Cluster-based Sampling Algorithm for Lightweight IoT Intrusion Detection System

Suzan Hajj[1*[0000-0002-5627-2625]], Rayane El Sibai[2[0000-0001-9948-3601]], Adam Barada[3[0000-0003-0228-9517]], Jacques Bou Abdo[4*[0000-0002-3482-9154]], Jacques Demerjian[3], Christophe Guyeux[5[0000-0003-0195-4378]], Abdallah Makhoul[6[0000-0003-0485-097X]] and Dominique Ginhac[1[0000-0002-5911-2010]]

[1] ImViA, Université de Bourgogne Franche-Comté, Dijon, France
[2] Faculty of Sciences, Al Maaref University, Beirut, Lebanon
[3] LaRRIS, Faculty of Sciences, Lebanese University, Fanar, Lebanon
[4] College of Business and Technology, University of Nebraska at Kearney, Kearney, NE, USA
[5] FEMTO-ST Institute, Université de Bourgogne Franche-Comté, CNRS, Besançon, France
[6] FEMTO-ST Institute, Université de Bourgogne Franche-Comté, CNRS, Montbéliard, France
*bouabdoj@unk.edu

Abstract. As IoT is becoming pervasive, the need for securing its weakest nodes, thin nodes operating in untrusted environments, is becoming critical. The evolution of lightweight IDS has been achieved to provide an acceptable level of security for such types of nodes, but a key component is still missing. IDS is step two of a two-step process starting with sampling the traffic into a well-representing sample before being utilized by the IDS for attack detection. The literature is still missing lightweight sampling algorithms to work hand-in-hand with lightweight IDS for achieving the security level intended for thin IoT nodes with minimal stress on their already limited resources. This work is the first to develop a lightweight sampling algorithm catered to perform at low sampling ratios and designed to require limited memory, limited computational resources and enable privacy-preserving collective learning. This work is the first conceptualization of lightweight sampling algorithms and has the potential of significantly reducing resource stress while only resulting in slight distortion to the sample. This compromise would be very beneficial for thin IoT nodes.

Keywords: Internet of Things, Intrusion Detection System, Lightweight Sampling Algorithm.

1. Introduction

Internet traffic, stimulated by the pervasive technological evolution, has witnessed tremendous development in its features. Thus, a new necessity for network-based security systems including Intrusion Detection Systems (IDSs) arises. Intrusion Prevention Systems (IPSs), including firewalls, can be used to prevent the exchange of prohibited messages between end-users and banned endpoints, with limited ability to analyze and detect user activity patterns and abnormal data traffic across the network [1]. With the rise of its complexity, networks are becoming more vulnerable and susceptible to various attacks, making IPSs insufficient for single-handedly providing the desired security level.

The Internet of Things (IoT) is one of those networks that introduce complexities and assumptions that render IDSs critical for maintaining network's security. IoT introduces a wide spectrum of wireless nodes that range from computationally powerful nodes, such as vehicular onboard units, to small energy limited and computationally thin implanted sensors. IoT nodes do not follow the trust structure of traditional networks, thus Firewall, IPS and IDS systems play an important role in securing the central network against unwanted IoT traffic. IoT nodes, especially those with limited computational and energy resources are left unsecure (or at a much lower security level). Implementing a lightweight IDS is thus very effective for elevating the security of IoT nodes [11].

The implementation of a successful IDS comes with many challenges. The complexity of the underlying network, the structure and heterogeneity of the data, and the congestion of the network traffic influence the efficiency and ability of an IDS in detecting malicious behaviors. IDSs' performance is highly affected by how fast it can treat and analyze real-time network traffic; some packets may be skipped in case of highly congested networks.

IDSs need to process and analyze the entirety of the network traffic, including those coming from various sources such as computers and network sensors. The main issue is the abundance of data and the way the traditional IDS operates: It processes each packet in the network and analyzes it to detect any unwanted behavior. Dealing with huge amount of data can be cumbersome, especially when the traditional data processing methods are not scalable. In addition, in the case of anomaly detection, an early warning about the time, nature, and source of the attack should be triggered for a timely response. For this to be achieved, considerable computation, memory and network resources are needed, and this can be too expensive for thin nodes or highly heterogeneous networks to achieve.

To help deal with these issues, many solutions were applied [2]. One solution aims at limiting the packets investigated by the IDS through the implementation of a data sampling technique. The network traffic, coming from various sources, are fused and sampled using a given sampling technique. Only then the obtained sample will be passed to the IDS for further data processing and analysis. This way, the IDS can leverage the limited resources and storage capabilities available. This solution poses a compromise where the IDS needs to maintain a high detection accuracy while dealing with a significantly lower amount of data. The goal is to sample the data while pre-venting the loss of information and keeping the sample as informative as possible. Over the years, many sampling techniques and algorithms were developed and tested for improved detection accuracy, mainly falling under two categories: static and dynamic sampling [3]. Static sampling focuses on sampling the data periodically or randomly with a predefined interval or rule, whereas dynamic or adaptive sampling uses different intervals or rules to sample the data. One major benefit of static sampling is its efficiency in reducing both network bandwidth and storage requirements.

It is difficult to choose a suitable sample of packets from all the traffic, because the IDS only analyzes the sampled data [2]. The attack may not be detected if malicious packets are not selected by the sampling method. As a result, an efficient sampling algorithm must ensure that packets with a malicious payload are sampled. The sampling procedure has been shown to impact, skew, and distort anomaly detection metrics and detection rates in previous research investigations. If a sampling method is to be used, selecting a suitable sample algorithm and sampling interval that provides a decent picture of the overall and original traffic is critical and delicate.

Lightweight IDS for IoT is receiving focus from the research community, as will be shown in section 2, but to the best of our knowledge, efficient sampling algorithms for lightweight IDS is still a gap. This work is the first sampling algorithm designed to help IoT nodes implement lightweight IDS system with minimal impact on its limited resources.

