Network Traffic Classification: Time-based vs Packet-based Approaches using Sub-flows

Baris Yamansavascilar
Innovations Department
Airties
Istanbul, Turkey
baris.yamansavascila@airties.com

Omer Topal
Innovations Department
Airties
Istanbul, Turkey
omer.topal@airties.com

Mehmet Şükrü Kuran
Innovations Department
Airties
Istanbul, Turkey
sukru.kuran@airties.com

***Abstract*—In this study, we introduce two machine and deep learning based approaches to network traffic classification using sub-flows and sliding windows for near-real-time use-cases such as providing QoS/QoE to application flows. We develop two approaches: (1) a packet-based sub-flow method that uses a fixed number of packets for each instance, and (2) a time-based sub-flow method that utilizes packets in a fixed time range for each instance. Unlike similar studies conducted in the literature, our methods are designed to be applicable to real-time use cases where the classification should give an accurate answer quickly after the start of a network traffic to provide the necessary QoS/QoE to the network flows. We have conducted experiments with both methods considering the efficient window sizes and sliding intervals. We present the results of two classification goals: (1) application identification and (2) application category identification. Our results show that the proposed time-based approach provides at least 89% accuracy in application identification (depending on the application), and 96% in application category identification which makes it highly accurate for use-cases to provide QoS/QoE to application flows.**

***Index Terms*—Network Traffic Classification, Application-based, Machine Learning, Deep Learning**

I. Introduction

In today's networking environment, network traffic classification (NTC) is a critical capability enabling many use cases such as anomaly detection, analytics calculation, application identification and web service detection. Among them, application identification is crucial for Internet Service Providers (ISPs) in home gateways to provide the necessary quality of service (QoS) and quality of experience (QoE) based on the service level agreement (SLA) requirements of end-user applications [1]. Moreover, the fact that various applications have different QoS/QoE requirements today greatly increases the importance of application identification.

NTC has been studied widely in the last two decades using different methods based on the existing environments and requirements [2]. Here, two classical methods are (1) using port addresses and (2) using the three-tuple information (server IP, port address, transport layer protocol) which are publicly provided by application vendors. Although straightforward and efficient, such methods may not be reliable since the port address and three-tuple information of the applications may change dynamically leading to false positive results for the detection. Also, many applications do not have such consistent server information limiting the usability of such methods.

Another well-known method for NFC is using deep packet inspection (DPI) [3]. Such methods utilize inspecting layer headers of each network packet and applying expert-system-like rulings for classification. However, DPI necessitates that each packet in the network traffic be investigated, which makes its deployment very costly in terms of computational complexity. Moreover, it may also violate the privacy of users.

In the last decade, machine learning (ML) and deep learning (DL)-based methods have been proposed for NTC using statistical flow information of network traffic that provides high accuracy, precision, and recall while working according to privacy regulations [4]. Similar to the DPI-based approach, these ML/DL methods may require considerable computational power which may not be suitable for home gateways. Considering the ML-based methods, many studies take all flow information into account for both model training and inference. This is not suitable for an application identification use-case due to its requirement for real-timeliness. Applications should be identified as fast as possible right after the flow starts so that the gateway can take a corresponding action. Moreover, they also use features that belong to the application layer whose information cannot be extracted due to increased computational complexity as well as encryption. On the other hand, DL-based methods use raw byte sequences of the network packets which may cause high computational complexity since there would be many active flows with high traffic volume in the network.

In this study, we propose an ML/DL-based sliding window approach to the NTC problem with the application identification use-case. We also utilize the same methods for identifying the application's category as well[1]. In order to apply the sliding window approach, we first convert network flows to sub-flows for both training and testing phases. Afterwards, we extract the corresponding features for each sub-flow and create the corresponding instances. Finally, we train models through instances and then compare the performance of different algorithms. Using this approach, deploying an NTC model in a real-time environment would be more efficient since applying sliding

[1]For the sake of brevity we will refer to application category identification as category-based classification.

