Flow-based Service Type Identification using Deep Learning

Mona Elsaadawy, Petar Basta, Yunjia Zheng, Bettina Kemme
McGill University
Montreal, Canada
first.last@mail.mcgill.ca, kemme@cs.mcgill.ca

Mohamed Younis
University of Maryland Baltimore County
Baltimore, Maryland, USA
younis@umbc.edu

***Abstract*—Automatic identification of the service type used by network flows (e.g., HTTP and MySQL) is an essential part of many cloud management and monitoring tasks for quality of service, security monitoring, resource allocation, etc. Several studies have adapted deep learning models for accurate service type identification of network traffic. These models vary in how the message flow data is used and what datasets are considered. There are no published guidelines on selecting the best approach for automating the service identification process. In this paper, we opt to fill such a technical gap and provide a detailed study of the trade-offs of different deep-learning based approaches for service type identification of network traffic. Towards this end, we generate flow-based datasets for a wide range of service types that are commonly deployed in the cloud. We consider two different deep learning models that have shown promising results in this context, and show their performance for both payload- and header-based datasets, considering fundamental parameters such as dynamic service port configuration, flow direction and the packet order in the flow stream.**

***Index Terms*—Service type identification, Convolutional neural network, Deep learning, Recurrent neural network.**

I. Introduction

Cloud applications are often deployed as multi-tier or multi-component systems where the individual components provide a specific service type. A classical example is a multi-tier architecture with a web-server front-end, a MemCached backend for caching purposes, and a MySQL database for persistence. The scope has further expanded to include replicated services, distributed services. e.g., multi-node Cassandra key-value storage, or computational services such as Spark. Inferring the service types of the individual components is crucial for various cloud management tasks such as QoS provisioning, identifying faults, resource allocations, and security monitoring. A Service Type Classifier (STC) is a tool that identifies the service and/or application type associated with a network connection by analyzing the messages exchanged through this connection. Service types are typically determined by the used client/server protocol. While HTTP is widely used in cloud applications, many software systems, such as, MySQL, MemCached, etc., have their own communication protocol. In addition, the exchanged messages between client and server in these software systems follow protocol-specific patterns.

Existing techniques for STC can be grouped based on the methodology used. When services have predefined static port settings, then a *port-based* methodology works well. *Deep packet inspection (DPI)* techniques match message formats with predefined protocol-specific characteristics [1], requiring to maintain an up-to-date rule database for all relevant services. Meanwhile, *flow statistics-based* approaches make use of the stochastic profile of network flows such as transmission length, packets arrival times, and flow duration, to group packets into specific application/service categories [2]. Traditional machine learning classifiers, such as logistic regression, decision trees and support vector machines are commonly used for such purpose. More recently, deep learning has been exploited as STC. The interesting aspect is that the statistical flow characteristics (features) are extracted automatically. Deep learning approaches avoid the need for detailed knowledge of the service specifics and message formats. Hence, in this paper, we focus on deep learning models.

In existing deep-learning based STCs, network flows are considered as a time-based sequence of data, just as video frames or text words, which can be fed to sequence-based learning models for classification purposes. In header-based approaches [3], [4], the learning features are extracted from the packet header of underlying TCP/UDP communication protocol. Examples of header-based learning features include source/destination port numbers, number of transmitted bytes, and packet inter-arrival time. The flows belonging to the same service type should have a local similarity in the values of those header-based features, which enables the deep learning model to identify different service types. In payload-based approaches [5], the learning features are extracted from the first bytes of the message payload, i.e., the header information of the service type's communication protocol. Thus, the URL string can be found for HTTP-based protocols, and the put/get headers for caching services, etc. One can expect header-based analysis to have a smaller model training time as the number of extracted features is limited, and also work better for secured communication as the packet-header information is not encrypted. However, as the learning features are limited, the STC accuracy might not be good enough. If one of the features is the port number, a header-based approach will work well when standard ports are used, but likely less so with dynamic port numbers.