The study of sampling algorithms for lightweight IDSs is still in its early stages and considered a gap. We introduce, in section 2.1, relevant work on lightweight IDS for IoT. In section 2.2, we discuss sampling algorithms for IDS. In section 3, we start bridging the gap between sampling algorithms and lightweights IDS by proposing a lightweight sampling algorithm for IDS. In section 4 we discuss the simulation environment, section 4.1, and the results, section 4.2. Section 5 concludes this work and discusses future trends of lightweight sampling.

2. Related Work

2.1. Lightweight IDS for IoT

Zarpelão et al. [13] surveyed the IDS research for IoT and identified a rising interest in lightweight IDS. The authors identified two tracks that claim to be lightweight. The first is signature-based lightweight IDS which is outside the scope of this work, such as [14]. The second is anomaly-based lightweight IDS which is the focus of this work. Lee et al. [12] detected 6LowPAN attacks by observing nodes' reported energy consumption. Le et al. [15] designed a lightweight IDS by restricting sensing operations to cluster heads and thus freeing the remaining nodes to operate normally. This approach is consistent with Reza et al. [16].

Jan et al. [11] focused on developing computationally lightweight IDS using supervised machine learning-based support vector machine. This is a different strategy that neither limits the IDS to one type of attacks (such as in [12]) nor limits the number of nodes running IDS (such as in [15] [16]). Soe et al. [17] used a different strategy for developing a lightweight anomaly-based IDS by limiting the number of investigated features. The proposed strategy selects the features that result in the highest gain ratio and discard all other features, thus limiting the amount of computation needed. It is worth noting that this strategy risks being oblivious to rare attacks that are only detectable using the discarded features. This approach is consistent with Davahli et al. [19] where they proposed feature selection using hybridization of genetic algorithm (GA) and Grey Wolf Optimizer (GWO). Instead of being selective on the features (such as in [17]) or on nodes (such as in [15] [16]), Sedjelmaci et al. [18] proposed a strategy that is selective on time. The authors proposed a game-theoretic approach for identifying the times where the attacks are most probably going to happen. Only then, the IDS functionality is enabled.

2.2. Sampling algorithms for IDS

Internally deployed Intrusion Detection Systems (IDSs) are now standard practice for preventing and/or limiting both internal and external threats. However, given the limited bandwidth of network lines and the IDSs' limited memory and processing capacity, monitoring and managing the network has become problematic. Sampling could be used to reduce the amount of traffic that needs to be processed as a possible solution to this problem. Current research projects are looking at which sampling policy and parameters give the optimum compromise between IDS performance (response time) and a high attack detection rate in terms of IDS performance (response time). A sampling policy seeks to estimate a metric of interest from a set of data while minimizing processing costs. This is accomplished by picking a "sample" subset of data and estimating the desired statistic from that subset. The sampling approach determines how the data subset is chosen. Packet sampling procedure, in particular, tries to create a sample of data on which future analytic tasks would be performed. The effectiveness of the sample and the precision of the estimation of traffic characteristics are influenced by several factors, the most important of which are the sampling strategy and the sampling rate. The most challenging challenge, given the original collection of packets, is to choose the appropriate sampling policy and parameters.

Packet sampling is a type of data sampling approach that uses packets as the basic unit of analysis. As a result, all of the observed packets are considered the original data set, while the selected packets comprise the sample. The sampling interval, commonly known as the sampling ratio, has a significant impact on the target sample size. However, the sample size acquired by some sampling algorithms may differ from the required sample size. There are three sorts of sampling decisions for a packet: count-based, time-based, and content-based. The sampling choice of a packet in a count-based sampling strategy is made based on its position in a stream of packets. The sampling decision in time-based sampling algorithms is dependent on the packet arrival time. Finally, content-based sampling methods focus their sample decisions on the content of the packet. Filtering algorithms are another term for content-based sampling procedures, which are beyond the scope of this paper.

The area of sampling network traffic has received a lot of attention from the research community in recent years, resulting in a slew of new works and benchmarking publications. Several studies have the effects of data sampling in this context. In their study, Mai et al. [4] looked at the impact of sampling high-speed IP-backbone network traffic on intrusion detection outcomes, particularly port scans and volume anomaly detection. To sample the traffic packets, various sampling algorithms were employed. The authors of [5, 6] looked into how packet sampling affected anomaly detection findings. The accuracy of the “Autonomous Algorithm for Traffic Anomaly Characterization” detector in detecting DDoS attacks over sampled traffic was tested by Roudiere et al. [7]. To sample the traffic, various sampling policies were utilized. The influence of traffic sampling on anomaly identification was investigated by Bartos et al. [8], who presented a new adaptive flow-level sampling algorithm to improve the sampling process' accuracy. Silva et al. [9] proposed a framework for assessing the effects of packet sampling. They analyzed each sampling algorithm's effectiveness and proposed a set of metrics for evaluating each sampling technique's ability to produce a representative sample of the original traffic. Brauckhoff et al. [10] evaluated the accuracy of alternative anomaly detection and data sampling algorithms using traces containing the Blaster worm. et al. [10] evaluated the accuracy of alternative anomaly detection and data sampling algorithms using traces containing the Blaster worm.

An extended related work can be seen in our survey [2] and benchmarking [3] where we examined all data sampling strategies, their influence on detecting various attacks, and the behavior and robustness of features under various sampling strategies. It also investigated how network features estimation differs depending on the sampling method, sample size, and other factors, and how this affects statistical inference from these data.

3. Cluster-based Sampling Algorithm

To reduce the sampling error and ensure a high level of sample representation where the smallest, extreme, and/or rare subgroups of the data are present, we propose in this section a new Cluster-based sampling algorithm. The algorithm classifies the data into different clusters and builds and maintains a sample for each cluster. Clustering algorithms aim to find new emerging patterns in the data and to detect any changes in the patterns and data distribution. Ideally, for a sample to be representative of the entire data stream, the sample and initial stream must be closed. The proposed algorithm manages the insertions of the data in the samples so that the difference between the sample and the corresponding cluster is minimal. That is, the algorithm ensures that all the subgroups within the data are present in the final sample.

