Training and Validating of Advanced Flow-Based Network Traffic Classifiers under Real-World Conditions

1st Sebastian Karius
Institute for Computer Science
MLU Halle-Wittenberg
Halle (Saale), Germany
sebastian.karius@informatik.uni-halle.de

2nd Mandy Knöchel
Institute for Computer Science
MLU Halle-Wittenberg
Halle (Saale), Germany
mandy.knoechel@informatik.uni-halle.de

3rd Sandro Wefel
Institute for Computer Science
MLU Halle-Wittenberg
Halle (Saale), Germany
sandro.wefel@informatik.uni-halle.de

***Abstract*—Robust network traffic classification (NTC) is an essential component of intrusion detection and intrusion prevention systems as well as quality of service applications. Modern NTCs use flow-based methods to either support or replace single-packet-based methods, since network traffic is usually encrypted. The flow-based methods are often implemented using neural networks. These neural networks are usually trained and tested using the same dataset. We propose a new testing method in which the data used to test the neural network occurs at a later time than the data used to train the neural network. For this, we used a dataset recorded by a honeypod that captured attacks, particularly on the SSH service, as well as authorized logins. The data was recorded over a period of over two years. Past data was used for training and recent data was used for testing, with a large time gap between the last datapoint of the training data and the first datapoint of the testing data. We show that existing neural network models developed for NTC are able to classify the attacks in our test scenario equally well, for authorized accesses the quality is worse. This shows that the proposed learning method for our chosen models is sufficient to robustly classify future network traffic in the scenarios tested here.**

***Index Terms*—network traffic classification, ntc, real-world, flow-based**

I. Introduction

Network traffic security is commonly achieved using Intrusion Detection Systems (IDS) or Intrusion Prevention Systems (IPS). They are centrally installed network monitoring systems that are often integrated directly into routers or similar network devices. Their purpose is to detect or block malicious traffic to protect the attached network nodes. The problem is that the network traffic is usually encrypted, so to our knowledge it is impossible to examine the content of the network packets in real time and thus detect malicious traffic. To avoid breaking end-to-end encryption, flow-based network traffic classifiers are used to detect malicious or unwanted network traffic, with the most efficient ones working on the basis of neural networks [1] that have to be trained before they can classify (live) network traffic.

A dataset with suitable sample data is needed to train the neural network accordingly. In order to determine the quality of the learned classifier, test data is required in addition to the training data. This test data is usually taken from the same initial dataset that was used to train the network, so that a part of the initial dataset is used for training and another, often much smaller part, is used for testing.

We argue that this approach of using data that was recorded more or less at the same time for learning and testing is questionable. Since the data obtained within a limited time period may have similar properties which are beneficial for classification. In a real scenario, however, the classifier must be able to handle new data that arise later in time. New variants of software and protocols, new attack methods, scanners and scripts or changes in the connected network can have an impact on the data. To determine the actual performance of the classifier, it is necessary to test with data that occurred at a (much) later time in the network than the training data.

In this paper, we investigate the suitability of current network traffic classifiers for classifying future network traffic in real-world conditions by applying it to data recorded over a very large period of time. We use data from the distant past (from 2 to 3 years ago) to train the methods, and data from the recent past to validate the methods trained on this old data. In addition, time slots are used to enable real-time classification.

II. Related Work

There are many approaches that apply neural networks to Network Traffic Classification (NTC). However, they differ from our approach in the way they are validated.

Bergmeir and Benítez [2] showed that using the last blocks of time series data for validating machine learning methods for time series forecasting (which is definitely a different problem from data classification) has no advantage over cross validation in terms of the robustness of the trained model. This shows that randomly splitting the data has no disadvantage over simply splitting it into old and new data for training and testing.

Moore et. al. [3] recorded real-world network traffic in ten half-hour segments over a 24-hour period. This dataset was used by Malik et. al. [4] to train and validate a neural network. In their approach, precomputed flow features were used as

data points. The classifier achieved an overall accuracy of 96%. Although the records are already divided into temporal sections and therefore would be suitable to train on older data than to validate, they have not taken advantage of this. Instead, training, validation and testing were performed within a single temporal section only.