In this paper, we conduct a thorough analysis of using deep learning for STC considering various service types that are commonly used in multi-component cloud applications. In particular, we generate a large flow-based dataset for a wide range of commonly used service types and provide a performance comparison for both header-based and payload-based approaches that demonstrates their trade-offs. We con-

978-1-6654-0522-5/21/$31.00 ©2021 IEEE

sider RNN and a combination of CNN and RNN architectures that have shown to outperform other approaches [3], [5]. Moreover, we analyze the sensitivity of these approaches to crucial parameters such as dynamic configuration of service port numbers, flow direction and the location of packets in the flow stream.

II. Related Work

Several studies have focused on network traffic classification, including rule-based or statistical correlation-based approaches [1]. Many of these studies have adapted general machine learning techniques. Some approaches use supervised learning, such as random forests [6], multi-layer perceptron, C4.5 decision tree, and support vector machines [7]. Singh et al. [8] use unsupervised K-means to form groups of different applications based on the similarity of their network traffic. However, most traditional machine learning models require tedious feature engineering to reduce data complexity and find appropriate parameters to be fed to the machine learning algorithm. Thus, the contribution of this paper has been inspired by recent work that has explored the use of deep learning models to classify network traffic. For instance, Wang et al. [4] have introduced a CNN model to differentiate between malware traffic and normal traffic using the flow ID. Similarly, Martín et al. [3] have applied several architectures of RNN and CNN models to detect the service type of network flows, while using packet-header information as learning data. On the other hand, Lim et al. [5], use packet payload for training two deep learning models, with focus on predicting the application type within HTTP-based web traffic. In most of these approaches the model automatically learns the features without manual feature extraction. However, so far only a limited set of design parameters is considered, such as the flow time-series length and payload size.

III. Dataset preprocessing

In this study, we have built datasets by collecting Packet Capture (PCAP) traces using tcpdump from a wide set of applications that use various service types. The considered service types are HTTP, various database systems as well as caching services. We present the details of the applications and the service types they use in Section V. We have formulated two dataset types: header-based and payload-based. For both types, the learning unit is a network flow that is labeled with a particular service type. We have collected the PCAPs as per each service type to serve as ground truth for the STC.

A network flow is defined for a pair of two communication endpoints defined by their IP addresses and port numbers, and includes the packets transferred using a specific transport protocol (TCP or UDP). To prepare an input record for our learning model, we take N consecutive packets of a network flow (representing a time-series) and represent each of the packets in form of a feature vector of length X. We use Min-Max Normalization to scale the feature values between 0 and 1 to enhance the learning process.

A. Header- and Payload-based datasets

To represent the network flow profile in our header-based dataset, we follow a similar approach as [3] and explicitly extract four meaningful features from each packet's header: the number of bytes in the packet payload, the TCP window size (set to zero for UDP), packet inter-arrival time, and direction of the packet. In contrast to [3], our default evaluations use a dataset that does not include the packet port number as a learning feature as we expect that including the port number might lead to poor learning outcome when dynamic port numbers are deployed. However, we have constructed a variant of the header-based dataset that considers the service port number to investigate its effect on the STC performance.

For our payload-based dataset, we adopt the pre-processing methodology proposed in [5], where each byte of the payload data of a packet is converted into an image pixel (i.e., 256 possible values). According to a pre-defined image size value X, the first X bytes of packet payload are extracted as the pixels of an image. In case the packet payload data is less than X, we use zero-padding to match X. We discard any packet with no payload such as the flow's control packets.

B. General Design Parameters

When evaluating the performance of the approaches, we considered a wide range of parameters during flow extraction. Some of them were fixed after some preliminary testing, the effect of others will be presented in detail in Section V.

