On Learning Hierarchical Embeddings from Encrypted Network Traffic

Nikolas Wehner*, Markus Ring†, Joshua Schüler*, Andreas Hotho*, Tobias Hoßfeld*, Michael Seufert*
*University of Würzburg, Würzburg, Germany, firstname.lastname@uni-wuerzburg.de
†Coburg University of Applied Sciences and Arts, Coburg, Germany, markus.ring@hs-coburg.de

***Abstract*—This work presents a novel concept for learning embeddings from encrypted network traffic. In contrast to existing approaches, we evaluate the feasibility of hierarchical embeddings by iteratively aggregating packet embeddings to flow embeddings, and flow embeddings to trace embeddings. The hierarchical embedding concept was designed to especially consider complex dependencies of Internet traffic on different time scales. We describe this novel embedding concept for the domain of network traffic in full detail, and evaluate its performance for the downstream task of website fingerprinting, i.e., identifying websites from encrypted traffic, which is relevant for network management, e.g., as a prerequisite for QoE monitoring or for intrusion detection. Our evaluation reveals that embeddings are a promising solution for website fingerprinting as our model correctly labels up to 99.8% of traces from 500 target websites.**

I. Introduction

Network operators face increasing challenges when managing communication networks, which are driven by the growing number of users, devices, applications, and their diverse requirements, as well as the consequently required increasing complexity of networks. Thus, network operators strive to monitor their networks in order to quickly detect and resolve performance or security issues. Considering performance management, the last decades saw a transition from monitoring of classical Quality of Service (QoS) metrics, such as bandwidth, latency, or packet loss, towards application-specific Quality of Experience (QoE) monitoring based on Deep Packet Inspection (DPI). For example, DPI allowed to inspect streaming flows to approximately reconstruct the client's playback buffer [1], [2], and thus, to detect imminent or ongoing video stalling, i.e., playback interruptions, which is a severe QoE degradation [3]. Such QoE monitoring allows to ensure a high service quality and a high user satisfaction, which avoids user churn in this competitive market. Also security management has to be constantly adapted, as sophistication and amount of cyber attacks against networked systems are increasing [4]. To face these threats, network operators no longer rely on static defenses only, such as firewalls, but more and more employ DPI as part of or in combination with complex monitoring-based systems, such as Intrusion Detection Systems (IDSs) [5], [6], which detect attacks based on previously defined or anomalous traffic patterns.

However, in the last years, end-to-end encrypted traffic over HTTPS has become the predominant type of traffic for all kinds of services accounting for over 85% of Internet traffic from desktop and mobile devices in 2019 [7]. This poses a huge challenge for network management, as the encryption hides the packets' content, and thus, DPI can no longer provide useful information for network operators. This means that network operators lose visibility in their own networks, and it is becoming more and more difficult for network operators to monitor performance or security.

Nevertheless, novel capabilities to monitor fine-granular, packet-level data in softwarized and virtualized networks (SDN/NFV), based on more sophisticated software packet processing technologies [8] like DPDK, or based on programmable switching hardware like P4 [9], allow for detailed, real-time monitoring of statistical properties of traffic flows. This availability of large amounts of data calls for the application of artificial intelligence (AI) and machine learning (ML) to overcome the reduced visibility of network operators [10]. For example, for the use case of monitoring the Quality of Experience (QoE) of video streaming, several approaches [11], [12], [13], [14] already managed to learn streaming behavior and the resulting QoE with a high accuracy in tested lab scenarios. However, these approaches are typically considering packet arrivals as being independently distributed, and do not fully capture the time series and sequence-like nature of network traffic flows. In addition, they often utilize handcrafted features, which require cumbersome preprocessing steps, and also potentially miss relevant information, which would be crucial for the considered or related ML downstream tasks.

To overcome this issue, we resort to learn embeddings for time series network data. Embeddings are numerical vector representations of categorical data and originate from the natural language processing domain. They are typically used to map words to vectors, which can serve as input to neural networks, and play a key role in the latest advances in ML-based text processing and understanding. However, the application to the domain of network traffic is still in its infancy. The major challenge here is that network traffic is not strictly sequential like text, but Internet services are often transmitted over multiple traffic flows at the same time, which not only might influence each other, but might also be influenced by background traffic and changing network conditions.

