed. One way to compute detection rate and false alarm rate is to consider number of observed flows for each category as ground truth (*i.e.,* total number of flows). In such setting, detection rate of NIDS that can detect all the flows that it sees is 100%. However, as we are evaluating the effect of sampler in NIDS, evaluation metric should reflect the flows that are not observed due to loss of visibility. Therefore, our ground truth is all the flows in the complete traffic as opposed to observed flows after sampling. As an example, lets assume only 30% of the DDoS-LOIC-UDP attack is observed after sampling where packet sampling rate was 1/10. Then, in our evaluation setting, NIDS that can identify 100% of the observed flows for the given attack has only 30% detection rate. Consequently, in this setting, a sampler that samples less benign but more malicious flows positively affects NIDS performance and considered favorable.

d: DERIVING FLOW LEVEL METRIC FROM RECORDS

For the given flow, multiple records can be extracted depending on the flow expiration settings of exporting stage (more details in Section III.C). This behavior is influenced by flow cache settings for flow expiration and when flow cache becomes under-dimensioned. Similarly, presence of sampling also affects the flow expiration policy. For instance, number of generated flow records can increase due to the effect of sampling on idle timeout-based expiration. This is because due to sampling time gap between two sampled packets is larger than previous time gap between two consecutive packets. Hence, because of this inconsistent number of data units, flow record cannot be used as a unit of data for evaluation. Number of flows on the other hand is consistent and not affected by the configurations of flow metering and export stage. Therefore, we propose our flow level evaluation metrics. It is noteworthy to mention that in the deployment scenario, the classifier performs inference for each arriving record in real time. Therefore, in line with the previous work, our input to NIDS is flow record, but our evaluation is on flow level. We propose the following procedure to derive flow level evaluation metrics from record level inferences.

- Any DR: if any of the records belonging to a flow is predicted correctly, the flow is considered to be detected. If none of the records predicted correctly, then most commonly predicted category is assigned to a flow.
- Majority DR: flow is considered detected as long as most commonly predicted category for its records is correct.
- All DR: if all of the records belonging to a flow are predicted correctly, only then the flow is considered to be detected.

Any DR is suitable metric for post attack analysis where real-time detection is not important. In other words, for logging purposes detecting at least one candidate record of the flow can sufficiently pinpoint the malicious flow ID. For real-time detection, the All DR metric may be preferred as it allows the attack to be captured as soon as first record arrives to NIDS. However, in some cases, especially in under-dimensioned flow cache scenarios [1], a flow may have too many records. In such settings each record is computed over a short period from a small number of packets. As a result, it is inevitable to have some records with very low quality.

Hence, All DR scenario can be considered too harsh. Therefore, as tradeoff between Any DR and All DR, we propose Majority DR as a flow level metric. Hence, unless otherwise mentioned, our ML experiments were evaluated using the Majority DR metric.

Flow level FAR is computed from flow level DR and flow visibility rate (v) for the benign category. Here, lower flow visibility for the benign category helps NIDS to achieve low FAR.

- Any FAR (f_{any}) metric is derived from Any DR of benign category and is proportional to flow visibility rate of benign category:

$$f_{\text{any}} = (1 - d_{\text{any}}^{\text{benign}})v_{\text{benign}} \tag{1}$$

  where d corresponds to detection rate and f corresponds to false alarm rate.
- Similarly, Majority FAR (f_{majority}) is derived from Majority DR of the benign category and is proportional to flow visibility:

$$f_{\text{majority}} = (1 - d_{\text{majority}}^{\text{benign}})v_{\text{benign}} \tag{2}$$

- All FAR (f_{all}) is also derived from All DR of Benign category and is proportional to flow visibility:

$$f_{\text{all}} = (1 - d_{\text{all}}^{\text{benign}})v_{\text{benign}} \tag{3}$$

V. EXPERIMENTS

Our experiments consisted of four parts. First, we explore how flow visibility is affected by the choices of sampling technique and sampling rate in the metering process. Second we evaluated NIDS performance in using the proposed flow level evaluation metric. Performance gain after addressing data imbalance and training speed up of CNN using larger mini-batch sizes is demonstrated. Third, we investigated the performance of each sampling method with different NIDS classifiers. Finally, we investigated the effect of the constrained flow cache on the NIDS in terms of performance and resource usage. Constrained flow cache has small table size that cannot store all the Working Set of Active Flows (WSAFs) [46].

A. FLOW VISIBILITY

When sampling is employed in the flow metering and exporting stage, the flow becomes invisible if none of its packets are sampled. The loss of flow visibility is critical as it implies that no flow record is exported. This in turn implies NIDS does not have the visibility for the flow. Here, we investigated the following two questions: (1) How are different types of malicious traffic affected by sampling on different sampling rates? (2) How do sampling methods compare against each other? In this regard we sampled the traffic data using four different sampling techniques for the three sampling rates of 1/10, 1/100, and 1/1000. The corresponding records were then accumulated from the sampled packets. The corresponding flow visibility of each malicious-flow category on the four samplers across the three sampling rates are shown in Fig. 4. Each row corresponds to a separate sampling rate. The dataset contained 13 types of malicious flows, which are positioned on the x-axis in ascending order based on their average flow observation rate. Additionally, as an aggregated metric for inter-comparisons between samplers, we also provide a macro averaged observation rates for the 13 malicious categories.

1) OBSERVATIONS

Overall, the graph shows that even mild packet-sampling rates, such as 1/10, result in a significant loss of information. For instance, at a 1/10 sampling rate, only approximately 50% of normal flows were observed and, on average, 80% of malicious flows were observed. An interesting observation was that malicious flows were more visible than normal traffic. This was true for all three sampling rates and sampling techniques.

In detail, we observed the following. First, the systematic-approximate-linear sampler SFS had high visibility for long flow attacks, whereas nonlinear SGS achieved better visibility for short-length malicious flows. Second, the higher-flow visibility of the sampler depended on the sampling rate. In particular, the average malicious observation rates of both SFS and SGS were better than the baseline SRS at a 1/10 sampling rate. However, at sampling rate of 1/1000, SGS ranked lower than the baseline, while SFS still ranked better than the baseline sampler. Third, owing to its design nature, SGS attempted to capture all types of flows regardless of their size and achieved similar flow visibility in both short and long flows.

2) INSIGHTS

We can conclude that even mild sampling rates, such as 1/100, have a drastic effect on malicious-flow visibility. However, if sampling is inevitable, flow visibility can be improved by selecting the sampling technique which is suitable for the target intrusive attack based on its flow length. For instance, if short-length malicious flows are the main source of network attacks, a nonlinear SGS sampler is a good candidate. If targeted malicious have longer flow lengths, the linear samplers such as SFS are more suited to the task. The relatively better performance of the SFS over baseline SRS can be explained by its superior accuracy owing to per-flow sampling.

B. EVALUATION FRAMEWORK FOR A FLOW-LEVEL ML-BASED NIDS

Proper evaluation framework ensures the validity of experimental comparisons. In this regard we proposed flow level evaluation metric in Section IV.E that is not affected by the changes in flow metering process. Our framework, also allows us to investigate the effect of sampling.

Here, we exploit the established metric for the comparison of ML techniques. To speed up CNN training we first investigate the effect of batch size on training time and model