TABLE I: Comparison Between Related Studies

Study	Sub-Flow	Inter-arrival Times	DPI Dependency	Full Flow Dependency	Identification Type
Huoh et al. [5]		✓	✓	✓	App
Lin et al. [6]		✓	✓	✓	Category
Wu et al. [7]	✓	✓	✓	✓	Category
Telikani et al. [8]		✓	✓		App
Guo et al. [9]				✓	App
Shapira et al. [10]		✓	✓		App
Xiao et al. [11]			✓		App
Lopez et al. [12]	✓	✓	✓		App
Our proposal	✓	✓			App

windows is less complex regarding packet/byte inspection. Moreover, we only utilize network-layer features so that a possible deployment in a home gateway would be seamless if hardware limitations are met. The main contributions of our paper are as follows.

- We propose two ML/DL-based NTC methods for application identification and category-based classification use-cases, one packet-based and another time-based, applying a sliding windows-based approach.
- Due to the nature of sliding windows, the proposed methods can be applied in near real-time for use cases such as delivering QoS/QoE to application flows.
- We evaluate the performance of the proposed methods in terms of accuracy, F-1 scores, and confusion matrices for both application and category-based variants. Our results show that the time-based sliding windowing approach gives the highest accuracy in both variants.

The rest of this paper is organized as follows. In Section II, we elaborate on the related works. Next, we introduce our system design in Section III. In Section IV, we evaluate the performance of different settings and represent the experimental results. Finally, we conclude our study in Section V.

II. Related Works

In the literature, recent studies have proposed ML and DL-based approaches to address NTC for application identification with different settings and architectures. Especially, since the dataset size affects the performance of the DL algorithms, choosing the correct architecture along with the corresponding algorithm is critical. On the other hand, some authors presented federated learning (FL) methods to provide distributed approaches considering potential privacy and network issues.

In [5], Huoh et al. proposed a graph neural network-based mechanism that utilizes a classifier considering a non-Euclidean domain consisting of three building blocks. They took the packet, packet interarrival times, and meta-features of flows into account in the graph representation. Their results show that their GNN-based approach was successful in terms of recall, precision, and F1-score using the ISCXVPN2016 dataset [13]. Lin et al. developed a multi-modal DL framework called PEAN [6]. They used raw bytes and lengths of the corresponding packets of flows in order to extract the related features. Their proposed framework consists of five layers considering embedding models. They compared the success of their approach with the benchmark applications from the literature regarding metrics of precision, recall, and F1-score. Shapira et al. developed a CNN-based DL approach, FlowPic for application identification and category classification [10]. They first took the packet size and packet arrival times into account and created an image for each flow. Afterwards, they used those image flows as input for CNN for the NTC. Their results showed that the accuracy of this approach is above 99.2%. Note that all these works utilize the whole packet flow to identify the application in question, making them unsuitable for a near real-time deployment. In [12], Lopez et al. proposed a DL approach that consists of a combination of CNN and LSTM. Unlike the previous methods, they only used the first 20 packets of the network flows in order to train their model. From these packets, six features have been used including TCP window size making the approach specific to TCP flows and dependent on a DPI mechanism. They classified several application classes whose ground truth was extracted by a DPI tool called nDPI [14]. In [15], Ozkan et al. modeled the application flows as a stochastic discrete-time Markov chain considering the sequentiality between packets. Next, they quantized the corresponding flow rates into different states. Furthermore, they extracted packet-level and flow-level features considering downlink and uplink traffic. Afterwards, they created instances, each of which is a single application flow. Thus, they proposed three Markovian-based models providing up to 89% classification accuracy. However, since their approach uses the full-flow information as an instance, its deployment in a near real-time environment would not be efficient.

Among the ML and DL solutions, some observe that NTC has an inherent class imbalance issue. Some application types (e.g., video) produce much more packets than other applications within each second. Therefore, related instances to those types would have much more in numbers in a dataset. As such, in [7], Wu et al. focused on the class imbalance problem and proposed a chain hierarchical structure considering both stationary and non-stationary environments. They showed the performance of their approach using the G-means and minority F-measure (MF) metrics. Telikani et al. proposed a cost-sensitive DL method considering the class-imbalance problem in NTC [8]. To this end, they created a cost matrix for the data distribution regarding different partitions of the data. They implemented two classifiers including a stacked autoencoder and convolution neural network. The results based on the ISCXVPN dataset showed that their approach outperformed

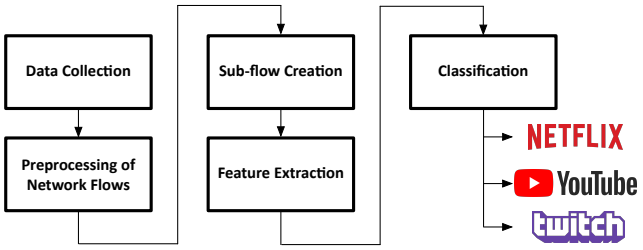


Fig. 1: Generic ML system design for network traffic classification-based application identification

the resampling-based approaches.