With the most known sampling algorithms, data are randomly sampled from the stream. The samples are built fast, but the arbitrary selection and deletion of items may lead to high sampling error. Our proposed approach can be considered as an alternative that aims to reduce the sampling error due to the data variance. The idea of pre-

clustering before applying the sampling allows taking into account the prior data distribution when making the selection decisions. When items close together in the data stream have similar values, they will be added to the same cluster. This cluster tends to be homogeneous so that a small sample from this cluster contains a large amount of information about all of the items in the cluster.

Our proposed sampling method consists of dividing the data stream into separated groups, called clusters, using the k-means algorithm, and then, build and maintain a sample from each cluster as shown in Figure 1. To guarantee that the final sample contains exactly items representative of every group of the initial data stream, the size of each cluster sample is kept proportional to the cluster size. The final sample is constructed by combining all the clusters' samples. From the data in the first window of the stream, *i* initial clusters are constructed.

To ensure that the sample is as close as possible to the original cluster, the proposed algorithm maintains a sample of a size proportional to the cluster size and modifies it as more stream items arrive, as shown in Figure 2. That is, for a sampling rate equal to *k/n*, *k* items will be sampled from each cluster. The first *k* items of the stream are classified using the k-means algorithm and serve as a learning base. Then, for each cluster, a sample of size k_j is built and maintained, k_j being the cluster j size constructed from the initial learning base, $\sum_{j=1}^{i} k_j = k$. From each cluster, k_j items are selected, each with a probability equal to $\left|1 - \frac{e_i}{center(C_j)}\right|$. Then, the initial items in each cluster which are selected and added to the samples are ranked by the "remove-from-sample" principle, where the sampling error is calculated by removing the item from the sample, as shown in Figure 3 and Procedure CLUSTER_BASED_SAMPLING. Higher ranks are assigned to the removal of the items which leads to lower sampling error.

When new items arrive, they will be processed in sequential order. Each new item will be added to one of the clusters based on the Euclidean distance between the item e_i and the center of each cluster. After fitting the item to one of the clusters, a decision will be made whether to add it to the corresponding cluster sample. This is done by comparing the sampling error of the cluster sample with and without the item. If not adding the item to the cluster sample leads to a higher sampling error, the new item will be sampled with a probability equal to $\left|1 - \frac{e_i}{center(C_j)}\right|$. On the other hand, if adding the item to the cluster sample leads to a higher sampling error, the item will be rejected and not added to the sample. To keep the cluster sample size fixed, an item must be removed from the sample when the featured cluster size exceeds k_j. In this case, the item having the highest-ranked value in the current sample will be deleted. Ideally, all items in the current cluster sample should be re-ranked after a new item is added to the sample. This is done by recalculating the sampling error using the "remove-from-sample" principle for all items in the current cluster sample. A trade-off between the sampling precision and needed computing resources is needed here. Thus, re-ranking is only done after $x\%$ of a cluster's items are changed, where *x* depicts $(number\ of\ replaced\ items/cluster\ size) * 100$.

Let m be the original mean calculated from the original data in a given cluster. The empirical mean $\overline{X}$ of the cluster sample is an unbiased estimator of m, calculated as follows: $\overline{X} = \frac{1}{k}\Sigma\, e_i$ where k is the sample size, and e_i is the value of the item at index i in the sample. The sampling error with/without the item is calculated as follows: $error = \left|\frac{m - \overline{X}}{m}\right| \cdot 100\%$

Three parameters impact the sampling accuracy: (1) the sampling ratio k/n, (2) change percentage x. It is possible to sample the data with a sampling ratio equal to *k/n* while using different *k* and *n* values, such that: $p = \frac{k}{n} = \frac{x \cdot k'}{x \cdot n'}$ where *x* is a positive integer.

Table 1 shows the conceptual idea of ranking. For instance, item 10 is ranked highest because its removal produces the minimum sampling error. Similarly, item 17 is ranked lowest because its removal produces the highest sampling error. Thus, it is the best candidate to be replaced by future incoming items.

Table 1 Conceptual items ranking within a cluster

Item Index	Sampling error	Rank
10	3.61	1
11	...	2
12	...	3
17	9.62	4

Thin IoT nodes have three main concerns that should be the focus of any lightweight sampling algorithm. The concerns are:

- **Limited memory**: Dynamic sampling algorithms have increasing sample size [2] and this exceeds thin nodes' capabilities. Static sampling algorithms have fixed sample size. In the current version of the proposed algorithm, a fixed number of elements are stored in clusters making the stored sample size similar to static sampling algorithms. As will be discussed in section 5, we are working on eliminating all the stored elements and only keep the cluster definitions. That will significantly drop the amount of memory needed into less than 1% of the current standard.
- **Limited computation/energy**: The proposed algorithm can be configured for running with significantly less computation using the Change Ratio (CR). Lower CR values result in lesser need for computational resources. Computational resources vs. sample distortedness should be optimized.
- **Privacy-preserving collective learning**: The ability for IoT nodes to share their knowledge about the network traffic is a significant enhancement to the security of the whole system and this allows for a significant drop in computational and energy resources needed. This is assuming that the exchange of information is privacy-preserving. As will be shown in section 5, we are experimenting this capability by relying on cluster definitions.

4. Simulation

4.1. Simulation Environment

Our algorithm is implemented using Python to test its efficacity and measure its accuracy. The code is accessible through GitHub [20]. In this simulation parameters are defined in table 2. The algorithm has been tested against the NSL-KDD test dataset [21]. Clusters' compactness is measured using the silhouette score [22].

Table 2 Simulation parameters

Parameter	Range of values
Change rate	0.5, 0.75, 0.9
Learning base size	0.1, 0.3, 0.5, 0.7, 0.9 (× Dataset size)
Number of clusters	2, 5 (2, 3 and 6 tested but not reported as discussed in section 4.2)

The specifications of our machine are RAM: 12GB, System disk: 108GB, and processor: 2.20GHz Intel Xeon CPU. We measured all the dataset features, but for space limitations, we will show the results of four features as discussed in section 4.2.