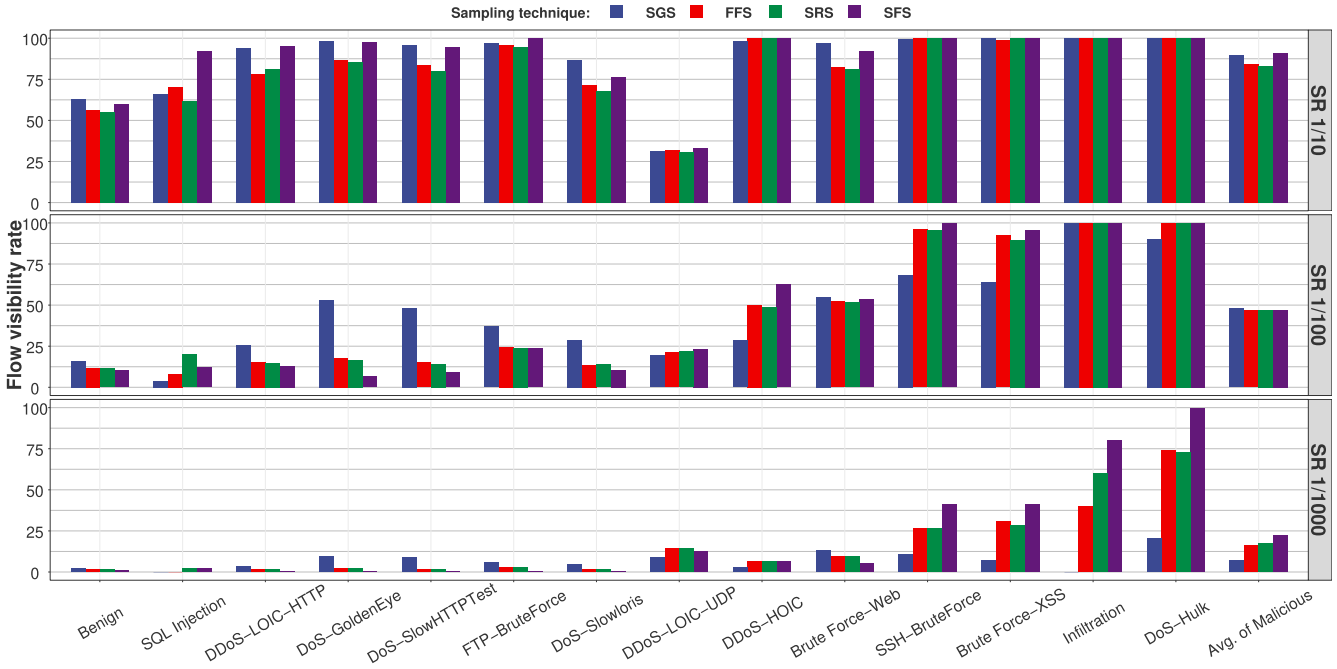

FIGURE 4. Flow visibility rate after sampling. *y*-axis represents per category flow visibility - percentage of flows retained after sampling. *x*-axis represents attack types that are positioned in ascending order based on their overall flow visibility rate. Sampling rate is reduced from 1/10 to 1/1000 with the factor of 10. Four different samplers (*i.e.,* SGS, FFS, SRS, SFS) are color-coded.

performance. Then we demonstrate the performance gain achieved by addressing data imbalance in training phase. Results herein concern non-sampled flow records and can be used later in the section as a reference to compare the performance drop when sampling is present.

Insights from these experiments add knowledge to the NIDS domain. Concretely, we (1) successfully reduced the training time of CNN model by selecting a larger training batch size without the loss of performance, (2) achieved up to 42% better DR and a 50% lower FAR by addressing data imbalance in the training set, and (3) provided fair comparisons of ML methods using the proposed flow-level evaluation metric.

1) EFFECT OF TRAINING BATCH SIZE OF CNN

Training CNNs on large data is time consuming. Using large batch sizes speeds up training by leveraging the parallel processing power of GPUs. However, previous studies suggested that large batch sizes reduce the performance of CNN models [14]. We posit that this is not necessarily valid for the NIDS task, in which flow features are low dimensional and heterogeneous. This is in opposition to image pixels that are high dimensional and homogeneous. Hence, in the following section, we explore the function of batch size in the training of a CNN model. Note this is only possible after having reliable flow-level evaluation metric which is discussed in Section IV.E. Fig 5 shows the reduction in training time as we increase the batch size. Additionally, figure shows there is no relationship between increased batch size and decreased model performance (*i.e.,* DR and FAR). Exact performance metrics are also shown in Table 7. The results are shown for the three different data balancing settings to demonstrate generalizability of the observations.

a: OBSERVATIONS

The results demonstrated that large batch sizes do not decrease the performance of CNN models on NIDS tasks. This holds true even when different balancing techniques are used. We also observed that there is no correlation between batch size and CNN model performance.

b: INSIGHTS

Our finding contrasts studies on computer vision communities [14]. This controversy can be explained by differentiating the input data of the two tasks: image and flow record. In the image domain, the dimension of image features (*i.e.,* pixels) is at a minimum of 784 for small 32×32 resolutions. In contrast, our flow record consisted of 78 features. Additionally, image pixels are homogeneous, while flow features are heterogeneous. It is heterogeneous because some features are in the units of time while others are in count, and some features such as destination port and transport protocol are categorical.

2) IMPORTANCE OF DATA BALANCING

Training on imbalanced data would result in the model being biased toward frequent categories. However, previous studies on ML-based NIDS, such as [7], [10]–[12], [36], [49] did not address this problem. In such a setting, performance metrics

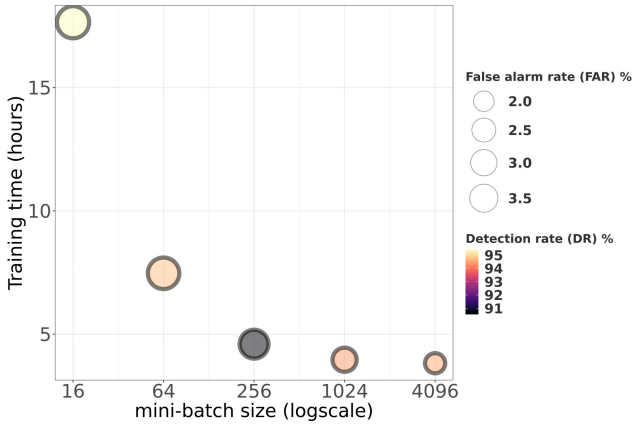

FIGURE 5. **Training time of CNN for different batch sizes.** `CIC-IDS-2018` **has 92,183,595 flow records and 4:1 train/test split is used. For each batch size training is done for 20 epochs. Detection rates are color-coded while magnitude of False alarm rates is depicted with the sizes of bubbles.**

TABLE 7. **Effect of batch size on CNN model performance using majority flow detection metric. Detection rate (DR) and false alarm rate (FAR) comparisons are provided for three different data balancing techniques.**

Metrics\ Batch size	16	64	256	1024	4096
Explicit balancing					
DR	95.39	94.75	90.55	94.39	94.51
FAR	3.59	3.4	2.75	2	1.8
Cost based balancing					
DR	15.12	43.58	63.55	64.88	73.21
FAR	31.62	10.5	38.72	30.1	31.64
No balancing					
DR	60.17	58.57	59.24	51.97	73.61
FAR	20.34	8.6	30.69	22.82	34.84

can indicate a false impression that the ML model has a high performance. This problem becomes even more dramatic if global (*i.e.,* micro) or (even worse) weighted averaging of performance scores are used (more details in Section IV.A).

To solve the data imbalance problem, the ML community primarily employs two approaches. The first is to assign a distinct cost to training examples based on the frequency of each class. The second is to re-sample training data by either over- sampling a small class or under- sampling the majority class [53], [54]. In a subsequent experiment, our aim was to indicate the performance that is lost in previous studies when data imbalance was neglected. We demonstrate this by comparing two balancing techniques against imbalanced setting. In each scenario, the ML classifier parameters were the same as described in Section IV.D. Data balancing was only applied to the training set.

As the first balancing technique, we considered under- and over- sampling [54]. From now on we use the term explicit balancing instead to eliminate the confusion with packet sampling. Small malicious categories are over- sampled to be equal in size to the most frequent malicious category. Benign records were under- sampled to be in equal size to the malicious categories.

Second balancing technique we considered is cost-based balancing (*i.e.,* cost sensitive learning. This technique modifies the optimization function to penalize more heavily when examples from infrequent categories are misclassified. For the DT and RF classifiers, we used scikit-learn [50] implemented interface by passing the weights of each class. For the CNN, we achieved cost-based balancing by passing class weights as an argument for the cross-entropy objective function in Pytorch [55]. Same balancing technique was used in traffic classification task by Aceto *et al.* [56] as well. Using the cost-based balancing, authors mitigated learning difficulty caused imbalanced data. Thus uniform misclassification rate is achieved.

Finally, to demonstrate the importance of balancing, we included the results for the no-balancing scenario. Table 8 presents the experimental results for the different balancing techniques.

a: OBSERVATIONS

The results demonstrated that the data balancing technique has a significant effect on the performance of the ML classifier. Although the cost-based balancing and no-balancing setting had no remarkable difference, explicit balancing was consistently better than no balancing on all three classifiers with a large margin. The RF classifier is known for effectively managing imbalanced datasets [48]. However, surprisingly, the balancing technique selected also has notable importance for RF. For the similar FAR, random forest with explicit balancing had 98.31% DR, while for cost-based balancing only 69.08% DR is achieved.

b: INSIGHTS

Major insight here is that without the careful design of the evaluation setting, fallacious conclusions can be derived. For instance, when data imbalance is not addressed one may conclude that the DT classifier has a high DR of 78.62%, whereas its more sophisticated predecessor RF has a 68.92% DR and a 0% FAR. However, here, neither classifier achieved its potential performance owing to data imbalance issue. Hence, such comparison leads to wrong conclusion. Another surprising insight is that cost-based balancing did not provide notable improvement as opposed to explicit balancing. Moreover, for tree-based classifiers(*i.e.,* DT and RF), the data imbalance problem can be converted into an advantage if reducing the FAR is the first priority.