Considering the privacy of users and also relieving the burden on the communication links through NTC, FL has also been investigated as an alternative and efficient approach. In [9], Guo et al. focused on a FL approach that considers bandwidth efficiency and the privacy of clients. They addressed the problem of skewness in the traditional FL-based approach that causes low accuracy due to the heterogeneity of clients. Therefore, they proposed a federated analytics (FA)-based FL approach, FEAT, that proactively selects the low-skewness clients for the global model. Their results showed that FEAT outperformed several benchmark methods under different user heterogeneities. In [11], Xiao et al. developed an FL-based framework that uses self-supervised learning through Generative Adversarial Networks (GANs). Hence, they could classify the unknown service types in the network without requiring labeled samples. They compared the performance of their approach with the corresponding clustering methods using the ISCXVPN2016 dataset. Their approach outperformed them by achieving a 20% improvement in accuracy.

Although achieving high accuracies, related works use either all flow information or conduct DPI over a few packets to identify an application as given in Table I. Such dependencies make these solutions incompatible for near real-time use cases in providing QoS/QoE to application flows. Here, note that to provide QoS/QoE via corrective actions over a traffic flow, classification should be done within a certain time window after the flow's initialization. Otherwise, it will simply be too late to take corrective actions and cannot achieve the desired QoS/QoE. To this end, we neither examine any individual network packets via DPI nor utilize all flow information in this study. Our proposed solution uses a sliding window approach through sub-flows of each flow and therefore becomes suitable for such near real-time use cases. To the best of our knowledge, the effect of a sliding window approach through sub-flows considering packet-based and time-based implementations has yet to be investigated deeply for NTC.

III. System Design

Considering the overall system design, we utilize five essential steps based on the generic ML classification scheme including data collection, preprocessing of flows, sub-flow creation, feature extraction, and classification as shown in Figure 1.

TABLE II: Application and corresponding categories in the dataset

Application	Category
Netflix	Video on Demand (VOD)
YouTube	
YouTube Live	Video Live Streaming (VLS)
Twitch	
Spotify	Sharing & Media Streaming (S&MS)
Texting (WhatsApp)	
Skype	Teleconferencing (TC)

A. Data Collection

We develop our application identification model based on supervised learning, hence we focus on labeled data. We use a publicly available dataset, [16], which was collected as part of our previous joint NTC study [15]. In this dataset, there are network flows of seven applications including Netflix, Skype, Spotify, Twitch, Texting (WhatsApp), YouTube Live, and YouTube. The applications are also mapped into four application categories: video on demand (VOD), video live streaming (VLS), sharing & media streaming (S&MS), and teleconferencing (TC). For each application, packets belonging to 30 different flow instances were collected using the Wireshark where the duration of each flow in the dataset is 660 seconds. This is especially important for this study since we can fairly evaluate the time-based approach for each type of application. The application and corresponding categories are given in Table II as defined in [15].

B. Sub-flow Approach

As explained in Section I, the application identification system we propose is aimed at near real-time use cases. Therefore, a system designed for such a goal should be able to make a decision by evaluating a random part of a flow instead of the whole. As explained in [15], by analyzing the collected data set, we observe that the network traffic of application flows repeats a certain utilization pattern over time. To this end, we divide each flow into smaller and consecutive flow subsets, which we call sub-flows. This is, in essence, similar to activity recognition systems [17]. Afterwards, we extract features for each sub-flow and then create the corresponding instances.

We also observe that if we divide a flow into sub-flows where consecutive sub-flows do not overlap in time, certain patterns may be lost, which negatively affects the performance of the application identification. Therefore, we choose to deploy a sliding window approach to generate sub-flows by defining sliding intervals.

C. Preprocessing

In order to apply the sub-flow approach efficiently, the parameter which is used to define the corresponding windows should be selected correctly. To this end, we apply two types of windows in this study: (1) packet-based, and (2) time-based.