4.2. Simulation Results

The selection of cluster sizes 2 and 5, shown in table 2, is not arbitrary. Cluster sizes 3, 4 and 6 have been tested, but not reported due to space limitation. 2 and 5 clusters performed consistently better on mean, median and standard deviation. 2 clusters group packets into normal or attack, while 5 clusters group packets into normal or one of the four attack types listed in table 3. To validate our observations, we measured the clustering performance using silhouette score for each of configurations presented in table 2. The silhouette measurements are shown in Figure 4.

As shown in Figure 4, the cluster fitness is increasing with the increase in the number of clusters, but this does not predict performance in maintaining the stream's mean, median and standard deviation within the sample. For this reason, cluster fitness is not a good metric for selecting the appropriate configuration of the IDS. Throughout the remaining of this work, we are going to report the values for clusters sizes 2 and 5 only as they performed the best. The x-axis in Figure 4 is *(Learning base, Change rate)*. The y-axis is the silhouette score.

The results shown in this work are presented in consistence with the format used in [3], including mean, median, standard deviation, and Overall Statistic (OS) value of the selected dataset features that are most representative of network attacks shown in table 3. It is worth noting that our proposed lightweight sampling algorithm will be compared to non-lightweight sampling algorithms, surveyed in section 2.2, due to it being the first of its category to the best of our knowledge.

```
procedure CLUSTER_BASED_SAMPLING {n: initial learning
base size}
Initialization: Swj is the initial sample of cluster
Cj, at the beginning S1 = ∅, k is the sample or
learning base size, x is the change ratio and N is the
number of clusters.
//starting pseudo-code presented in figure 2
for i ∈ [1, k] do
   for each item ei in the learning base of size k do
    end for
end for
//ending pseudo-code presented in figure 2
//starting pseudo-code presented in figure 3
While a new item ei is received, (i<n) do
   i <- i + 1 {i is the index of e}
   Add ei to the corresponding cluster Cj
   CalculateError(ei, Sj) {Compute the sampling error
with and without adding ei to Sj}
   if samplingErrorWithSampling <
samplingErrorWithoutSampling then
      Add ei to Sj with a probability |1- ei/center(Cj)|
      Cj.change_count <- Cj.change_count + 1
      if(Sj.size() > k/n) then
        Remove the item having the highest rank from Sj
      end if
   end if
   if(Sj.size()* x < Cj.change_count) then
      UpdateRankTable();
   end if
end while
return Sj
end procedure=0
//ending pseudo-code presented in figure 3
```

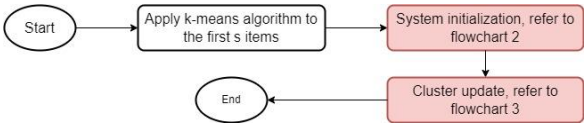

Figure 1 Proposed algorithm

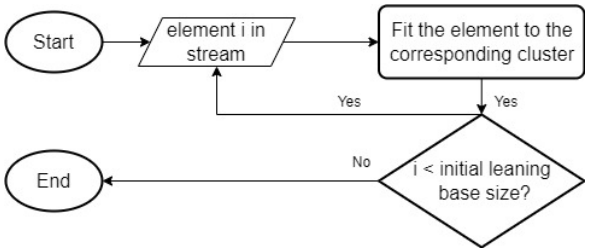

Figure 2 Item admission

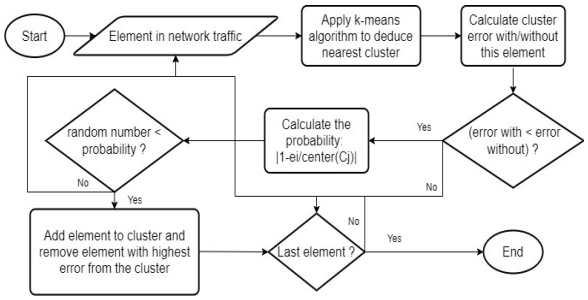

Figure 3 Cluster management

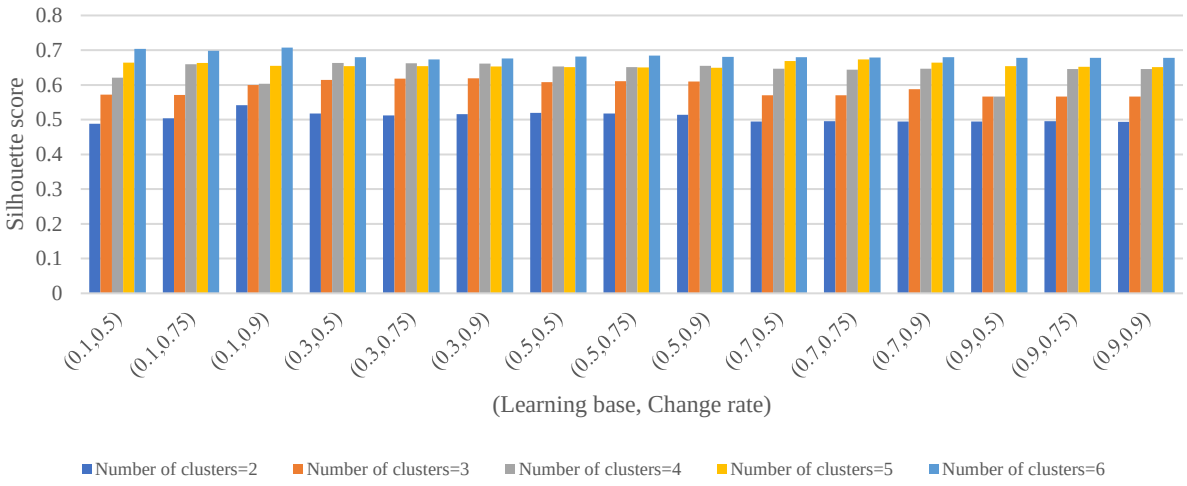

Figure 4 Cluster fitness

Table 3 Features vs. attack types in dataset

Attack	Feature Names	Features
DoS	Source bytes, land, wrong fragment	5,7,8
Probe	Source bytes, srv error rate, diff srv rate, src port rate	5,28,30,36
R2L	Dest bytes, failed logins, count, dst host error rate	6, 11,23,39
U2R	Root shell, srv count, src port rate	14, 24,36

Figure 5 shows the mean of feature 5 in the sample generated by each sampling algorithm. The red dotted line represents the dataset's original mean for feature 5. The closer the sample's mean to the that of the dataset the better. Similarly, Figure 6 shows the standard deviation for feature 5 with the red dotted line representing the dataset's standard deviation for feature 5.