3) COMPARISON OF CLASSIFIERS

Although different ML methods have been proposed previously, their reliable comparison on `CIC-IDS-2018` dataset in flow level is missing. In Table 9, we provide the DR and FAR comparisons of the three most popular classifiers that are used for NIDS. The results are shown for all three flow level detection metrics proposed in Section IV.E.

TABLE 8. Effect of data balancing on ML-based NIDS performance. Comparisons are provided using flow level Majority DR and Majority FAR metrics. Experiments include three different cases of data balancing.

Metrics\ Classifiers	DT	RF	CNN
Explicit Balancing			
DR	97.22	98.31	95.64
FAR	0.09	0.01	0.16
Cost based Balancing			
DR	95.82	69.08	76.84
FAR	19.41	0.02	25.91
No Balancing			
DR	78.62	68.92	76.87
FAR	0	0	30.72

For the training phase, explicit balancing was employed, and the other hyperparameters were the same as described in Section IV.D.

a: OBSERVATIONS

For all three flow-level detection metrics, RF had better or comparable performance, except for the All metric, for which the CNN had a higher DR but suffered from high FAR. As explained in Section IV.E, Any and All metrics were too extreme. Hence, we recommend referring to the Majority metric for the comparison of different models. Hence, RF was considered the most suitable because it had a high DR of 98.31% and an FAR of 0.01% on Majority DR and Majority FAR metrics.

TABLE 9. Detection rate (DR) and false alarm rate (FAR) comparisons of three classifiers on non-sampled flow records. Results are provided for three flow level evaluation metrics. Train data is balanced using explicit balancing.

Flow level metric	Any		Majority		All	
Classifier	**DR**	**FAR**	**DR**	**FAR**	**DR**	**FAR**
DT	**99.09**	0.08	97.22	0.09	84.18	1.46
RF	99.08	**0.01**	**98.31**	**0.01**	90.0	**0.2**
CNN	96.74	0.05	95.64	0.16	**94.51**	1.8

C. FLOW-BASED NIDS ON SAMPLED DATA

Here, we investigate how the presence of sampling affects NIDS performance in terms of flow level Majority DR and Majority FAR metrics. Four different sampling techniques are compared for their suitability for NIDS application. Our ML experiments are done in three different ML classifiers (*i.e.,* DT, RF, CNN). As a result each of three classifiers are evaluated on four times on different data. Different data corresponds to flow records extracted in the presence of different sampling techniques. In addition to suitable sampler for NIDS, this setting enables us to observe which classifier and sampler are a good pair. Experiments are repeated three times for sampling rates of 1/10, 1/100, and 1/1000. This allows us to explore the effect of sampling rate. It is noteworthy to mention that based on the definition of our metrics in Section IV.E, FAR cannot exceed benign flow visibility rate.

Fig. 6 shows the performance of the ML classifiers on each sampling method. The first and second columns show the DR and FAR, respectively, whereas each row corresponds to different sampling rates. ML classifiers are color-coded. One-way ANOVA test is conducted to verify the difference in the performance of ML methods are statistically significant. Each cell in Table 10 corresponds to separate test. ANOVA test quantifies if at least one of the three classifiers have significantly different mean detection rate. According to test results, almost each comparison has at least one ML classifier with statistically significant difference (*i.e.,*low p-value). This applies for both Majority DR and Majority FAR metrics. Exception is observed for SGS ($p = 0.167$) and SRS ($p = 0.656$) on sampling rate of 1/100 for Majority DR.

b: COMPARISONS ON MULTIPLE THRESHOLDS

To compare classifiers in multiple cut-off points (similar to ROC curve), above evaluation procedure was repeated in ten different benign thresholds. In previous experiment, benign and other thirteen malicious categories were treated the same where a category with the highest probability was assigned to a flow record. We apply the given benign threshold on a record level as NIDS returns the estimated probability of each category for the given record and not the flow itself. If the predicted probability of that record for the benign class is larger than the threshold, then we assign the label 'benign' to that record. Otherwise, the malicious attack category with the highest probability is assigned. Once records are assigned with their corresponding labels, then flow level metric is derived using the 'Majority' procedure explained in Section IV.E. Results for multiple threshold comparison is shown in Fig. 7 and Fig. 8.

c: OBSERVATIONS

At a high level, we observe that sampling had the disadvantage of increasing the FAR by orders of magnitude. For instance, for a sampling rate of 1/10 and RF classifier, we observed, at best, a 0.5% FAR for the SFS sampler as opposed to a 0.01% FAR on non-sampled data, which was 50 times better. For the same SFS and RF pair, at a sampling rate of 1/1000, an FAR of 0.03 % was obtained, but at the expense of missing out 85% of the malicious flows.

When comparing classifiers, we can observe from the results that, for any given sampling rate, RF has a higher DR with a better or comparable FAR. Another observation is that on low sampling rates such as 1/1000, the selected classifier has a relatively minimal effect for a particular sampler.

For the harsher sampling rates, the best pair was the SFS sampler with the RF classifier. Specifically, at a 1/10 sampling rate, SFS had a high DR (80%) with a comparably low FAR (0.5%), and at 1/100, SFS had the third-highest DR,

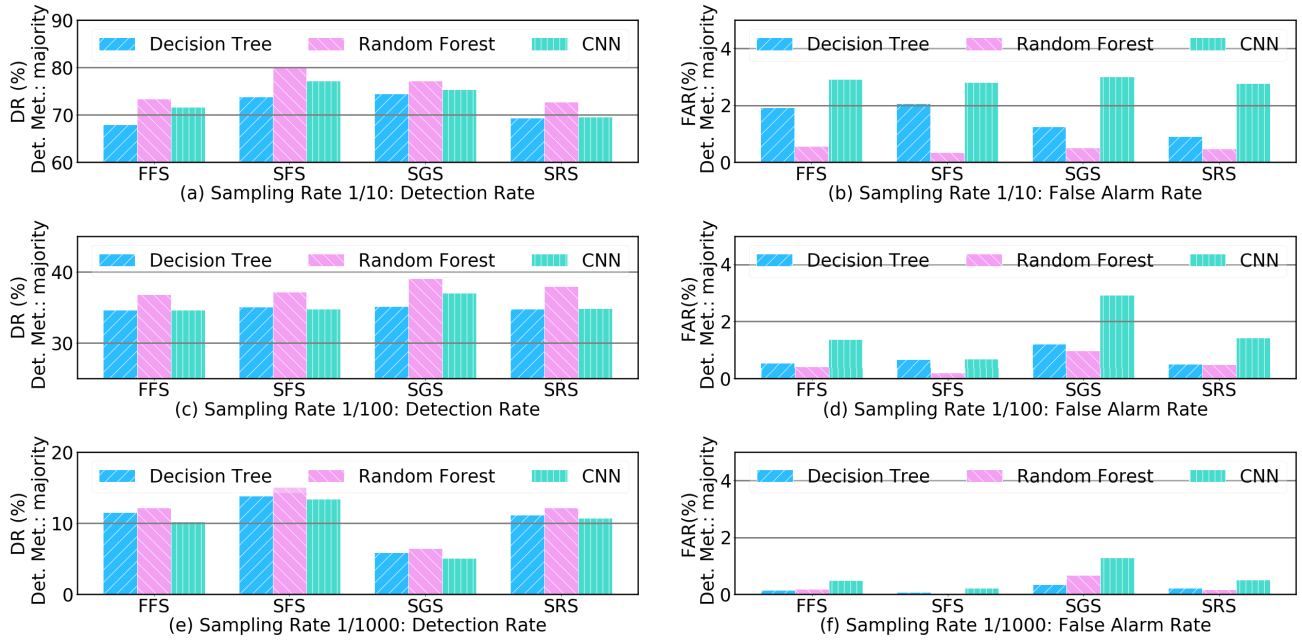

FIGURE 6. Performance of ML classifiers on a given sampler. *x*-axis represents choice of sampler deployed during flow information export. *y*-axis represent DR and FAR. FAR is reduced as harsher sampling rates used. flow records are obtained from sampled packets which sampling rates of 1/10, 1/100 and 1/1000. Simple Random Sampling (SRS), Fast Filtered Sampling (FFS), SketchFlow Sampling (SFS) and Sketch-Guided Sampling (SGS) samplers are color-coded.