1) Packet-based Windows: In packet-based windows, we create sub-flows using a predefined number of packets. In this approach, each window consists of the same amount of network packets. However, even though window sizes would

be the same, this approach can cause a class imbalance problem since some application types such as video streaming have much more packets than the other application types.

2) Time-based Windows: In time-based windows, sub-flows are created using a predefined time interval. Hence, each window would be the same size in terms of time. However, in this approach, each window size may have a different number of packets based on the application type. Additionally, if the number of flows for each application is the same in a dataset, it would automatically eliminate the class imbalance problem which is a crucial issue in NTC.

D. Feature Extraction

We perform feature extraction for packet and time-based windows separately since the former consists of a fixed number of packets for each window while the latter has fixed time intervals. We also pay attention that each extracted feature should belong to the network layer. Furthermore, we carry out normalization for each feature in order to accelerate the training process.

1) Packet-based Features: In packet-based windows, we use the *arrival time* and *packet size* of each packet as features since we have a fixed amount of packets. Thus, for example, if the window size is 30 packets, each instance in the dataset would have 60 features.

2) Time-based Features: In time-based windows, we extracted the features for each second and then used all of them regarding the window size. For example, if three features are extracted for each second, and the corresponding window size is 10 seconds, then 30 features are used for each instance. To this end, we exploit the following eight features for each second: *the number of packets*, min/max/average/total *packet sizes*, and min/max/average *interarrival times*.

E. Classification

After the feature extraction, we create labeled instances for the classification step using the corresponding windows. Accordingly, for both packet-based and time-based approaches, we split 80% of instances for the training phase, and the remaining 20% for the testing phase. For the classification, we use three traditional ML algorithms: decision tree, random forest, and k-NN. Additionally, we use multi-layer perceptron as DL. After the completion of the training and testing phases, we evaluate the performance of each final model considering the accuracy, recall, precision, and F-1 score metrics. However, since the F-1 score is the harmonic mean of recall and precision, we only use the F-1 score and accuracy in our performance evaluation.

IV. Performance Evaluation

For each algorithm, we first train models with different configuration parameters (layer count, node per layer, activation function, etc.) and choose the model with the highest performance as the algorithm's final model in order to evaluate the performance of packet-based and time-based approaches. Afterwards, we used the final models for each algorithm for the performance evaluation. Note that we point out the accuracy and F-1 score regarding the performance. We ran the final models in a MacBook Pro M1 Pro chipset with 16GB of RAM and observed that each model run under one second.

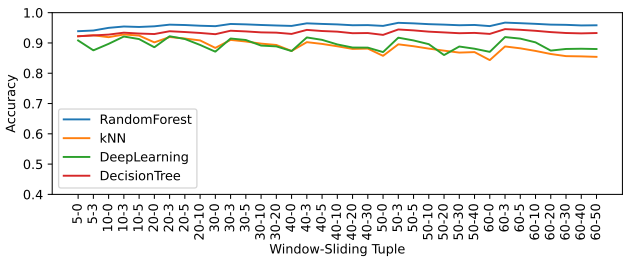


Fig. 2: Accuracy of algorithms based on applications using packet-based windows.

A. Model Configurations

Considering DL, we used the *PyTorch* framework for the implementation. Our final model consists of four layers. Each layer has a number of nodes, which is half of its previous layer. Note that we deployed 512 nodes in the initial layer. For each node, we deployed the ReLU activation function. Furthermore, we used the Adam optimizer with a learning rate of 0.0001. For the loss, we calculated the minimum square error.

On the other hand, regarding decision tree, random forest, and k-NN algorithms, we used the *scikit-learn* framework for the implementation. In these algorithms, the random forest has 100 estimators, while k-NN utilizes the information of five nearest neighbors. Moreover, Gini impurity is calculated for the decision tree to measure the quality of a split.