Another approach to the problem of changing network traffic is to build live-learning traffic classifiers [5], [6]. The idea is to collect unidentifiable flows and present them to the operator or admin, who then has to manually label the traffic. After labeling, the classifier is trained with the new data. This approach requires on-site learning and thus additional computing and human resources at the location where the traffic is classified. To estimate the effort for this, an estimation of how often the classifier needs to be adapted is required.

III. Our Approach

A. Dataset

To determine the impact of the temporal distance of the training data from the data to be classified, we classify data taken at different temporal distances from the training data with the same trained model.

The data we use was recorded over several years in a honeypot used to capture attacks via Secure Shell (SSH). Due to the original purpose of the honeypot, only SSH traffic, meaning Transmission Control Protocol (TCP) traffic on server port 22, is recorded. Since the honeypot was operated over a long period of time, network traffic with attack patterns and methods was recorded that changed over time.

The captured network traffic can be divided into four classes: brute force attacks, successful logins (without interaction on the server or opening a tunnel), successful logins with subsequent interaction on the server, and successful logins with subsequent opening of a tunnel. Since brute force attacks just consist of a series of unsuccessful logins, these four classes can be combined into two more general classes: brute force attacks and intrusion. The four classes and the time interval of recording are shown in Table I. Combining the classes is also beneficial regarding the distribution of the recorded data points since the classes brute force attacks and successful logins with subsequent opening of a tunnel contain about 10 times more data points than the other two classes. For the training of a neural network it is more suitable that the classes have equal amounts of data points. The classification was performed with the decrypted traffic so that the network traffic was perfectly classified. The classification is stored in a database where for each flow of the recorded network traffic the relevant information is stored. Although this already is a solution to the classification, it cannot be used for an IDS or IPS, because the network traffic can only be decrypted on the end devices.

The dataset contains both the raw recorded network data as well as already calculated network flows. However, since we need special network flows, they are calculated directly from the raw data. Hereafter, we are able to compute features from the flows as well as using raw packets as data points for the neural network. So the various models from the literature can be used.

For the validation of the models we had to search the dataset for periods where both brute force attacks and intrusion traffic is contained. In our recorded network traffic data we found corresponding sections in the periods 2021-07-28 00:00 - 2021-07-29 12:00 (I), 2021-08-18 12:00 - 2021-08-20 12:00 (II) and 2021-08-25 00:00 - 2021-08-25 12:00 (III), i.e., 63, 84 and 91 days or approximately 2, 2.5, and 3 months after the last data point in the recording of the training data. We were not free in choosing those periods, because the dataset is a real-world recording, network traffic of the different classes is not evenly distributed.

For the test data the raw network traffic data is ideal, because it is stored as it appeared at the server and therefore is most similar to the applications in IDS or IPS systems.

B. Preprocessing

The preprocessing step includes creating the flows and, in the case of pre-calculated features, the calculation of these features. The generation of flows is optionally preceded by a step for the generation of time slots.

The network traffic from the dataset is first divided into flows consisting of 100 packets each. If there are not enough packets in the flow (which are calculated from packets within a slot), it is padded with zeros, as usually done; if there are more packets, they are truncated to 100. Since mainly the type of login is to be classified and a typical SSH login duration is about 20-30 packets, 100 packets are sufficient to determine the type of login and the type of action performed on the server immediately after the login.

For real-time or near real-time applications, it is often not feasible to wait for a flow to complete before classifying it. IPS applications, for example, need information about the flow as soon as possible to block malicious network traffic. IDS, on the other hand, only need to notify of a possible intrusion so that the flow can be classified when it is complete, regardless of whether it is a thread. Bursts or time slots can be used to enable real-time classification. A burst *logically divides the network traffic into discrete, manageable portions by grouping packets until no new packets arrive within a certain amount of time, the so-called burst threshold* [7]. The problem with bursts is that if the network traffic is recorded at a router to which a network with multiple clients and servers is connected. Here, the constant stream of packets could result in no burst being completed and, consequently, no classification or severely delayed classification. To solve this problem of a never idle system, we put hard limits on the number of packets and the amount of time a single burst is allowed to have. We refer to these as time slots to distinguish them. The advantage of time slots is that they end predictably and therefore provide a more predictable classification delay.