TABLE 10. One-way ANOVA test for significant difference between the means of different ML classifiers. Each cell corresponds to separate ANOVA test conducted over three classifiers trained and evaluated on sampled flow records. Those flow records are extracted in the presence of corresponding sampler.

Flow level Detection Metric	Sampling rate	FFS	SFS	SGS	SRS
Majority DR	1/10	F(2,27)=8.276, p=0.002	F(2,27)=12.074, p<0.001	F(2,27)=9.591, p=0.001	F(2,27)=18.307, p<0.001
	1/100	F(2,27)=8.819, p=0.001	F(2,27)=6.755, p=0.004	F(2,27)=1.912, p=0.167	F(2,27)=0.429, p=0.656
	1/1000	F(2,27)=9.640, p=0.001	F(2,27)=5.523, p=0.010	F(2,27)=28.998, p<0.001	F(2,27)=23.789, p<0.001
Majority FAR	1/10	F(2,27) = 4.068, p=0.029	F(2,27) = 23.803, p<0.001	F(2,27) = 45.341, p<0.001	F(2,27) = 57.381, p<0.001
	1/100	F(2,27) = 23.051, p<0.001	F(2,27) = 5.466, p=0.010	F(2,27) = 14.986, p<0.001	F(2,27) = 22.403, p<0.001
	1/1000	F(2,27) = 44.499, p<0.001	F(2,27) = 26.303, p<0.001	F(2,27) = 46.648, p<0.001	F(2,27) = 55.297, p<0.001

but the best FAR at 1/1000. Hence, SFS had both a high DR and a lower FAR. Perhaps, an undesirable sampler for NIDS is SGS, as its corresponding FAR continued to worsen as higher sampling rates were applied. In particular, at a 1/1000 sampling rate, SGS had an FAR that was twice as large as that of SFS but also had a lower DR.

d: INSIGHTS

A significant increase in the FAR suggests that sampling significantly reduces the quality of the features. This insight should be considered when network or data center operators decide whether to buy hardware or employ sampling.

If sampling must be employed, per-flow systematic samplers such as SFS should be favored over the baseline SRS or nonlinear samplers such as SGS. This could be explained by their systematic nature for measuring features such as the inter-arrival time between two packets.

D. EFFECT OF CONSTRAINED FLOW CACHE ON NIDS

Many flow exporters have fixed-size flow caches. When this size turns out to be small (under-dimensioned), flow data loss or low performance can be the result [1]. In this study, we explored the effect of under-dimensioned flow cache from the perspective of real-time NIDS. If the flow cache is not sufficiently large, all of its entries are occupied at some point, and there is no room to create a new entry for the incoming packet of a new flow. For our setting, when the flow cache is full, the least recently updated flow record is kicked out. Kicked-out records are still sent to the collector, and the collector then passes the record to the NIDS in real time. However, such a scenario creates a problem in which there are now more records to analyze, and the quality of each record is degraded because it is estimated from fewer packets.

For instance, assume flow f_1 has two packets that have a smaller inter-arrival time than both active and idle timeouts.

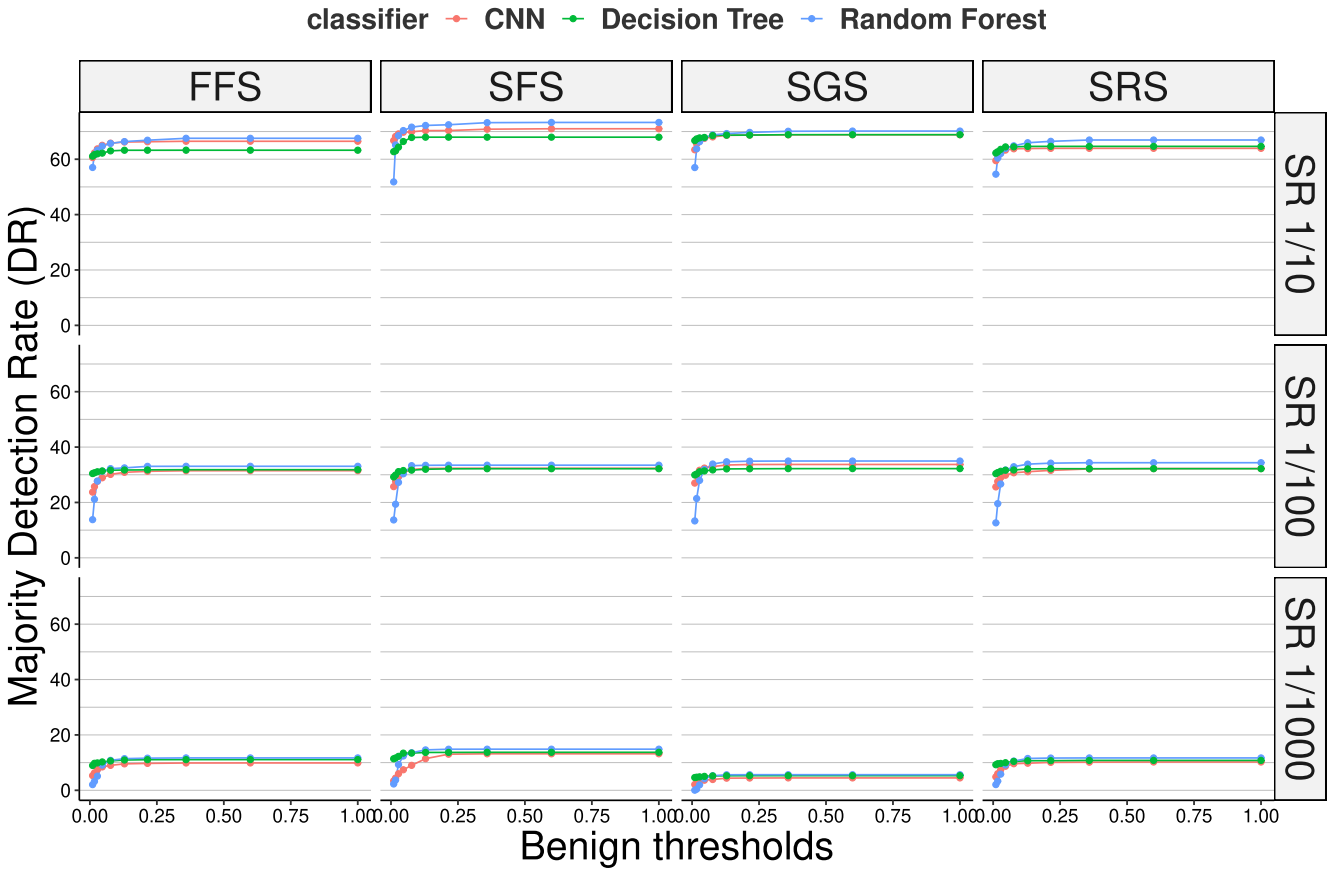

FIGURE 7. Corresponding evaluation curve for Majority DR on multiple benign thresholds for the results shown in Fig. 6.

Subsequently, let us assume two flow cache scenarios: one with unlimited flow cache size and second with constrained flow cache. In the former, a single corresponding record r_1 is extracted as normally. In the latter, before the second packet of the flow f_1 arrives, the flow cache becomes under-dimensioned, and to make empty entry for new incoming flow f_2, record r_1^1 of flow f_1 is prematurely kicked out. When the second packet of flow f_1 arrives, a new record r_1^2 is formed and later exported. We posit that the quality of both r_1^1 and r_1^2 records are lower than that of a single extracted record r_1 in the first case.

In summary, under-dimensioned flow cache generates more flow records due to premature kickouts. It also causes flow records to be less accurate. This in turn leads to degradation in NIDS performance and higher computing resource consumption by NIDS. In the following, we first quantify how the DR of NIDS is affected when the flow cache becomes under-dimensioned. Next we analyze how CPU resource usage time increases as a function of the number of extracted flow records.

Our setting for this experiment was as follows. We found the ideal flow cache size for each of the nine PCAP traces of `CIC-IDS-2018`. Here, ideal cache size was the minimum number of table entries that was sufficient to store all working sets of active flows in the flow cache until they were naturally (not prematurely) exported. We considered ideal cache size as 100%. Subsequently, we reduced the flow cache size from 100% to 0.1% with a factor of 10 to simulate the constrained scenario. For each of four (*i.e.,* 100%, 10%, 1% and 0.1%) cache size, four different flow records were extracted. They correspond to four different samplers that were deployed in flow metering stage.