B. Results of Packet-based Windowing Approach

We first evaluated the performance of the packet-based windowing approach considering both application and category-based classification. Figure 2 shows the accuracy of application-based classification of the aforementioned four algorithms where the x-axis represents the $(window, sliding\ interval)$ tuple. The results show that the random forest model is the most successful, followed by the decision tree, then DL, and finally, the k-NN model. The low accuracy of the DL model compared to the random forest is expected since traditional neural networks require a high amount of instances for tabular data to achieve high accuracy. Therefore, traditional ML-based models can outperform them if the size of the corresponding dataset is small or medium, as shown in [18], [19]. On the other hand, we also observed that applying sliding intervals enhances the accuracy corresponding to the window size. With the exception of a window size of five packets, when we apply sliding, the accuracy of models increases since they can capture more patterns in the dataset.

Since the random forest model is the most accurate, we further analyzed its results regarding F-1 scores for each application as shown in Figure 3. The results show that the effect of sliding intervals is similar to the accuracy results. On the other hand, as pointed out in [15], the performance of the model

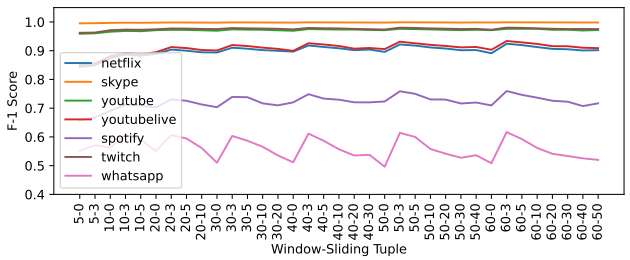


Fig. 3: F-1 score of each application using random forest in packet-based windows.

for Texting (WhatsApp) and Spotify instances is the least successful. Therefore, for further investigation, we analyzed the corresponding confusion matrices for applications and categories by selecting the $(window, sliding\ interval)$ tuple as $(40, 3)$. Thus, as shown in Figure 4a, Texting (WhatsApp) and Spotify traffic are confused with each other in this dataset, as also highlighted in [15]. The main reason for this confusion is that the interarrival times between consecutive packets in a window would be very long considering that those applications perform a file transfer within a small time range. Therefore, the corresponding features cannot reflect the actual patterns correctly. Coming to category-based classification, as shown in Figure 4b, the accuracy is very high in three categories since, here, misclassification between applications of the same category disappears. The only exception is the S&MS category (Texting and Spotify) where accuracy is 86% due to the same reasons as in application-based classification.

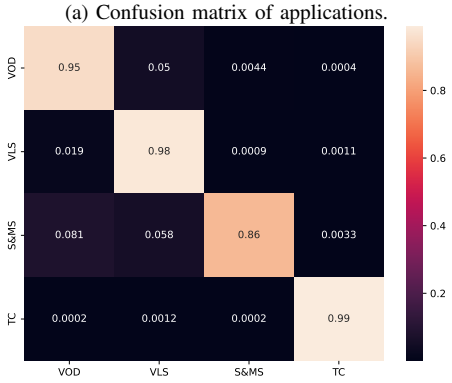
C. Results of Time-based Windowing Approach

For the time-based windowing approach, the accuracy of algorithms for application-based classification is shown in Figure 5. The results show that the accuracy order among the four algorithms is the same as that of the packet-based windowing approach. However, the accuracy variance between different parameter selections is much higher. The main reason for this outcome is that the number of packets for each instance would be different based on the sampling and application type. Therefore, the accuracy would be affected by a corresponding $(window, sliding\ interval)$ tuple. On the other hand, we observed that sliding intervals would continue to affect accuracy, similar to packet-based windows.

Since the random forest model is the most successful again, we evaluated the F-1 score of each application using its results as shown in Figure 6. Based on these results, $(60, 3)$ and $(30, 3)$ tuples provide higher F-1 scores, respectively. However, since predicting the class of an instance in a short amount of time considering a real-time system is also essential, we chose the final model created by instances of $(30, 3)$ tuple. The confusion matrices of this model for both applications and categories are shown in Figures 7a, and 7b, respectively. The results show that the performance of the time-based approach outperforms the packet-based approach. This is a significant outcome since the time-based approach achieves



(a) Confusion matrix of applications.



(b) Confusion matrix of categories.

Fig. 4: Confusion matrix of random forest model using $(40, 3)$ tuple for packet-based windows.