One consequence of the use of time slots is that flows are split. This occurs whenever packets of a flow span across two or more time slots. The original flow is thereby split into several flows, most often only two, that is calculated

TABLE I
DATE OF THE FIRST AND LAST FLOW FROM THE RESPECTIVE CLASSES OF FLOWS IN THE TRAINING DATASET FOR EACH OF THE FOUR CLASSES.

Class	Specific class	Start	End
	Brute force attacks	2019-01-08	2021-04-30
Intrusion	Successful login	2017-07-25	2021-05-24
Intrusion	Successful login with subsequent interaction	2017-07-23	2021-05-01
Intrusion	Successful login with subsequent tunneling	2017-08-31	2021-05-26

within the time slots (Figure I). Since those split flows do not necessarily contain the information to decide whether a flows was a brute force attack or a successful login, those flows should be classified as a separate "split"-class. The neural network model predicts only the given classes, so the just mentioned "split"-class must additionally be presented to the neural network in the training data. This was achieved by artificially creating split flows by randomly splitting random sampled flows.

Another disadvantage is, that low throughput applications are poorly represented because there are only a few packets in a slot. Depending on the capacity of a slot, the classification of short flows may be delayed because time slots, unlike flows or packets, are not padded. For example, a single SSH login attempt requires about 20 packets; if the slot has a capacity of 100 packets, the classification can be delayed by 80 packets in the worst case.

Despite the disadvantages, the use of slots is preferable because of the predictable termination. As noted above, early classification of long flows is more relevant to IPS than undelayed classification of short flows. The delay can be controlled by the slot generation parameter explained next.

As flows with low throughput are poorly represented by the time slots, they are not considered in this work. This includes, for example, flows that only transmit a keep-alive packet or a similar status message.

We used two different parameter sets for the time slot generation. The first slot parameter set represents a slow detection where fewer errors are tolerated, the second parameter set represents a faster but less accurate detection. For this purpose, the first slot parameter set was limited to a capacity of 2000 packets and 10 seconds (A), and the second one is limited to 100 packets and 5 seconds (B). The selection of the parameters depends on the captured network traffic and the maximum acceptable delay of the classification. For the captured network, the total throughput, the throughput and number of attached network nodes and the number of communicating services are most relevant. The two sets we have chosen are intended to represent two sets of parameters for different objectives. Set A aims at less splitted flows and thus at a more robust classification. Set B aims at a fast classification that tolerates a less robust classification in return.

Depending on the neural network model used, the packets themselves are either represented as pre-calculated features or as a byte vector. As stated in the next section, for the pre-calculated features, the features proposed by Lopez-Martin et al. [8] are used except for the window size. In the case of the byte vector representation, since the packets are of variable length, the packets are zero-padded or truncated to 1500 bytes. For better numerical stability each byte is divided by 255 to get a number in the range $[0, 1]$. To preserve the temporal relation between the packets, the byte vector representation is extended by a timestamp value, which was recorded and stored in the pcap files.

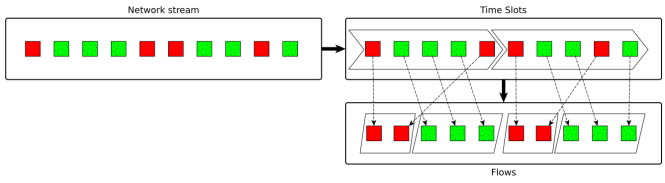

Fig. 1. Creation of time slots and flows. Boxes represent network packets. Packets of the same color belong to the same original flow. Because of the time slots, a single original flow can be split into multiple flows.

C. Neural Network Model

Since we used the features proposed by Lopez-Martin et al. [8], we also used the neural network model proposed in their paper. The paper describes multiple neural networks which are variations of a combination of Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs). We decided to use the neural network "CNN+RNN-1", which is not the best performing in the paper, but was the best performing in our tests, which is most likely due to the different datasets used.

For the byte vector representation data we use the "1D-CNN for traffic characterization" neural network model proposed by Lotfollahi et al. [9].