We assumed that the training phase of ML classifier was performed on the flow records that were extracted when the flow cache size was sufficient (*i.e.,* 100%). Therefore, flow cache size affected NIDS only during inference phase and not in training phase. RF was used as a candidate NIDS classifier due to its superior performance over DT and CNN in Section V.C. To quantify inference time, we assumed that the NIDS made inferences for each flow record individually and had no parallel processing functionality. Our scikit-learn [50] implementation of RF used a single CPU core for inference. The system had an Intel(R) Xeon(R) E5-2640 v4 @ 2.40-GHz CPU with 40 cores and 64 G RAM.

1) EFFECT ON THE NIDS DETECTION RATE

Fig. 9 shows the effect of a constrained flow cache on the NIDS DR for four scenarios in terms of the proportion of the

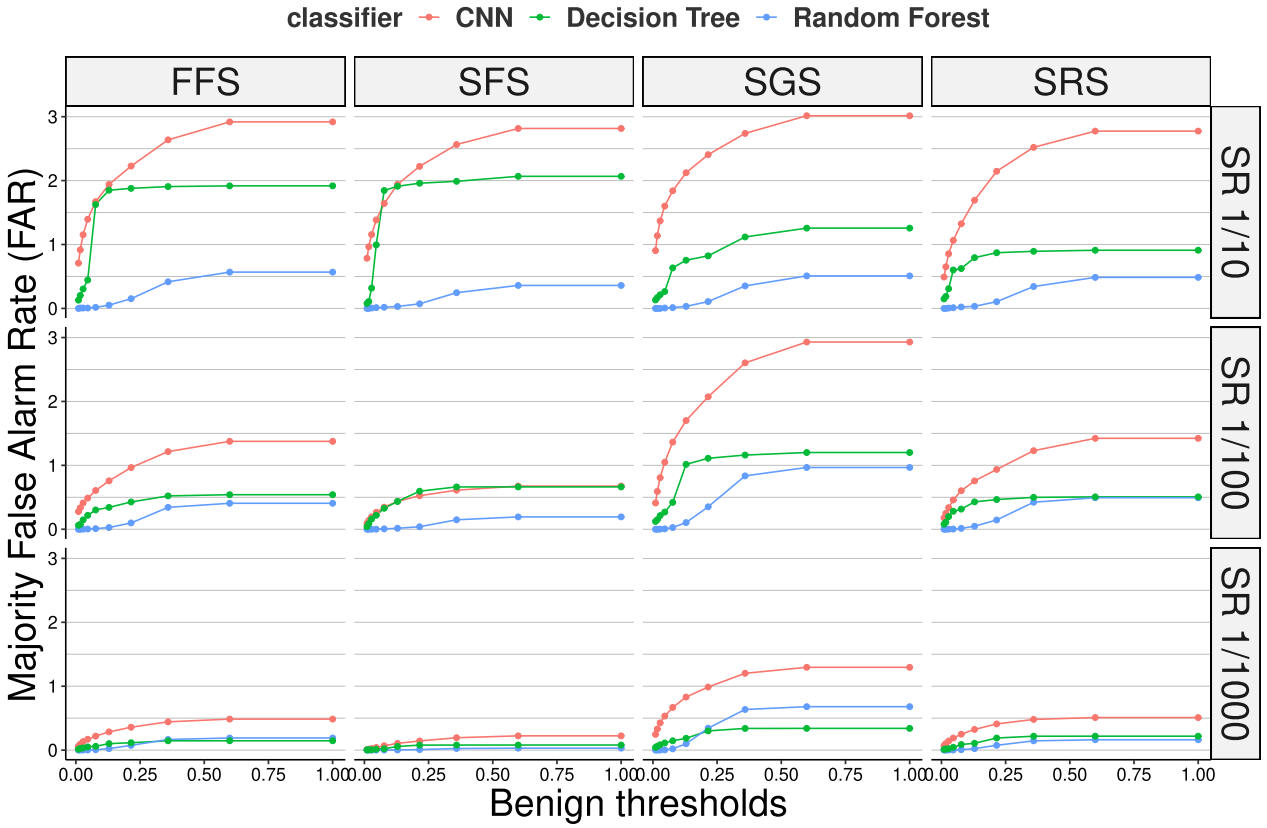

FIGURE 8. Corresponding evaluation curve for Majority FAR on multiple benign thresholds for the results shown in Fig. 6.

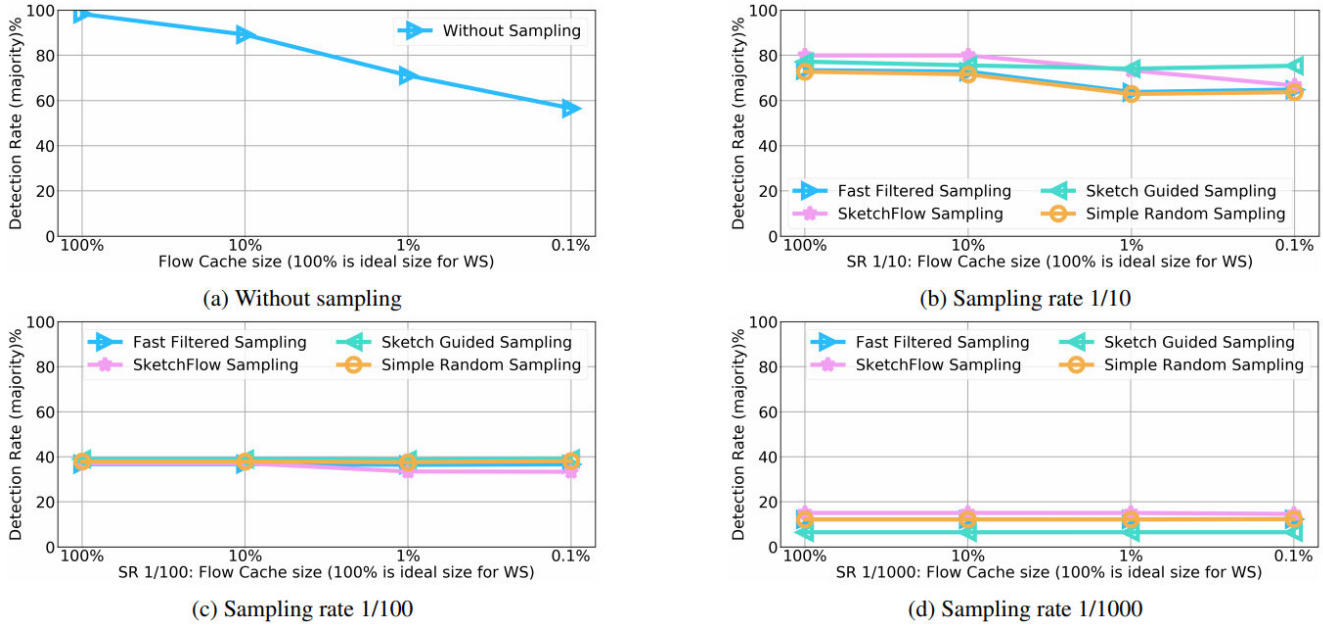

FIGURE 9. Impact of under-dimensioned Flow Cache on NIDS Detection Rate: Each subfigure corresponds to different sampling rates. *x*-axis depicts flow-level detection rate with majority metric. *y*-axis shows the flow cache size as the percentage of ideal size. Simple Random Sampling (SRS), Fast Filtered Sampling (FFS), SketchFlow Sampling (SFS) and Sketch-Guided Sampling (SGS) samplers are color-coded.

packets used to estimate flow records at the sampling rates of 1/1, 1/10, 1/100, and 1/1000. The sampling rate 1/1 is equivalent to no sampling.

a: OBSERVATIONS

Overall, we observed that the constrained flow cache degraded the NIDS DR the most when no sampling was