Number N of analyzed packets per flow: The considered length of the flow sequence is an important parameter [3], [5]. We considered 20, 60 and 100 packets of each network flow for both header-based and payload-based datasets. Our preliminary results show that longer sequences are good for payload-based models, while header-based approaches work better with lower number of N. The more packets are considered per flow, the more the header-based model has difficulty to distinguish between different service types. Thus, N is set to 20 for the header-based dataset and 100 for the payload-based dataset in the experiments presented here.

Extracted payload size X: For the header-based approach, X is fixed as we explicitly extract the features. In contrast, X is a configurable parameter for the payload-based datasets. Thus, we have tested with extracting the first 9, 12, 16, 20, 25, 36 and 1024 bytes of each packet. Our results showed that a relatively large packet size is beneficial. Therefore, X is set to 36 in the experiments presented in this paper. However, compared to [5], very large X values, such as 1024 bytes, were not beneficial for our STC. We believe the reason is that [5] classifies different applications, most of them running over HTTP such as Facebook and Google, while we aim in classifying service types, such as HTTP or MySQL. Thus, for us the relevant information can be found in the first few bytes of the payload, which holds the header information of a service type, e.g., the HTTP or MySQL header. Larger payloads will have a larger portion of application-specific data which is good for application identification, yet misleading for service type identification.

Flow direction: Given a pair of communicating endpoints (e.g., a client and a server), a bidirectional flow contains the sequence of the packets as they are exchanged in both directions. In contrast, unidirectional flows contain only the packets that go in one direction, and there are typically two such unidirectional flows for each endpoint pair. Unidirectional flows are guaranteed to have the packets that belong to a single service request or response one after the other in the sequence, while in bidirectional flows they might be interleaved with messages that travel in the other direction. Bidirectional flows might better reflect the timing in handshake protocols at the beginning of a connection or can better correlate requests and responses. In the literature, [5] only construct unidirectional flows and [3], [4] only construct bidirectional flows. In contrast, we have created both unidirectional and bidirectional datasets for both our header- and payload-based approaches.

Position of packets in the flow: In the published work both for header- and payload-based datasets, the N packets taken are always extracted from the beginning of the connection. Thus, the flow contains the packets of the handshake protocol to set up the connection. However, the service type identification might become necessary at a random time after establishing a connection, and when sniffing the network packets to create the flow only happens at that time, then the flow does not contain these handshake messages. As this might influence the STC performance, our default datasets contain the first messages exchanged for a connection while derived datasets do not contains these first messages.

IV. Deep Learning Models

We have chosen the two deep-learning models that have shown to work best for STC.

***- Multi-layer Long Short Term Memory (LSTM)*:** is well-suited for time-series data. LSTM utilizes circulation structure to reflect previous learning data into the current ones for sequential data learning. According to [5], the three-layer LSTM model architecture provides best application identification results. The flow-based datasets are applied to the input layer of the three-layer LSTM model, and processed by LSTM cells sequentially until a final classification result is produced. We refer to this model as 3-LSTM.

***- Combined CNN and LSTM model*:** This architecture integrates convolutional neural network (CNN) and LSTM. The feature maps of input data are first extracted through the convolution layer, and then used as a refined sequential data input to the LSTM model. CNNs are commonly used for image classification. A kernel (filter) action is used to automatically produce feature maps by extracting location invariant patterns from the image. The matrix formed by the time-series of payload data or the feature vectors of header information, described in Section III, can present a correlated local behavior, similar to images, and enable adaption of these models in the context of network traffic classification [3].

According to [3], chaining two CNN layers and one LSTM layer achieves good classification performance. Chaining several CNNs allows automatic extraction of complex features from the input datasets. A reshaping process is then performed on the output of the last convolution layer before passing it to the LSTM layer. The feature maps produced by the CNN layer will be processed by LSTM layer cells sequentially, and a final classification result is produced. We refer to this model as CNN+LSTM.

V. Experimental Evaluation

This section reports the STC performance for the header-based and payload-based datasets discussed in Section III, while applying the deep learning models of Section IV.