Since we used a smaller dataset, we had to reduce the degrees of freedom of the neural network models to prevent overfitting. For this purpose, we reduced the size of the fully connected layers.

The concrete used models are shown in Table II and III. NoF stands for number of features, Kl is the used kernel with the dimensions in the order of the dimensions of the data, Str stands for stride (only used in one table), BN indicates whether batch normalization is applied to the layer and DO stands for dropout.

Besides the reduction of some of the number of features, the number of epochs was adopted according to the amount of classes, the training data and the used neural network model. To prevent overfitting a lower amount of epochs is used, the decision when to stop training was made based on the loss and accuracy curves on the training and validation data which were generated by the tensorflow framework during the training

TABLE II
ADOPTED MODEL OF CNN+RNN-1 [8].

Type	NoF	Kl	Str	BN	DO
Conv2D	34	4, 2	-	No	-
MaxPooling2D		3, 1	-	Yes	-
Conv2D	34	3, 2	-	No	-
MaxPooling2D		3, 2	-	Yes	-
Reshape	-	-	-	-	-
RNN	100	-	-	No	0.2
Dense	100	-	-	No	0.2
Dense	NoC	-	-	-	-

TABLE III
ADOPTED MODEL OF 1D-CNN FOR TRAFFIC CHARACTERIZATION [9]. THE VALUES IN BRACKETS SHOW THE NUMBER OF FEATURES USED IN THE ORIGINAL MODEL WHICH WHERE ADOPTED IN THIS WORK.

Type	NoF	Kl	Str	BN	DO
Conv1D	200	5	3	Yes	0.05
Conv1D	200	4	3	Yes	0.05
MaxPool	-	-	2	No	0.05
Flatten	-	-	-	-	-
Dense	20 (200)	-	-	No	0.05
Dense	10 (100)	-	-	No	0.05
Dense	5 (50)	-	-	No	0.05
Dense	NoC	-	-	-	-

phase. Since there is no best practice for this, the decision must be made based on experience. The 1D-CNN model was trained for 30 epochs, and the CNN+RNN-1 model for 50 epochs.

D. Testing

One consequence of time slots is that originally contiguous flows are broken into multiple flows (see Fig. 1). The problem arises that these subflows must be assigned to the original flows so that the classification of the subflows can be checked against the given classification of the orginal flows. A subflows is mapped to an original flow if the IP addresses and the port numbers match and the timestamp of the first packet of the subflows is not earlier the timestamp of the first packet of the original flow, and equivalently for the last packet. All this properties except the timestamps have to match exactly.

We decided to consider flows with less than 21 packets to be split flows. The problem is that it is difficult to give a practical lower bound for the amount of packets in a flow to be considered split, because of the different ways to perform the the SSH handshake and authentication. Three packets are needed for the TCP handshake [10], another seven for the SSH handshake, and three for the SSH authentication [11]. Both sides of the connection may decide to send multiple SSH commands in a single TCP packet which would reduce the number of packets needed. In addition, the number of packets increase if additional TCP acknowledge packets are sent, which is also not predictable. In summary, flows that contain full SSH authentication contain at least 21 packets.

The amount of split flows increases when time slots are created with fewer packets. This can occur especially with manual logins, e.g. due to password entries that delay the login. In contrast, machine scans are significantly faster. A wide timeslot is required to capture the entire manual logon. This leads to a trade-off between fast classification with small time slots that are completed after a short time and more robust classification with fewer split flows.

We used common metrics to determine the performance of the classifiers. The scikit library [12] provides a wide variety of metrics, of which we use the following: weighted F1, macro F1 and balanced accuracy. The weighted and macro version of the F1 score is the average of the F1 scores of each class, since the normal F1 score is only defined for binary classification and therefore is not applicable to our multi-class classification. In case of weighted F1, the average is calculated using the arithmetic mean, weighted by the amount of datapoints per class. For macro the mean F1 score of all classes is used. Balanced accuracy is the arithmetic mean of the accuracy per class, weighted by the amount of datapoints per class.

IV. RESULTS

From the training data we separated 10% as validation data, which are not the same as the test data, which are from a different time period. This data is used to determine the quality of the learning process of the neural network, during the learning process. For the validation data we used blocks that are chronologically posterior to the training data because the classifier is supposed to predict future data, which is to be represented in the validation data.