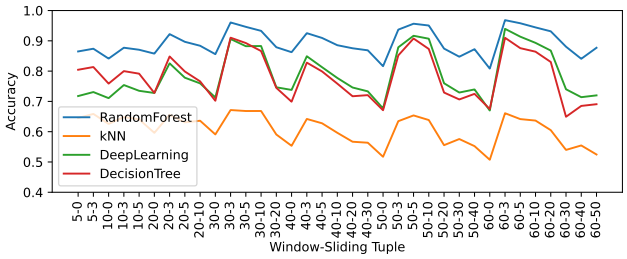


Fig. 5: Accuracy of algorithms using time-based windows.

better results when using fewer instances. Note that some categories, such as VOD and the corresponding applications, can produce more packets than others. Hence, the packet-based approach can make many instances. On the other hand, the time-based approach would produce instances based on fixed time intervals. Therefore, the number of instances in the packet-based approach is usually more than the time-based approach. Furthermore, consecutive packets in a window using the packet-based approach would have distant time frames between them. As a result, the time-based windows can capture the corresponding traffic patterns for each application and category better since the consecutive packets for each window are in the same time range. Therefore, Spotify and Texting (WhatsApp) are predicted with higher accuracy than the packet-based windowing approach. Moreover, the time-

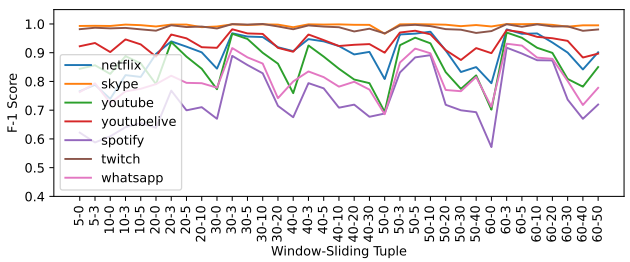
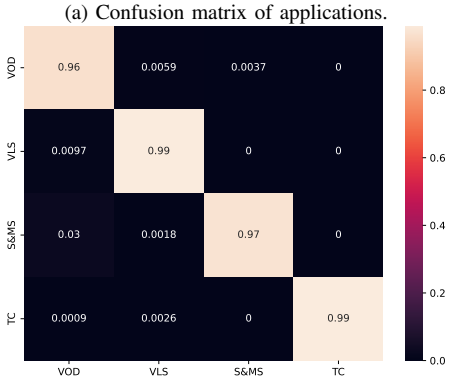


Fig. 6: F-1 score of each application using random forest in time-based windows.



(a) Confusion matrix of applications.



(b) Confusion matrix of categories.

Fig. 7: Confusion matrix of random forest model using $(30, 3)$ tuple for time-based windows.

based approach automatically eliminates the class imbalance problem since instances are created using fixed time intervals for each flow.

For both time-based and packet-based approaches, it is important to note that we applied 10-fold cross-validation (CV) to measure the quality of final models correctly and to compare their overall performance fairly considering the statistical significance. For each model, 10-fold CV results show that the average absolute accuracy difference is 0.00250, which validates the performance of the models. On the other hand, since the difference between the accuracy of the different models is consistent for each fold, which is also validated by applying a t-test, we statistically validated the observed results.

V. Conclusion and Future Work

In this study, we applied a sub-flow approach through sliding windows to identify popular applications and their categories. Since performing classification in a short amount of time is essential for real-time systems, we avoided complex approaches such as utilizing byte chunks of packets. Instead, we exploited statistical features using packet-based sub-flows and time-based sub-flows. To this end, we compared the performance of these two sub-flow approaches using different machine learning algorithms. Moreover, we investigated the effect of different window sizes and corresponding sliding intervals on the performance of final models. The experimental results showed that applying sliding intervals can increase the success of the corresponding models significantly as they can capture more patterns from the training data.

In the future, we plan to collect a high amount of labeled flow data from the field and then apply more advanced DL methods such as Long Short-Term Memory (LSTM) and Convolutional Neural Networks. We expect that the corresponding DL models can achieve a high performance considering the extensive tabular data. On the other hand, since our model depends on the corresponding classes in the dataset, the performance of $(window, sliding\ interval)$ tuples would be different in a new dataset. Therefore, we plan to apply our methodology using a more extensive dataset in order to observe its effect.

References

[1] F. Metzger, S. Geißler, A. Grigorjew, F. Loh, C. Moldovan, M. Seufert, and T. Hoßfeld, "An Introduction to Online Video Game QoS and QoE Influencing Factors," *IEEE Communications Surveys & Tutorials*, vol. 24, no. 3, pp. 1894–1925, 2022.