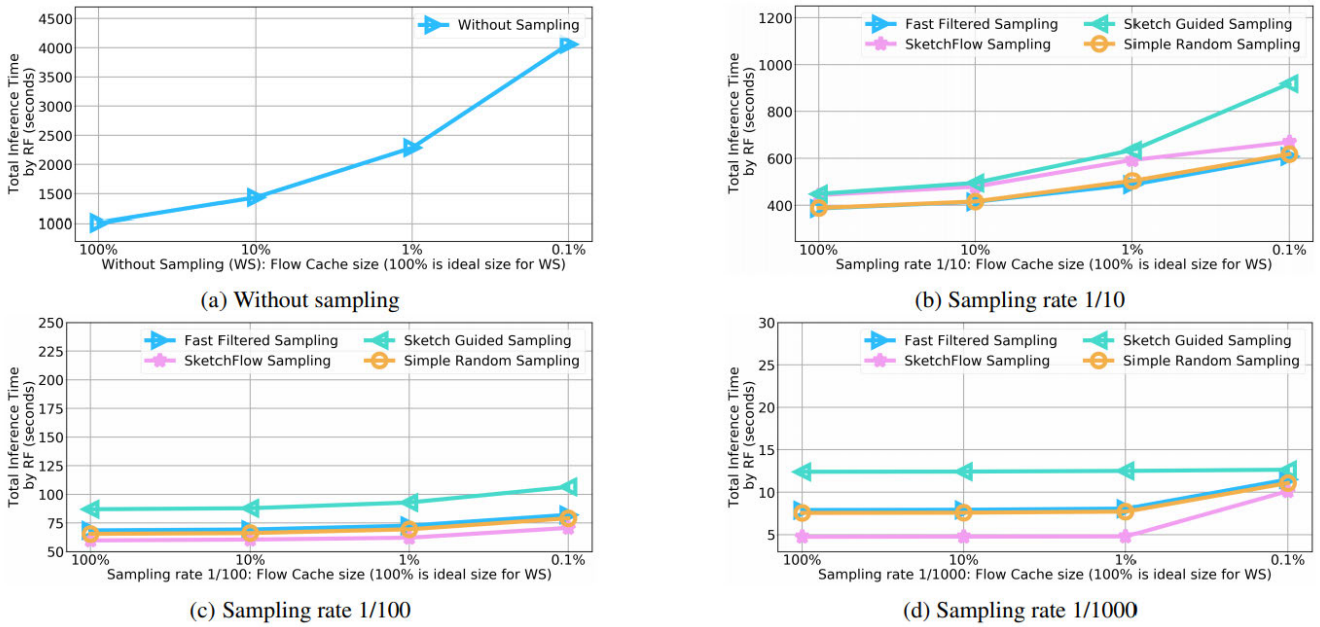

FIGURE 10. Impact of under-dimensioned Flow Cache on NIDS Recourse Usage (Inference Time). Ideal flow cache size is the minimum size that ensures no active flow is prematurely kicked out during measuring process. Flow cache size is varied between 100% to 0.1% with 10 times decrease interval where 100% refers to ideal flow cache size that is just enough to measure flows without premature kickout. Simple Random Sampling (SRS), Fast Filtered Sampling (FFS), SketchFlow Sampling (SFS) and Sketch-Guided Sampling (SGS) samplers are color-coded.

used. In particular, when the flow cache was 0.1%, the WS scenario had a DR of less than 60%, whereas samplers with 1/10 sampling rate had DRs higher than 60%. At the 1/10 sampling rate, the DR of each sampling method decreased slightly for flow cache of sizes 1% and 0.1%, except for the SGS. Intuitively, flow cache size had almost no effect on low sampling rates.

b: INSIGHTS

When the flow cache is under-dimensioned, it seems better strategy is to apply sampling with carefully selected sampling rate that will keep the DR relatively higher. This is because the use of sampling has fewer premature kick-outs, which preserves the flow-record quality relatively better. However, the sampling rate should be set very carefully as excessively harsh sampling rates may not fully utilize the flow cache but reduce the flow-record quality owing to records being estimated from fewer packets.

2) EFFECT ON NIDS RESOURCE USAGE

NIDS consumes certain CPU resources when making inference for each arriving flow record. Hence, we explored the cost in terms of CPU-time usage by the NIDS when flow cache is under-dimensioned. Fig. 10 shows the increasing trend on the total inference time of all records as we constrained the flow cache size. The results for four different flow cache sizes are indicated. The y-axis indicates the total CPU time required when making inference for all the exported flow records. The increase in the total inference time is caused by an increase in the number of flow records.

a: OBSERVATIONS

Overall, the total NIDS inference time increased when the flow cache was under-dimensioned. However, when the sampling rate was harsher, premature kick-outs owing to the constrained flow cache size were less frequent.

b: INSIGHTS

In resource-constrained settings, to reduce the CPU time of NIDS resources, employing sampling is beneficial as it reduces the rate of premature flow-record kick-outs in contrast to without sampling scenario.

VI. CONCLUSION AND FUTURE PERSPECTIVES

We proposed an evaluation framework for the proper evaluation of ML-based NIDS. To the best of our knowledge, we are the first to propose flow-level NIDS evaluation framework that is applicable even in the presence of sampling. Using proposed evaluation framework we demonstrated a remarkable performance gain achieved by addressing training-data imbalance. Sampling experiments show that 50% of the malicious flows are not exported even with mild 1/10 sampling rate. Our investigation on the feasibility of the state-of-the-art ML-based NIDS in the presence of sampling reveal that generally sampling degrades NIDS performance. However, we observed that sampling can also increase performance compared with no sampling when resources such as the flow cache of the measuring device (*e.g.,* switch) are constrained.

This study was the first attempt to explore the effect of sampling in ML-based NIDSs. To keep the scope manageable, we only focused on misuse detection based NIDS.

In future, effect of sampling on anomaly based NIDS should be investigated as well. Another limitation of current study is its relatively small scale where we investigated only three sampling rates, four sampling techniques and three ML methods. Our flow cache experiments were also limited to four different cache sizes. Therefore, in future larger scale studies could be carried out in multiple dimensions. One direction is to investigate on the effect of sampling with exhaustive sampling rates. Another direction is to cover comprehensive list of sampling techniques and ML/DL methods. Finally, conducting such experiments in multiple, diverse datasets provides more rigorous arguments in the effect of sampling on NIDS performance.

REFERENCES

[1] R. Hofstede, B. Trammell, I. Drago, R. Sadre, A. Sperotto, and A. Pras, "Flow monitoring explained: From packet capture to data analysis with NetFlow and IPFIX," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 4, pp. 2037–2064, 4th Quart., 2014, doi: 10.1109/COMST.2014.2321898.

[2] A. Sperotto and A. Pras, "Flow-based intrusion detection," in *Proc. 12th IFIP/IEEE Int. Symp. Integr. Netw. Manage. (IM) Workshops*, May 2011, pp. 958–963, doi: 10.1109/INM.2011.5990529.

[3] B. Claise, B. Trammell, and P. Aitken, *Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information*, document RFC 7011, Internet Requests for Comments, RFC Editor, Sep. 2013. [Online]. Available: http://www.rfc-editor.org/rfc/rfc7011.txt

[4] M. F. Umer, M. Sher, and Y. Bi, "Flow-based intrusion detection: Techniques and challenges," *Comput. Secur.*, vol. 70, pp. 238–254, Sep. 2017, doi: 10.1016/j.cose.2017.05.009.

[5] B. Claise, *Cisco Systems NetFlow Services Export Version 9*, document RFC 3954, Internet Requests for Comments, RFC Editor, Oct. 2004. [Online]. Available: http://www.rfc-editor.org/rfc/rfc3954.txt

[6] V. Kumar. *Network Intrusion Detection on UNSW-NB15*. Github. Accessed: Sep. 2021. [Online]. Available: https://github.com/vinayakumarr/Network-IntrusionDetection/blob/master/UNSW-NB15/CNN/multiclass/cnn2.py

[7] R. Vinayakumar, M. Alazab, K. Soman, P. Poornachandran, A. Al-Nemrat, and S. Venkatraman, "Deep learning approach for intelligent intrusion detection system," *IEEE Access*, vol. 7, pp. 41525–41550, 2019, doi: 10.1109/ACCESS.2019.2895334.

[8] J. Kevric, S. Jukic, and A. Subasi, "An effective combining classifier approach using tree algorithms for network intrusion detection," *Neural Comput. Appl.*, vol. 28, no. S1, pp. 1051–1058, Dec. 2017, doi: 10.1007/s00521-016-2418-1.

[9] N. Sultana, N. Chilamkurti, W. Peng, and R. Alhadad, "Survey on SDN based network intrusion detection system using machine learning approaches," *Peer-Peer Netw. Appl.*, vol. 12, no. 2, pp. 493–501, Jan. 2019, doi: 10.1007/s12083-017-0630-0.

[10] A. Javaid, Q. Niyaz, W. Sun, and M. Alam, "A deep learning approach for network intrusion detection system," in *Proc. 9th EAI Int. Conf. Bio-Inspired Inf. Commun. Technol. (BIONETICS)*, 2016, pp. 21–26, doi: 10.4108/eai.3-12-2015.2262516.

[11] N. Shone, T. N. Ngoc, V. D. Phai, and Q. Shi, "A deep learning approach to network intrusion detection," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 2, no. 1, pp. 41–50, Feb. 2018, doi: 10.1109/TETCI.2017.2772792.