A. Validation Environment and Datasets

The training of the service type identification models is performed on a Ubuntu 16.04 LTS machine with 64GB RAM and two GPU cards (NVIDIA GTX 1080Ti 12GB), using Keras 2.3.1 with a TensorFlow-gpu 2.1 backend, operated with Python 3.7.7. We use the sequential model-based optimization SMBO [9] to find the optimal hyper-parameters for the aforementioned deep learning models with respect to our datasets. In addition, we have validated the produced hyper-parameters by K-fold cross-validation. In particular, we separate each flow-based dataset into learning and test datasets. Further, we separate 20% of the learning data for validation, and the verification of the model is performed on the basis of the given SMBO hyper-parameter set and the k-fold value. Due to space constrains, we provide the details and values of those hyper-parameters for each dataset-model pair in [10].

Datasets, Services and Applications: Having a traditional service architecture in mind, we have aimed at having classical service types such as HTTP-based services, database systems, and caching services in our repertoire. We also wanted to see how good the STC is if the services are conceptually very similar. Thus, apart of HTTP, we have included four relational database systems (DB2, MySQL, PostgreSQL, MonetDB), the distributed NoSQL database system Cassandra, and two key-value caches (Redis and Memcached). Furthermore, we have included the distributed compute platform Spark as an example of a distributed data processing service commonly used in the cloud. For each of them, we have collected traces for various applications. A summary is provided in Table I which also indicates which applications are used for training and which for testing. The TeaStore benchmark [11], the YCSB benchmark [12], and three small-scale in-house developed applications are used to generate traces for HTTP, the database systems and the caching services. For HTTP traffic, we additionally use the dataset provided by the UPC's Broadband Communications Research Group [13]. The caching systems use different formats for data storage in order to evaluate the capability of deep learning models of recognizing the cache service despite the different data format. For Spark traces, we run several Spark-Bench workloads [14]. In total, the dataset contains around 20,000 unidirectional network flows with 10 distinct labeled services [10]. HTTP flows made up around 44% of all our flows, less than 1% for Spark and the data management flows between Cassandra server nodes. The rest is equally distributed among the other service types.

B. Performance Metrics

Our setup is, in principle, a multi-class classification problem. For each service, we determine the correctly labeled flows (correctly labeled as belonging to the service or not belonging to the service), and the not correctly labeled flows. From there we model performance in terms of accuracy, recall, precision and F1-score for each dataset-model pair. The F1-score represents the weighted average of the precision and

TABLE I: Services and applications used for model training and testing (AL: ArrayList, LL: LinkedList, JS: Json String).

	Teastore	YCSB	UPC's dataset	Netflix	University	Venues
HTTP	Test.	Test.	Train (67%)/ Test (33%).	-	-	-
PostgreSQL	-	Train/Test.	-	Test.	Train	Train
MySQL	Test.	Test.	-	Test.	Train.	Train.
DB2	-	-	-	Test.	Train.	Train.
MonetDB	-	-	-	Test.	Train.	Train.
Cassandra Multiple nodes		Train/Test.	-	Test.	Train.	Train.
Memcached	-	Train/Test.	-	Test(all data formats).	Train (AL) /Test(LL & JS).	Train (AL) /Test(LL & JS).
Redis		Train/Test.	-	Test(all data formats).	Train (AL) /Test(JS).	Train (AL) /Test(JS).

TABLE II: Classification performance aggregated metrics vs. network models for header-based and payload-based datasets.

	Header-based datasets							
	Unidirectional flows				**Bidirectional flows**			
	Accuracy	**F1-Score**	**Recall**	**Precision**	**Accuracy**	**F1-Score**	**Recall**	**Precision**
3-LSTM [5]	0.525	0.480	0.525	0.557	0.782	0.796	0.782	0.861
CNN+LSTM [3]	0.490	0.444	0.490	0.502	0.808	0.818	0.808	0.864
	Payload-based datasets							
	Unidirectional flows				**Bidirectional flows**			
	Accuracy	**F1-Score**	**Recall**	**Precision**	**Accuracy**	**F1-Score**	**Recall**	**Precision**
3-LSTM [5]	0.940	0.940	0.940	0.955	0.937	0.937	0.937	0.952
CNN+LSTM [3]	0.942	0.942	0.942	0.954	0.948	0.949	0.948	0.958

recall, and offers a more accurate indication of the classification performance in particular when the dataset is unbalanced. For some of the results, we show the performance aggregated over all services using a weighted average calculated using scikit-learn. We have run each test five times and report the average. The standard deviations are always below 0.1.