It can be seen in Figures 5, 6 and 7 that the most accurate algorithms at high sampling ratios/learning base sizes, ≥ 0.7, are systematic and deterministic sampling algorithms. For lower sampling ratios/learning base sizes, those same algorithms are among the least accurate. Those observations are consistent with [3]. For low sampling rates, SRS offline, stratified and the proposed cluster-based sampling algorithm (with 2 clusters and 90% change ratio) are the consistently good performers (least distorted). Thin IoT nodes with limited resources benefit from low sampling ratios/learning base sizes as this significantly decreases the computational load of both sampling and IDS.

The results of feature 5 are consistent with those of feature 6, shown in Figures 8, 9 and 10. Feature 6 shows even better performance for the proposed cluster-based sampling algorithm (with 2 clusters and 90% change ratio) for lower sampling ratios than SRS offline, stratified. The proposed cluster-based sampling algorithm (with 2 clusters and 90% change ratio) resulted in least distortion for two sampling ratios/learning base sizes 0.3 and 0.5.

The results for feature 14, shown in figures 11, 12 and 13, are not as consistent with the other features. The proposed cluster-based sampling algorithm, all configurations, resulted in the least distortion for sampling ratio/learning base sizes = 0.3 only. The performance of the proposed algorithm is average for all other sampling ratios/learning base sizes.

The results for feature 28 are consistent with that of features 5 and 6. The proposed cluster-based sampling algorithm (with 2 clusters and 90% change ratio) achieved the lowest distortion for sampling ratios/learning base sizes 0.1, 0.3 and 0.5 as shown in figures 14, 15 and 16. The results of the remaining features are consistent with the ones presented but will not be shown due to space limitations.

5. Conclusions and Future Work

Thin IoT nodes benefit from lesser stress on their already limited resources, but that should not result in lower security than intended, especially that IoT nodes operate in untrusted environments. The literature covers sampling algorithms for traditional IDSs and lightweight IDSs for IoT, but lightweight sampling algorithms for IDSs operation in IoT is still a gap. This work is the first, to the best of our knowledge, to touch this gap and develop a sampling algorithm that requires minimal resources and achieves good results for low sampling ratios, which is the type of configuration suitable for thin IoT nodes.

The uniqueness of this proposal is the ability to describe the sample in a privacy-preserving manner and this allows the exchange of information of node for federated learning. This protocol still requires upgrades to be able to reach its full potential which are (1% memory utilization compared to other algorithms, configurable computational resources, privacy-preserving collective learning). This version of the protocol succeeded in performing the second goal (configurable computational resources) while achieving comparable and sometimes best performance and least distortion to the generated sample.