The validation data show that the models were properly trained. This is clearly indicated by the high precision and recall values shown in the confusion matrix (Fig. 2) by the rightmost column and the bottom row, respectively. All classes were learned equally well, and no outliers are apparent.

A. Test dataset

This section presents the results obtained using the two neural network models, the two slot generation parameter sets and the three test datasets. We used the performance metrics described in Section III-D: balanced accuracy, weighted F1 and macro F1. The results are presented in Table IV and Fig. 3. To fully represent the influence of the slot generation parameter, we have included the percentage of split flows.

Both neural network models achieve a performance above 0.947 in all three parameters for parameter set A, and above 0.799 for parameter set B. The weaker performance of B reflects our expectation as flows are split more often and thus incomplete flows need to be classified more often. For parameter set A around 20% of the flows are split while at parameter set B around 60% are split. The performance metrics also include the correct classification of the split flows as such.

As expected, the higher the ratio of split flows, the worse the quality of the classifier. With parameter set A, the scores of the metrics are similar to the results with the validation data. Using parameter set B, compared to A, the ratio of split flows increases from 0.2 to 0.6, while the balanced accuracy and the macro F1 scores decreas from 0.98 to 0.89. Due to the weighting of the classes, the weighted F1 score only decreases

TABLE IV
RESULTS FOR EACH MODEL, SLOT PARAMETER SET AND TEST DATA SET. THE METRICS USED ARE BALANCED ACCURACY, WEIGHTED F1 AND MACRO F1.

Model	Para.	Data	Accuracy	F1 weigh.	F1 macro	% split
1D-CNN	A	I	0.986	0.987	0.983	18.8
		II	0.986	0.984	0.984	21.7
		III	0.957	0.981	0.962	23.4
	B	I	0.887	0.925	0.915	61.8
		II	0.896	0.942	0.924	64.5
		III	0.887	0.939	0.893	63.5
CNN+RNN-1	A	I	0.980	0.980	0.976	18.8
		II	0.974	0.970	0.972	21.7
		III	0.947	0.976	0.949	23.2
	B	I	0.864	0.914	0.879	61.8
		II	0.799	0.888	0.812	64.5
		III	0.864	0.914	0.879	61.8

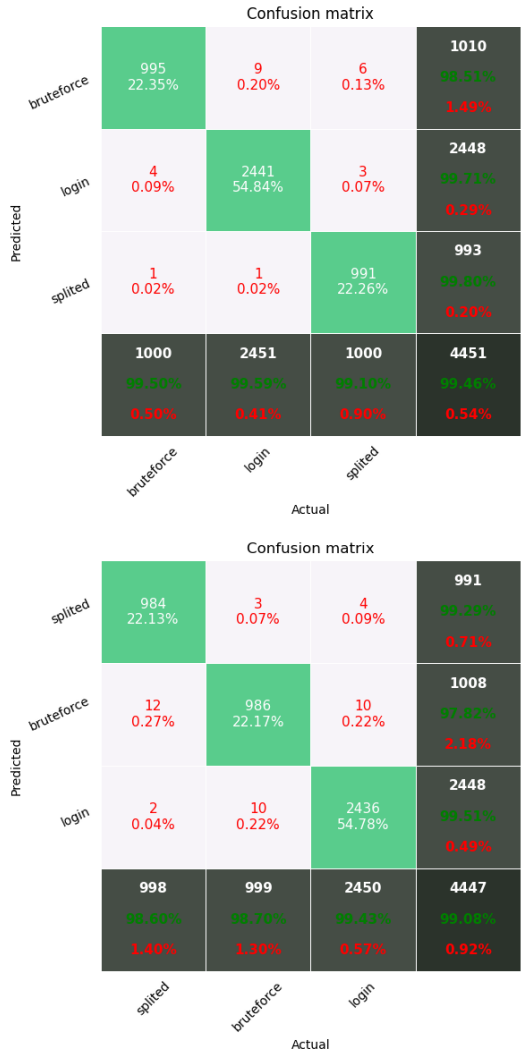

Fig. 2. Confusion matrix of the validation data for the 1D-CNN model (top) and CNN+RNN-1 model (bottom).