C. Experiment Results

1) Impact of Network Model: Table II shows the aggregated service type identification performance for the 3-LSTM and CNN+LSTM deep learning models described in Section IV. The table shows results for header-based and payload-based datasets with unidirectional and bidirectional flows with the packets extracted from the beginning of the connection.

a) Header-based dataset: Using unidirectional flows for the header-based dataset achieves a maximum of around 52% accuracy and 48% F1-score by the 3-LSTM model while the bidirectional flows improve the STC performance to be higher than 78% accuracy and 79% F1-score. It seems that the correlation between the incoming and outgoing messages for a service type is crucial for better recognition of the service type in the header-based dataset. While CNN+LSTM slightly outperforms 3-LSTM for the bidirectional header-based dataset, 3-LSTM performs slightly better for the unidirectional one.

As an extra experiment, we have evaluated the impact of adding source and destination port numbers as extra learning features for the header-based dataset. We consider two types of test data; the first contains the same service port numbers as the training dataset, while the second has alternative standard port numbers for each service, e.g., 8079 instead of 8080 for HTTP and 33060 instead of 3306 for MySQL. We run the two tests against CNN+LSTM. The first test data results in 93.8% accuracy and 93.6% F1-score, increasing performance compared to when we ignored port numbers. In contrast, using alternative port numbers for the services decreases the identification accuracy and F1-score significantly to around 60% and 57%, respectively. This shows that the performance of header-based deep learning models that consider the service port numbers is tightly correlated to the static service configuration.

b) Payload-based dataset: Table II shows that the classification performance of unidirectional and bidirectional flows in the payload-based dataset is quite similar for both models with CNN+LSTM performing slightly better for bidirectional flows. Overall, the performance on the payload-based datasets is significantly better than for header-based datasets.

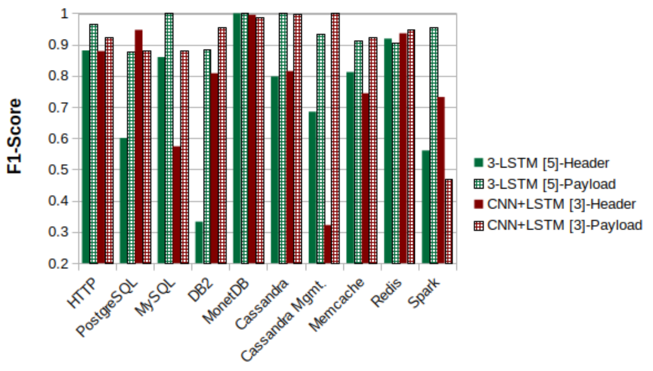

Fig. 1: F1 score for various service types vs. network models for both payload and header-based datasets.

2) Performance on a per-service basis: Having a closer look at the performance for the individual services, Figure 1 shows the F1-score for each service label achieved by both models for the bidirectional payload-based and header-based datasets. The F1-score for the payload-based dataset is high for both models and above 87% for nearly all services. Only exception is the poor performance of CNN-LSTM for Spark.