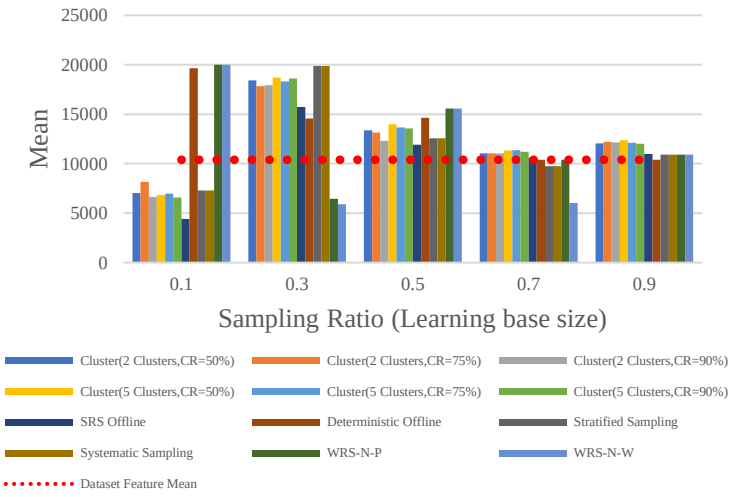

Figure 5 Mean of feature 5 per sampling algorithm

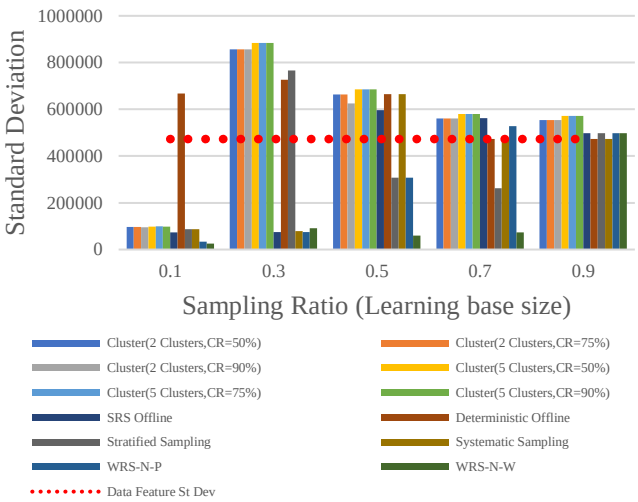

Figure 6 St dev of feature 5 per sampling algorithm

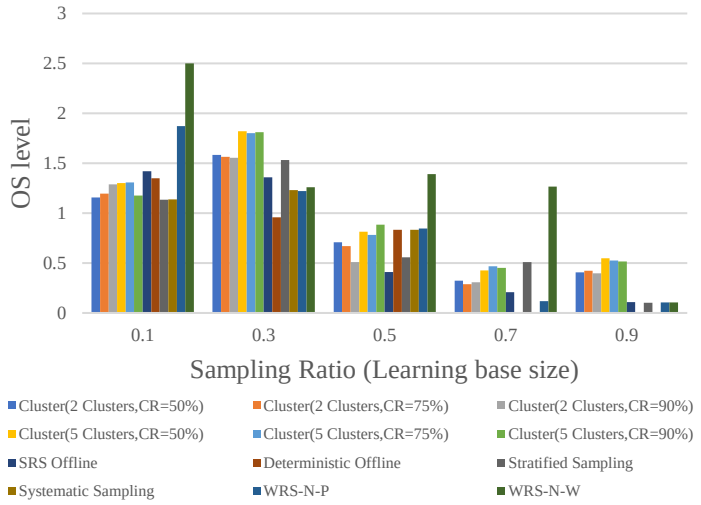

Figure 7 OS level of feature 5 per sampling algorithm

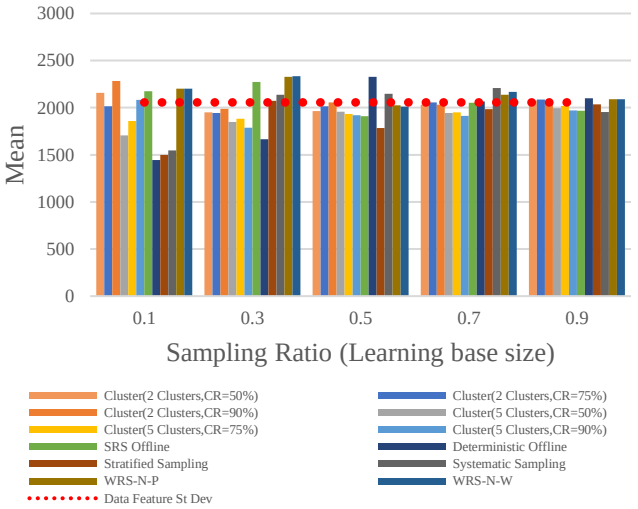

Figure 8 Mean of feature 6 per sampling algorithm

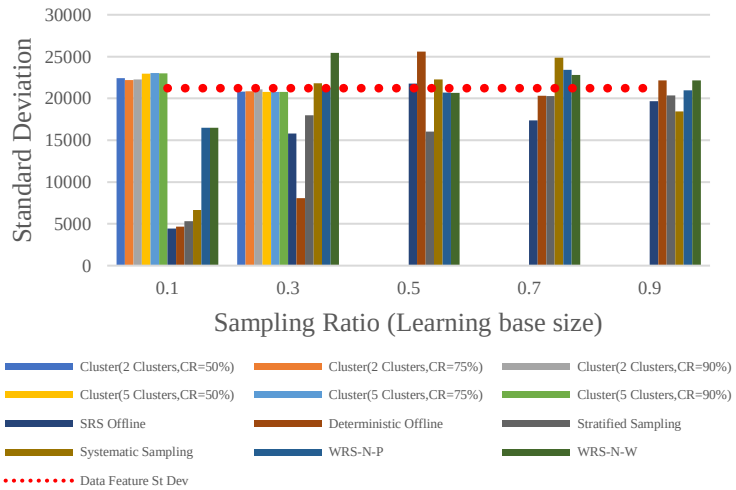

Figure 9 St dev of feature 6 per sampling algorithm

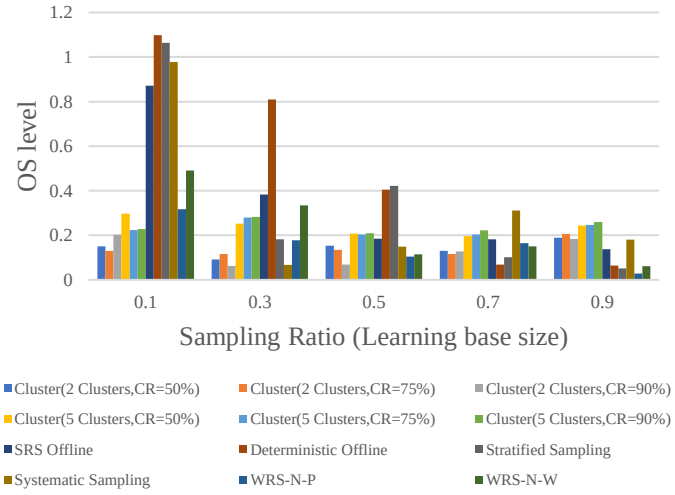

Figure 10 OS level of feature 6 per sampling algorithm

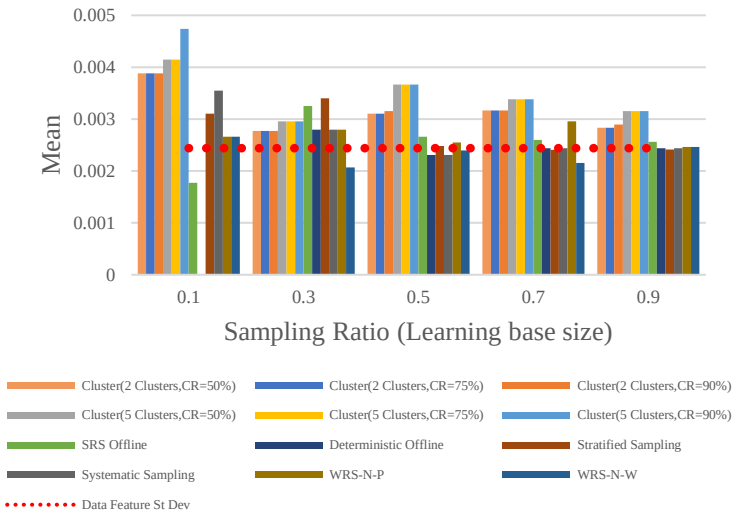

Figure 11 Mean of feature 14 per sampling algorithm

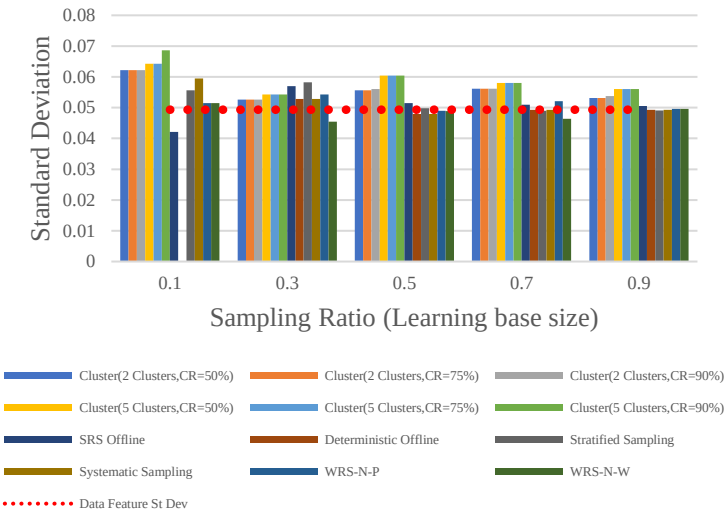

Figure 12 St dev of feature 14 per sampling algorithm

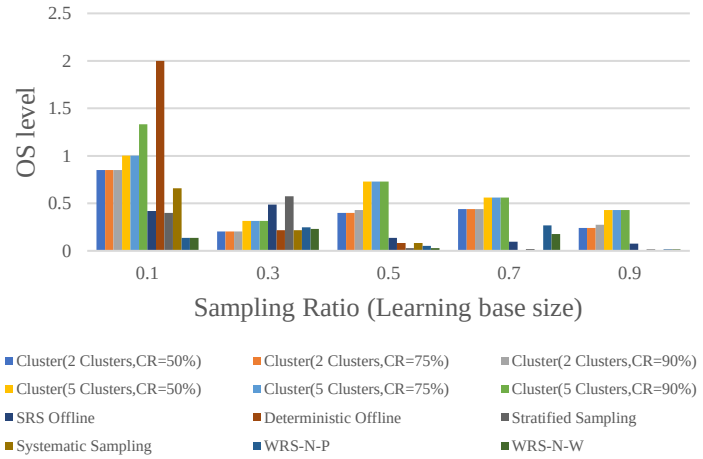

Figure 13 OS level of feature 14 per sampling algorithm

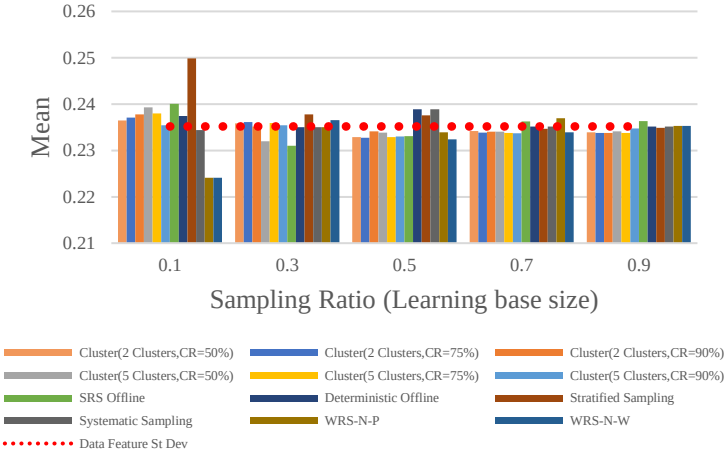

Figure 14 Mean of feature 28 per sampling algorithm

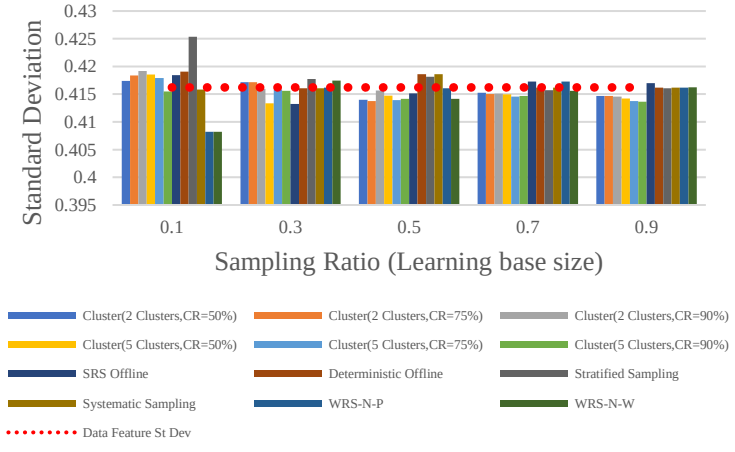

Figure 15 St dev of feature 28 per sampling algorithm

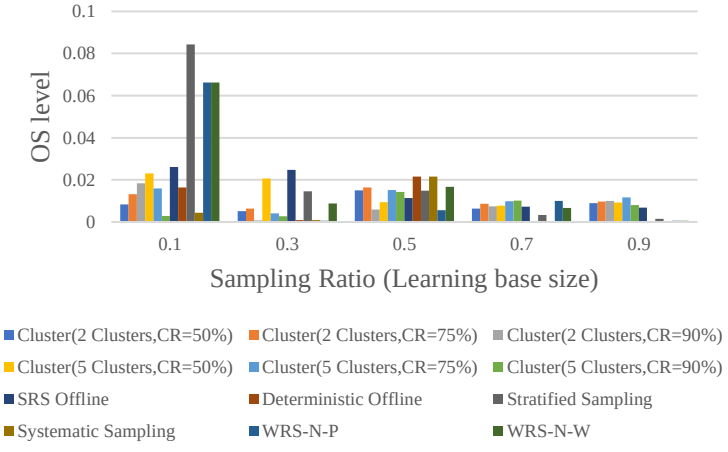

Figure 16 OS level of feature 28 per sampling algorithm

References

1. Tuck, Nathan, Timothy Sherwood, Brad Calder, and George Varghese. "Deterministic memory-efficient string matching algorithms for intrusion detection." In IEEE INFOCOM 2004, vol. 4, pp. 2628-2639. IEEE, 2004.
2. Hajj, Suzan, Rayane El Sibai, Jacques Bou Abdo, Jacques Demerjian, Abdallah Makhoul, and Christophe Guyeux. "Anomaly based intrusion detection systems: The requirements, methods, measurements, and datasets." Transactions on Emerging Telecommunications Technologies 32, no. 4 (2021): e4240.
3. Hajj, Suzan, Rayane El Sibai, Jacques Bou Abdo, Jacques Demerjian, Christophe Guyeux, Abdallah Makhoul, and Dominique Ginhac. "A critical review on the implementation of static data sampling techniques to detect network attacks." IEEE Access (2021).