[12] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *Proc. 4th Int. Conf. Inf. Syst. Secur. Privacy*, 2018, pp. 108–116, doi: 10.5220/0006639801080116.

[13] M. A. Ferrag, L. A. Maglaras, H. Janicke, and R. Smith, "Deep learning techniques for cyber security intrusion detection: A detailed analysis," in *Proc. Electron. Workshops Comput.*, vol. 10, Sep. 2019, pp. 126–136, doi: 10.14236/ewic/icscsr19.16.

[14] D. Masters and C. Luschi, "Revisiting small batch training for deep neural networks," 2018, *arXiv:1804.07612*.

[15] K. C. Claffy, G. C. Polyzos, and H.-W. Braun, "Application of sampling methodologies to network traffic characterization," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 23, no. 4, pp. 194–203, Oct. 1993, doi: 10.1145/167954.166256.

[16] M. Wang, B. Li, and Z. Li, "SFlow: Towards resource-efficient and agile service federation in service overlay networks," in *Proc. 24th Int. Conf. Distrib. Comput. Syst.*, 2004, pp. 628–635, doi: 10.1109/ICDCS.2004.1281630.

[17] *Using NetFlow Sampling to Select the Network Traffic to Track*. Accessed: Sep. 7, 2021. [Online]. Available: https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/netflow/configuration/xe-3s/nf-xe-3s-book/nflow-filt-samp-traff-xe.pdf

[18] R. Jang, D. Min, S. Moon, D. Mohaisen, D. Nyang, and S. S. Event, "SketchFlow: Per-flow systematic sampling using sketch saturation event," in *Proc. IEEE Conf. Comput. Commun. (IEEE INFOCOM)*, Jul. 2020, pp. 1339–1348, doi: 10.1109/INFOCOM41043.2020.9155252.

[19] A. Kumar and J. Xu, "Sketch guided sampling–using on-line estimates of flow size for adaptive data collection," in *Proc. 25th IEEE Int. Conf. Comput. Commun. (IEEE INFOCOM)*, Apr. 2006, pp. 1–12, doi: 10.1109/INFOCOM.2006.326.

[20] C. Hu, B. Liu, S. Wang, J. Tian, Y. Cheng, and Y. Chen, "ANLS: Adaptive non-linear sampling method for accurate flow size measurement," *IEEE Trans. Commun.*, vol. 60, no. 3, pp. 789–798, Mar. 2012, doi: 10.1109/TCOMM.2011.112311.100622.

[21] A. Ramachandran, S. Seetharaman, N. Feamster, and V. Vazirani, "Fast monitoring of traffic subpopulations," in *Proc. 8th ACM SIGCOMM Conf. Internet Meas. Conf. (IMC)*, Vouliagmeni, Greece, 2008, pp. 257–270, doi: 10.1145/1452520.1452551.

[22] J. Mai, A. Sridharan, H. Zang, and C.-N. Chuah, "Fast filtered sampling," *Comput. Netw.*, vol. 54, no. 11, pp. 1885–1898, Aug. 2010, doi: 10.1016/j.comnet.2010.01.015.

[23] N. Duffield, C. Lund, and M. Thorup, "Learn more, sample less: Control of volume and variance in network measurement," *IEEE Trans. Inf. Theory*, vol. 51, no. 5, pp. 1756–1775, May 2005, doi: 10.1109/TIT.2005.846400.

[24] N. Hohn and D. Veitch, "Inverting sampled traffic," *IEEE/ACM Trans. Netw.*, vol. 14, no. 1, pp. 68–80, Feb. 2006, doi: 10.1109/TNET.2005.863456.

[25] P. Tune and D. Veitch, "Towards optimal sampling for flow size estimation," in *Proc. 8th ACM SIGCOMM Conf. Internet Meas. Conf. (IMC)*, 2008, pp. 243–256, doi: 10.1145/1452520.1452550.

[26] R. Koch, "Towards next-generation intrusion detection," in *Proc. 3rd Int. Conf. Cyber Conflict*, 2011, pp. 151–168.

[27] J.-H. Jun, C.-W. Ahn, and S.-H. Kim, "DDoS attack detection by using packet sampling and flow features," in *Proc. 29th Annu. ACM Symp. Appl. Comput.*, Mar. 2014, pp. 711–712, doi: 10.1145/2554850.2555109.

[28] T. Ha, S. Kim, N. An, J. Narantuya, C. Jeong, J. Kim, and H. Lim, "Suspicious traffic sampling for intrusion detection in software-defined networks," *Comput. Netw.*, vol. 109, pp. 172–182, Nov. 2016, doi: 10.1016/j.comnet.2016.05.019.

[29] G. Androulidakis and S. Papavassiliou, "Improving network anomaly detection via selective flow-based sampling," *IET Commun.*, vol. 2, no. 3, pp. 399–409, Mar. 2008, doi: doi:10.1049/iet-com:20070231.

[30] J. Mai, A. Sridharan, C.-N. Chuah, H. Zang, and T. Ye, "Impact of packet sampling on portscan detection," *IEEE J. Sel. Areas Commun.*, vol. 24, no. 12, pp. 2285–2298, Dec. 2006, doi: 10.1109/JSAC.2006.884027.

[31] J. Mai, C.-N. Chuah, A. Sridharan, T. Ye, and H. Zang, "Is sampled data sufficient for anomaly detection?" in *Proc. 6th ACM SIGCOMM Internet Meas. (IMC)*, 2006, pp. 165–176, doi: 10.1145/1177080.1177102.

[32] H. H. Jazi, H. Gonzalez, N. Stakhanova, and A. A. Ghorbani, "Detecting HTTP-based application layer DoS attacks on web servers in the presence of sampling," *Comput. Netw.*, vol. 121, pp. 25–36, Jul. 2017, doi: 10.1016/j.comnet.2017.03.018.

[33] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 2, pp. 1153–1176, Oct. 2016, doi: 10.1109/COMST.2015.2494502.

[34] G. Bovenzi, G. Aceto, D. Ciuonzo, V. Persico, and A. Pescape, "A hierarchical hybrid intrusion detection approach in IoT scenarios," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2020, pp. 1–7, doi: 10.1109/GLOBECOM42002.2020.9348167.

[35] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: An ensemble of autoencoders for online network intrusion detection," 2018, *arXiv:1802.09089*.

[36] C. Yin, Y. Zhu, J. Fei, and X. He, "A deep learning approach for intrusion detection using recurrent neural networks," *IEEE Access*, vol. 5, pp. 21954–21961, 2017, doi: 10.1109/ACCESS.2017.2762418.