We had a closer look at the confusion matrices (not shown in a figure), and observed some misclassifications between DB systems. For example, 20% of PostgreSQL flows were misclassified as DB2 by the 3-LSTM model, while the same percentage of PostgreSQL flows were misclassified as MySQL by the CNN+LSTM model. We found a similar pattern for Memcached and Redis. For instance, the CNN+LSTM model misclassified around 10% of Redis cases as MemCached while the 3-LSTM model misclassified 15% of Memcached flows as Redis. In fact, Memcached and Redis requests have a fair amount of similarity in their wording for commands such as GET, SET, APPEND; and also in their reply messages. While still being a wrong classification, labeling a service as being of the "same kind" might be viewed as more acceptable than classifying it as something completely different.

For the header-based dataset, the confusion matrices show a lot more variation. Misclassifications happen across all services with no clear preferences for services that are similar. It appears that the header data does not reveal too much commonality among services of the same kind.

3) The impact of the packet positions in the flow: We now want to look at the influence that the messages exchanged at the start of the connection have on prediction performance. We only look at bidirectional flows and CNN+LSTM as they tend to have better performance overall. For the header-based dataset, we consider the versions with and without port

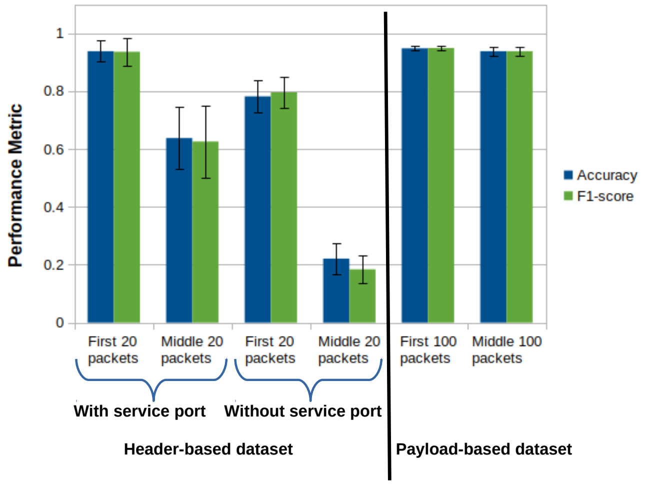

Fig. 2: The impact of the extracted packets position in the flow in both header-based and payload-based datasets.

TABLE III: Training time (in seconds) of different deep learning models for header-based and payload-based datasets.

	Header-based dataset		Payload-based dataset	
	Unidirectional flows	**Bidirectional flows**	**Unidirectional flows**	**Bidirectional flows**
3-LSTM [5]	89.383	42.001	178.505	156.567
CNN+LSTM [3]	177.276	55.714	286.817	130.528

numbers. Figure 2 shows the aggregated accuracy and F1-score for both header-based and payload-based datasets when the 20 resp. 100 packets were taken from either the start or the middle of the network flows. In the header-based dataset with port numbers, not having the first messages of a flow decreases the F1-score by 33%, while around 76% F1-score loss happens in the header-based dataset that excludes the port numbers. This shows that header-based approaches in scenarios with dynamic ports can only work reasonably well if they can learn the service type's flow characteristics from the first handshaking packets between the service and client. Once the model misses extracting those first packets from the service flow, it wouldn't be able to recognize the service type anymore. But even if standard ports are used and the model learning considers them, not having access to the handshake messages significantly reduces performance. On the other hand, the payload-based dataset performance is only slightly affected when skipping the first packets in the flow. This is because the model can infer the service type from the service type headers in the request and response messages.

4) Training time: Table III shows the training times for the 3-LSTM and CNN+LSTM models on the different datasets. The training time grows with the increase in the number of learning features (header-based has 4 compared to 36 in payload-based), flows in the training dataset (unidirectional has double as many flows as bidirectional) and model complexity. For the latter, the number of model layers and their type influence the model training time. While 3-LSTM and CNN+LSTM have the same number of layers, the latter requires more model training time for most of the datasets. For instance, for the unidirectional payload-based dataset, which has many learning features, the training time of CNN+LSTM is around 60% higher than that of 3-LSTM. However, the STC performance is improved by only 0.2%, as shown in Table II. Similarly, the CNN+LSTM model increases the STC performance for the bidirectional header-based dataset by only 2% with around 32% overhead in the training time compared to the 3-LSTM model. Thus, there is a trade-off to be made.