4. Mai, Jianning, Chen-Nee Chuah, Ashwin Sridharan, Tao Ye, and Hui Zang. "Is sampled data sufficient for anomaly detection?." In Proceedings of the 6th ACM SIGCOMM conference on Internet measurement, pp. 165-176. 2006.
5. Pescapé, Antonio, Dario Rossi, Davide Tammaro, and Silvio Valenti. "On the impact of sampling on traffic monitoring and analysis." In 2010 22nd International Teletraffic Congress (lTC 22), pp. 1-8. IEEE, 2010.
6. Zhang, Hu, Jun Liu, Wenli Zhou, and Shuo Zhang. "Sampling method in traffic logs analyzing." In 2016 8th international conference on intelligent human-machine systems and cybernetics (IHMSC), vol. 1, pp. 554-558. IEEE, 2016.
7. Roudiere, Gilles, and Philippe Owezarski. "Evaluating the Impact of Traffic Sampling on AATAC's DDoS Detection." In Proceedings of the 2018 Workshop on Traffic Measurements for Cybersecurity, pp. 27-32. 2018..
8. Bartos, Karel, Martin Rehak, and Vojtech Krmicek. "Optimizing flow sampling for network anomaly detection." In 2011 7th international wireless communications and mobile computing conference, pp. 1304-1309. IEEE, 2011.
9. Silva, João Marco C., Paulo Carvalho, and Solange Rito Lima. "A modular sampling framework for flexible traffic analysis." In 2015 23rd International Conference on Software, Telecommunications and Computer Networks (SoftCOM), pp. 200-204. IEEE, 2015.
10. Brauckhoff, Daniela, Bernhard Tellenbach, Arno Wagner, Martin May, and Anukool Lakhina. "Impact of packet sampling on anomaly detection metrics." In Proceedings of the 6th ACM SIGCOMM conference on Internet measurement, pp. 159-164. 2006.
11. Jan, Sana Ullah, Saeed Ahmed, Vladimir Shakhov, and Insoo Koo. "Toward a lightweight intrusion detection system for the internet of things." IEEE Access 7 (2019): 42450-42471.
12. Lee, Tsung-Han, Chih-Hao Wen, Lin-Huang Chang, Hung-Shiou Chiang, and Ming-Chun Hsieh. "A lightweight intrusion detection scheme based on energy consumption analysis in 6LowPAN." In Advanced technologies, embedded and multimedia for human-centric computing, pp. 1205-1213. Springer, Dordrecht, 2014.