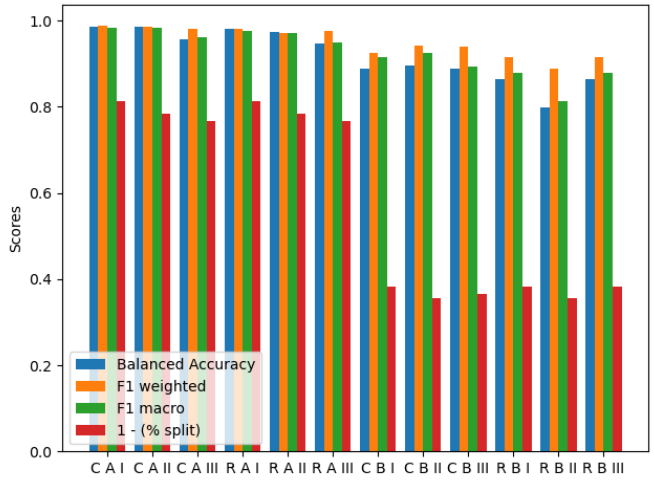

Fig. 3. Results for each model, slot parameter set and test data set. C stands for 1D-CNN, R for CNN+RNN-1. A and B are the parameter sets. The roman numerals I, II and III denote the datasets.

from 0.98 to 0.93 for the 1D-CNN and 0.9 for the CNN+RNN-1 model.

Fig. 4 shows that split flows are classified more robust than the other classes regardless of the parameter set. This has the effect that for parameter set B, the weighted F1 score is lower than the scores of the other two metrics, compared to parameter set A, where the scores of the metrics are closer together. This is because the weighted F1, unlike the other two metrics, takes into account the imbalance of the classes [13]. There are about 5 times more split flows and only about $\frac{2}{3}$ the number of datapoints for brute force attacks and successful login for parameter set B. Because of this the weights for the weighted F1 metric change, unlike the weights for the other two metrics, which depend only on the number of classes. The effect of this can be seen in Fig. 3. For parameter set B, the bar for weighted F1 sticks out above the other two metrics. In terms of robustness, this means that the classes were less equally well classified.

In comparison, the 1D-CNN model was able to classify the data better than the CNN+RNN-1 model. This difference was

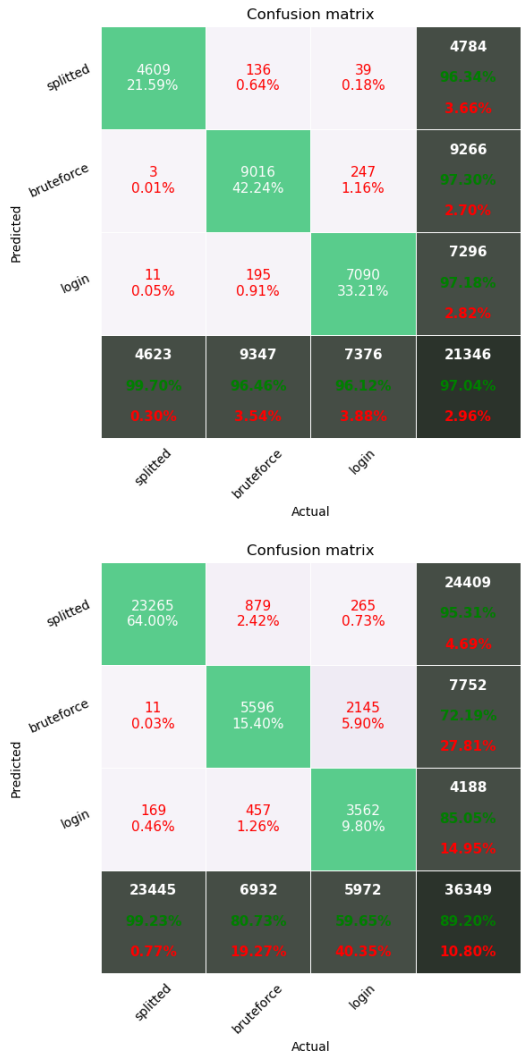

Fig. 4. Confusion matrix of the CNN+RNN-1 model, on dataset II, on parameter set A (top) and B (bottom). The split class perform equal, the bruteforce and login classes perform worse with parameter set B.

not evident in the validation data.