[2] A. Azab, M. Khasawneh, S. Alrabaee, K.-K. R. Choo, and M. Sarsour, "Network Traffic Classification: Techniques, Datasets, and Challenges," *Digital Communications and Networks*, 2022.

[3] M. Finsterbusch, C. Richter, E. Rocha, J.-A. Muller, and K. Hanssgen, "A Survey of Payload-based Traffic Classification Approaches," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 2, pp. 1135–1156, 2013.

[4] F. Pacheco, E. Exposito, M. Gineste, C. Baudoin, and J. Aguilar, "Towards the Deployment of Machine Learning Solutions in Network Traffic Classification: A Systematic Survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1988–2014, 2018.

[5] T.-L. Huoh, Y. Luo, P. Li, and T. Zhang, "Flow-based Encrypted Network Traffic Classification with Graph Neural Networks," *IEEE Transactions on Network and Service Management*, 2022.

[6] P. Lin, K. Ye, Y. Hu, Y. Lin, and C.-Z. Xu, "A Novel Multimodal Deep Learning Framework for Encrypted Traffic Classification," *IEEE/ACM Transactions on Networking*, 2022.

[7] Z. Wu, Y.-n. Dong, J. Jin, H.-L. Wei, and G. Xie, "Multimedia Traffic Classification for Imbalanced Environment," *IEEE Transactions on Network Science and Engineering*, vol. 9, no. 3, pp. 1838–1852, 2022.

[8] A. Telikani, A. H. Gandomi, K.-K. R. Choo, and J. Shen, "A Cost-sensitive Deep Learning-based Approach for Network Traffic Classification," *IEEE Transactions on Network and Service Management*, vol. 19, no. 1, pp. 661–670, 2021.

[9] Y. Guo and D. Wang, "FEAT: A Federated Approach for Privacy-preserving Network Traffic Classification in Heterogeneous Environments," *IEEE Internet of Things Journal*, vol. 10, no. 2, pp. 1274–1285, 2022.

[10] T. Shapira and Y. Shavitt, "FlowPic: A Generic Representation for Encrypted Traffic Classification and Applications Identification," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1218–1232, 2021.

[11] Y. Xiao, R. Xia, Y. Li, G. Shi, D. N. Nguyen, D. T. Hoang, D. Niyato, and M. Krunz, "Distributed Traffic Synthesis and Classification in Edge Networks: A Federated Self-supervised Learning Approach," *IEEE Transactions on Mobile Computing*, vol. 23, no. 2, pp. 1815–1829, 2024.

[12] 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.

[13] G. Draper-Gil, A. H. Lashkari, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of Encrypted and VPN Traffic using Time-related Features," in *Proceedings of the 2nd international conference on information systems security and privacy (ICISSP)*, 2016, pp. 407–414.

[14] T. Bujlow, V. Carela-Español, and P. Barlet-Ros, "Independent Comparison of Popular DPI Tools for Traffic Classification," *Computer Networks*, vol. 76, pp. 75–89, 2015.

[15] H. Ozkan, R. Temelli, O. Gurbuz, O. K. Koksal, A. K. Ipekoren, F. Canbal, B. D. Karahan, and M. Ş. Kuran, "Multimedia Traffic Classification with Mixture of Markov Components," *Ad Hoc Networks*, vol. 121, p. 102608, 2021.

[16] "WiFi-Dataset-SU-Airties," https://github.com/receptemelli/WiFi-Dataset-SU-Airties, accessed: 2024-09-03.

[17] F. Gu, M.-H. Chung, M. Chignell, S. Valaee, B. Zhou, and X. Liu, "A Survey on Deep Learning for Human Activity Recognition," *ACM Computing Surveys (CSUR)*, vol. 54, no. 8, pp. 1–34, 2021.

[18] S. Rezaei and X. Liu, "Deep Learning for Encrypted Traffic Classification: An Overview," *IEEE communications magazine*, vol. 57, no. 5, pp. 76–81, 2019.

[19] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapé, "Mobile Encrypted Traffic Classification using Deep Learning," in *2018 Network traffic measurement and analysis conference (TMA)*. IEEE, 2018, pp. 1–8.