13. Zarpelão, Bruno Bogaz, Rodrigo Sanches Miani, Cláudio Toshio Kawakani, and Sean Carlisto de Alvarenga. "A survey of intrusion detection in Internet of Things." Journal of Network and Computer Applications 84 (2017): 25-37.
14. Oh, Doohwan, Deokho Kim, and Won Woo Ro. "A malicious pattern detection engine for embedded security systems in the Internet of Things." Sensors 14, no. 12 (2014): 24188-24211.
15. Le, Anhtuan, Jonathan Loo, Kok Keong Chai, and Mahdi Aiash. "A specification-based IDS for detecting attacks on RPL-based network topology." Information 7, no. 2 (2016): 25.
16. Raza, Shahid, Linus Wallgren, and Thiemo Voigt. "SVELTE: Real-time intrusion detection in the Internet of Things." Ad hoc networks 11, no. 8 (2013): 2661-2674.
17. Soe, Yan Naung, Yaokai Feng, Paulus Insap Santosa, Rudy Hartanto, and Kouichi Sakurai. "Towards a lightweight detection system for cyber attacks in the IoT environment using corresponding features." Electronics 9, no. 1 (2020): 144.
18. Sedjelmaci, Hichem, Sidi Mohammed Senouci, and Mohamad Al-Bahri. "A lightweight anomaly detection technique for low-resource IoT devices: A game-theoretic methodology." In 2016 IEEE international conference on communications (ICC), pp. 1-6. IEEE, 2016.
19. Davahli, Azam, Mahboubeh Shamsi, and Golnoush Abaei. "Hybridizing genetic algorithm and grey wolf optimizer to advance an intelligent and lightweight intrusion detection system for IoT wireless networks." Journal of Ambient Intelligence and Humanized Computing 11, no. 11 (2020): 5581-5609.
20. Barada, Adam. “Cluster-based-sampling.” https://github.com/AdamBarada/Cluster-based-sampling, (2022).
21. Tavallaee, Mahbod, Ebrahim Bagheri, Wei Lu, and Ali A. Ghorbani. "A detailed analysis of the KDD CUP 99 data set." In 2009 IEEE symposium on computational intelligence for security and defense applications, pp. 1-6. IEEE, 2009.
22. Pedregosa, Fabian, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.