D. Summary

In general, payload-based approaches outperform the header-based ones with accuracy and F1-score always higher than 93%. Furthermore, for header-based approaches, having access to the first handshaking packets in the flow stream and working on bidirectional flows are both important for high service type identification accuracy. Both these things are not so important for payload-based approaches to work well. A further advantage of payload-based datasets is that if the data is mislabeled, the wrong label often belongs to a "similar" service type, e.g., another caching or database system.

VI. Conclusion

In this paper, we have provided a comprehensive study of the use of deep learning models in service type identification of network flows. We have compared the performance of RNN and a combination of RNN and CNN models, while using header-based and payload-based data for training. We have highlighted the trade-offs and the impact of various parameters on the classification performance and required model training time. Future work includes extending the scope by considering more application traces and service types, and studying how encryption and compression can be handled by deep-learning.

References

[1] M. Finsterbusch, C. Richter, E. Rocha, J. Muller, and K. Hanssgen, "A survey of payload-based traffic classification approaches," *IEEE Commun. Surv. Tutorials*, vol. 16, no. 2, pp. 1135–1156, 2014.

[2] J. Zhang, Y. Xiang, Y. Wang, W. Zhou, Y. Xiang, Y. Guan, "Network traffic classification using correlation information," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 1, pp. 104–117, 2013.

[3] M. L. Martín, B. Carro, A. Sánchez-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.

[4] W. Wang, M. Zhu, X. Zeng, X. Ye, and Y. Sheng, "Malware traffic classification using convolutional neural network for representation learning," in *ICOIN 2017, Da Nang, Vietnam*, pp. 712–717.

[5] H.-K. Lim, J.-B. Kim, K. Kim, Y.-G. Hong, and Y.-H. Han, "Payload-based traffic classification using multi-layer lstm in software defined networks," *Applied Sciences*, vol. 9, no. 12: 2550, 2019.

[6] Q. Wang, A. Yahyavi, B. Kemme, and W. He, "I know what you did on your smartphone: Inferring app usage over encrypted data traffic," in *IEEE Conf. on Communications and Network Security (CNS)*, 2015.

[7] M. Shafiq, X. Yu, and D. Wang, "Network traffic classification using machine learning algorithms," in *Advances in Intelligent Systems and Interactive Applications*. Springer, 2018, pp. 621–627.

[8] H. Singh, "Performance analysis of unsupervised machine learning techniques for network traffic classification," in *Proc. 5th Int'l Conf. on Advanced Comp. Comm. Tech., Washington, DC, 2015*, pp. 401–404.

[9] A. Thammano and P. Poolsamran, "SMBO: A self-organizing model of marriage in honey-bee optimization," *Expert Syst. Appl.*, vol. 39, no. 5, pp. 5576–5583, 2012.

[10] "Service type classifier project." 2020. [Online]. Available: https://www.cs.mcgill.ca/~kemme/disl/STC.html

[11] e. a. Jóakim von Kistowski, "TeaStore: A Micro-Service Reference Application for Benchmarking, Modeling and Resource Management Research," in *Proc. of MASCOTS '18*, 2018.

[12] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in *Proc. of SoCC'10, Indianapolis, Indiana, USA, 2010*, pp. 143–154.

[13] V. Carela-Español, T. Bujlow, and P. Barlet-Ros, "Is our ground-truth for traffic classification reliable?" in *PAM, Los Angeles, CA, USA, 2014*, ser. Lecture Notes in Computer Science, vol. 8362. Springer, pp. 98–108.

[14] M. Li, J. Tan, Y. Wang, L. Zhang, and V. Salapura, "Sparkbench: a spark benchmarking suite characterizing large-scale in-memory data analytics," *Clust. Comput.*, vol. 20, no. 3, pp. 2575–2589, 2017.