V. Conclusion

Our evaluation of flow-based network traffic classifier provide evidence that neural network based classifiers, even if tested only on a dataset recorded over a short period of time, can reliably classify future network data and are thus suitable for real-world Network Traffic Classification. For live-learning traffic classifiers, which were not the main focus of this work, the results show that the learned models can remain accurate for a long time without much manual effort.

The existing methods based on neural networks we have evaluated in this work can be used for network classification in real-world conditions. For this purpose, the classifiers were tested with a new dataset. To simulate a real-world environment, the training data was taken from very early recordings in the dataset while the test data was taken from recent recordings.

The requirement for modern IDS and IPS is not only to classify as accurately as possible, but also as quickly as possible. This was made possible by using time slots. The associated losses in the quality of the classification were also investigated. Here, our assumption was confirmed that with a shorter available time and the resulting smaller number of packets, the quality of the classification is affected. If a sufficient number of packets are used, the quality of the classification is at a level similar to that achieved in the original work.

References

[1] E. Biermann, E. Cloete, and L. M. Venter, "A comparison of Intrusion Detection systems," *Computers & Security*, vol. 20, no. 8, pp. 676–683, Dec. 2001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404801008069

[2] C. Bergmeir and J. M. Benítez, "On the use of cross-validation for time series predictor evaluation," *Information Sciences*, vol. 191, pp. 192–213, May 2012. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025511006773

[3] A. Moore, D. Zuev, and M. Crogan, "Discriminators for use in flow-based classification," *undefined*, 2013. [Online]. Available: https://www.semanticscholar.org/paper/Discriminators-for-use-in-flow-based-classification-Moore-Zuev/68919d1eaf5708163da8172691a8bec25b64105b

[4] A. Malik, R. de Fréin, M. Al-Zeyadi, and J. Andreu-Perez, "Intelligent SDN Traffic Classification Using Deep Learning: Deep-SDN," in *2020 2nd International Conference on Computer Communication and the Internet (ICCCI)*, Jun. 2020, pp. 184–189.

[5] J. Zhang, F. Li, H. Wu, and F. Ye, "Autonomous Model Update Scheme for Deep Learning Based Network Traffic Classifiers," in *2019 IEEE Global Communications Conference (GLOBECOM)*, Dec. 2019, pp. 1–6, iSSN: 2576-6813.

[6] A. Tongaonkar, R. Keralapura, and A. Nucci, "SANTaClass: A Self Adaptive Network Traffic Classification system," in *2013 IFIP Networking Conference*, May 2013, pp. 1–9.

[7] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic, "AppScanner: Automatic Fingerprinting of Smartphone Apps from Encrypted Network Traffic," in *2016 IEEE European Symposium on Security and Privacy (EuroS P)*, Mar. 2016, pp. 439–454, iSSN: null.

[8] M. Lopez-Martin, B. Carro, A. Sanchez-Esguevillas, and J. Lloret, "Network Traffic Classifier With Convolutional and Recurrent Neural Networks for Internet of Things," *IEEE Access*, vol. 5, pp. 18 042–18 050, 2017, conference Name: IEEE Access.

[9] M. Lotfollahi, M. Jafari Siavoshani, R. Shirali Hossein Zade, and M. Saberian, "Deep packet: a novel approach for encrypted traffic classification using deep learning," *Soft Computing*, vol. 24, no. 3, pp. 1999–2012, Feb. 2020. [Online]. Available: https://doi.org/10.1007/s00500-019-04030-2

[10] "rfc793." [Online]. Available: https://datatracker.ietf.org/doc/html/rfc793section-3.4

[11] "rfc4253." [Online]. Available: https://datatracker.ietf.org/doc/html/rfc4253

[12] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[13] "sklearn.metrics.f1_score." [Online]. Available: https://scikit-learn/stable/modules/generated/sklearn.metrics.f1$_score.html$