[37] A. Alharbi, W. Alosaimi, H. Alyami, H. T. Rauf, and R. Damaševičius, "Botnet attack detection using local global best bat algorithm for industrial Internet of Things," *Electronics*, vol. 10, no. 11, p. 1341, Jun. 2021, doi: 10.3390/electronics10111341.
[38] J. Kim, M. Shim, S. Hong, Y. Shin, and E. Choi, "Intelligent detection of IoT botnets using machine learning and deep learning," *Appl. Sci.*, vol. 10, no. 19, p. 7009, Oct. 2020, doi: 10.3390/app10197009.
[39] A. H. Lashkari. *CICFlowmeter-V4.0. Github*. Accessed: Sep. 2021. [Online]. Available: https://github.com/ahlashkari/CICFlowMeter
[40] *NetFlow*. Accessed: Sep. 2021. [Online]. Available: https://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/index.html
[41] N. Brownlee, C. Mills, and G. Ruth, *Traffic Flow Measurement: Architecture*, document RFC 2722, Internet Requests for Comments, RFC Editor, Oct. 1999. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc2722
[42] J. Quittek, T. Zseby, B. Claise, and S. Zander, *Requirements for IP Flow Information Export (IPFIX)*, document RFC 3917, Internet Requests for Comments, RFC Editor, Oct. 2004. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc3917
[43] A. Dainotti, A. Pescape, and K. C. Claffy, "Issues and future directions in traffic classification," *IEEE Netw.*, vol. 26, no. 1, pp. 35–40, Jan./Feb. 2012, doi: 10.1109/MNET.2012.6135854.
[44] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," in *Proc. IEEE Symp. Comput. Intell. Secur. Defense Appl.*, Jul. 2009, pp. 1–6, doi: 10.1109/CISDA.2009.5356528.
[45] A. H. Lashkari. (2019). *CSE-CIC-IDS 2018 on AWS*. [Online]. Available: https://www.unb.ca/cic/datasets/ids-2018.html
[46] R. Jang, S. Moon, Y. Noh, A. Mohaisen, and D. Nyang, "InstaMeasure: Instant per-flow detection using large in-DRAM working set of active flows," in *Proc. IEEE 39th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2019, pp. 2047–2056, doi: 10.1109/ICDCS.2019.00202.
[47] Y. Li, R. Miao, C. Kim, and M. Yu, "FlowRadar: A better netflow for data centers," in *Proc. 13th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, Santa Clara, CA, USA, Mar. 2016, pp. 311–324.
[48] P. A. A. Resende and A. C. Drummond, "A survey of random forest based methods for intrusion detection systems," *ACM Comput. Surveys*, vol. 51, no. 3, pp. 1–36, Jul. 2018, doi: 10.1145/3178582.
[49] R. Vinayakumar, K. P. Soman, and P. Poornachandran, "Applying convolutional neural network for network intrusion detection," in *Proc. Int. Conf. Adv. Comput., Commun. Informat. (ICACCI)*, Sep. 2017, pp. 1222–1228, doi: 10.1109/ICACCI.2017.8126009.
[50] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, and J. Vanderplas, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Jan. 2011.
[51] K. Hao. *Training a Single AI Model Can Emit as Much Carbon as Five Cars in Their Lifetimes*. Accessed: Sep. 7, 2021. [Online]. Available: https://www.technologyreview.com/2019/06/06/239031/training-a-single-ai-model-can-emit-as-much-carbon-as-five-cars-in-their-lifetimes/
[52] A. Gulli and S. Pal. *Keras*. Accessed: Sep. 2021. [Online]. Available: https://keras.io/
[53] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," *J. Artif. Intell. Res.*, vol. 16, no. 1, pp. 321–357, 2002, doi: 10.1613/jair.953.
[54] M. Bach, A. Werner, J. Żywiec, and W. Pluskiewicz, "The study of under- and over-sampling methods' utility in analysis of highly imbalanced data on osteoporosis," *Inf. Sci.*, vol. 384, pp. 174–190, Apr. 2017, doi: 10.1016/j.ins.2016.09.038.
[55] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, and A. Desmaison, "PyTorch: An imperative style, high-performance deep learning library," in *Proc. NeurIPS*, 2019, pp. 8024–8035.
[56] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapè, "MIMETIC: Mobile encrypted traffic classification using multimodal deep learning," *Comput. Netw.*, vol. 165, Dec. 2019, Art. no. 106944, doi: 10.1016/j.comnet.2019.106944.
[57] M. Roesch, "Snort–lightweight intrusion detection for networks," in *Proc. 13th USENIX Conf. Syst. Admin.*, 1991, pp. 229–238.
[58] A. Rouari, A. Moussaoui, Y. Chahir, H. T. Rauf, and S. Kadry, "Deep CNN-based autonomous system for safety measures in logistics transportation," *Soft Comput.*, vol. 25, pp. 1433–7479, Jun. 2021.
[59] A. Daniel, K. Subburathinam, B. A. Muthu, N. Rajkumar, S. Kadry, R. K. Mahendran, and S. Pandian, "Procuring cooperative intelligence in autonomous vehicles for object detection through data fusion approach," *IET Intell. Transp. Syst.*, vol. 14, no. 11, pp. 1410–1417, Nov. 2020.
[60] F. Khan, R. L. Kumar, S. Kadry, Y. Nam, and M. N. Meqdad, "Autonomous vehicles: A study of implementation and security," *Int. J. Elect. Comput. Eng.*, vol. 11, no. 4, pp. 3013–3021, 2021.
[61] S. Malik, H. A. Khattak, Z. Ameer, U. Shoaib, H. T. Rauf, and H. Song, "Proactive scheduling and resource management for connected autonomous vehicles: A data science perspective," *IEEE Sensors J.*, vol. 21, no. 22, pp. 25151–25160, Nov. 2021, doi: 10.1109/JSEN.2021.3074785.
[62] S. Lal, S. U. Rehman, J. H. Shah, T. Meraj, H. T. Rauf, R. Damasevicius, M. A. Mohammed, and K. H. Abdulkareem, "Adversarial attack and defence through adversarial training and feature fusion for diabetic retinopathy recognition," *Sensors*, vol. 21, no. 11, pp. 1424–8220, 2021.

JUMABEK ALIKHANOV (Student Member, IEEE) received the B.S. degree from the Tashkent University of Information Technologies, in 2014, and the M.E. degree from Inha University, in 2017, where he is currently pursuing the Ph.D. degree with the Department of Computer Science and Information Engineering. His research interests include machine learning and its applications on computer vision, natural language processing, sensor data science, and information security.

RHONGHO JANG (Member, IEEE) received the B.S., M.E., and first Ph.D. degrees from Inha University, South Korea, in 2013, 2015, and 2020, respectively, and the Ph.D. degree from the Department of Computer Science, University of Central Florida, in 2020. He is currently an Assistant Professor with the Department of Computer Science, Wayne State University. To date, he has published, as a lead author, several peer-reviewed research papers, including papers in top-tier conferences and premier journals, such as IEEE ICDCS, IEEE INFOCOM, and IEEE TRANSACTIONS ON MOBILE COMPUTING (IEEE TMC). As public services, he is serving as the Publicity Chair for IEEE ICDCS 2021, and served as the Web Chair for ACM CoNEXT 2019 and a Reviewer for IEEE TMC, IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT (IEEE TNSM), IEEE/ACM TRANSACTIONS ON NETWORKING (IEEE ToN), and *ETRI Journal*.

MOHAMMED ABUHAMAD (Member, IEEE) received the master's degree in information technology (artificial intelligence) from the National University of Malaysia, Bangi, Malaysia, in 2013, the first Ph.D. degree in computer science from the University of Central Florida (UCF), in 2020, and the second Ph.D. degree in electrical and computer engineering from Inha University, Incheon, Republic of Korea, in 2020. He is currently an Assistant Professor with the Department of Computer Science, Loyola University Chicago.

DAVID MOHAISEN (Senior Member, IEEE) received the Ph.D. degree from the University of Minnesota, in 2012. He is currently an Associate Professor with the University of Central Florida, where he directs the Security and Analytics Lab (SEAL). Before joining UCF in 2017, he was a Senior Research Scientist with Verisign Labs, from 2012 to 2015, and an Assistant Professor with SUNY Buffalo, from 2015 to 2017. His research interests include networked systems and their security, online privacy, and measurements. He is a Senior Member of ACM (2018). He is also the Editor-in-Chief of *EAI Endorsed Transactions on Security and Safety*, and an Associate Editor of the IEEE TRANSACTIONS ON MOBILE COMPUTING, *Computer Networks* (Elsevier), and *ETRI Journal* (Wiley).

DAEHUN NYANG (Senior Member, IEEE) received the B.Eng. degree in electronic engineering and computer science from the Korea Advanced Institute of Science and Technology (KAIST), in 1994, and the M.S. and Ph.D. degrees in computer science from Yonsei University, South Korea, in 1996 and 2000, respectively. He had been a Senior Researcher at the Electronics and Telecommunications Research Institute (ETRI), South Korea, from 2000 to 2003. From 2003 to 2020, he was a Professor at the Computer Science and Engineering Department, Inha University, South Korea. Since 2020, he has been a Professor at the Cyber Security Department, ELTech Engineering College, Ewha Womans University. He has been the Founding Director of the Information Security Research Laboratory, since 2003. He is currently an Executive Director of the Korean Institute of Information Security and Cryptology. His research interests include all algorithmic aspects of AI security, AI attack, network security, privacy, usable security, biometrics, and their applications to authentication and public key cryptography. His recent interest lies also in network measurement, load balancing algorithm, counting algorithm, caching algorithm, fast hash table, key value store for in-memory DB, random sampling theory, and in-network security (INS) powered by programmable routers.

He is a member of KIISC. He serves as the Editor-in-Chief (EiC) for *KIISC Journal* and a Section Editor (Information Security Section) for *ETRI Journal*.

YOUNGTAE NOH (Member, IEEE) received the B.S. degree in computer science from Chosun University, in 2005, the M.S. degree in information and communication from the Gwangju Institute of Science and Technology (GIST), in 2007, and the Ph.D. degree in computer science from the University of California at Los Angeles (UCLA), in 2012. He is currently an Assistant Professor with the Department of Computer Science and Information Engineering, Inha University. Before joining Inha University, he was a Staff Member with Cisco Systems, until 2014. His research interests include data center networking, wireless networking, future internet, and mobile/pervasive computing.