Figure 1 illustrates the idea behind the concept of learning embeddings for encrypted network traffic. Neglecting the IP

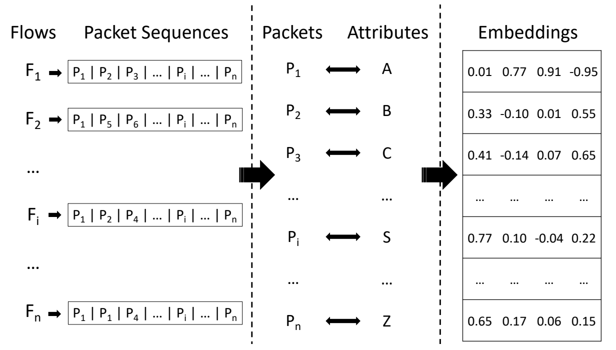

Fig. 1: Embedding concept for encrypted network traffic.

addresses and port information, we assume that the information provided in the encrypted IP packets are sufficient to estimate the purpose of a packet within a flow. We further assume that there exist packets with similar purposes in independent flows, e.g., flows F_1 and F_2 both contain packet P_1. This packet P_1 may have a specific attribute A, which could represent a DNS request or a TCP packet with enabled SYN flag. The goal of embeddings is to convert these instances P_1-P_n or their corresponding attributes A-Z, respectively, to representative continuous vectors (embeddings), which capture the contexts and relations of the packets within the flows, and thus, improve the performance for a ML downstream task.

In this work, we survey and discuss existing embedding approaches for network traffic, and present a novel concept for learning embeddings for encrypted network traffic. In contrast to existing approaches, which learn embeddings for single IP addresses or packets, we take into account that sequences must be considered for analyzing network traffic. Therefore, we evaluate the feasibility of a hierarchical embedding by iteratively aggregating packet embeddings to flow embeddings, and flow embeddings to trace embeddings. Thereby, a packet embedding is the learnt representation for one network packet and a flow embedding corresponds to the representation for one bidirectional flow. Finally, we define a trace embedding as the representation of all flows of one source IP address in a certain time window. This hierarchical embedding concept is supposed to especially consider complex dependencies on different time scales, which is a well-known property of Internet traffic [15]. We describe this novel embedding concept in full detail, and evaluate its performance for the downstream task of website fingerprinting, i.e., identifying the websites from encrypted traffic, which is both relevant for QoE monitoring as well as for security monitoring.

This paper is structured as follows. Section II discusses related works on learning embeddings for network traffic. Section III presents our novel concept of hierarchical network traffic embeddings. The evaluation methodology for the website fingerprinting task is described in Section IV, and the evaluation results are presented in Section V. Section VI concludes this work and provides an outlook to future works.

II. Related Work

This section reviews related work on learning embeddings for network traffic and their application in downstream tasks like QoE monitoring and intrusion detection.

Embeddings are vector representations for non-numerical values and originate from NLP. Typical and widely used approaches include BERT [16], FastText [17], GloVe [18] and Word2Vec [19]. These methods usually rely on some kind of neural networks or co-occurrence matrix in order to learn vector representations for words. Meanwhile, embeddings are widely used in other domains such as medicine [20] or music [21], [22].

There are some approaches which transfer the idea of learning embeddings to the area of network traffic. A fairly simple adaptation of Word2Vec to flow-based network data is presented by Henry [23]. This approach is called Flow2Vec and learns an embedding for each flow. Thereby, a flow is represented by its IP addresses, ports, bytes, and packets, which leads to a huge number of different values and raises problems in classification scenarios when new values are occurring. In contrast to that, IP2Vec [24] aims to transform categorical values of flow-based network traffic like IP addresses or port numbers into a continuous feature space $\mathbb{R}^m$. Thereby, similar IP addresses are mapped to similar vector representations. Similarities are defined by extracting available context information of flow-based network traffic. The authors evaluate IP2Vec in the context of clustering and visualization and achieve promising results. Fei et al. [25] further developed this approach and use it to classify servers according to their offered services.

Similar to IP2Vec, Goodman et al. [26] adapt the idea of Word2Vec to packet-based network traffic. Then, the authors evaluate their method Packet2Vec to identify malicious network traffic based on a 2009 DARPA network data set. Mimura and Tanaka [27] also learn embeddings for network traffic by adapting Doc2Vec [28] to packet-based network data. The authors use the header attributes transport protocol, ports, and size of packets, and group 100 packets as one paragraph. Then, a continuous vector representation is learned for each paragraph and used for classification tasks afterwards.

Henry [23], Ring et al. [24] and Goodman et al. [26] learn embeddings for network characteristics like IP addresses, ports, or single flows. In contrast to them, our approach learns embeddings for sequences of network traffic like [27]. While the authors of [27] learn embeddings for sequences of packets, we also consider hierarchical information by extracting knowledge from flows and traces.

III. Hierarchical Network Traffic Embedding

Next, we present our novel approach on learning hierarchical network traffic embeddings. The overall architecture of our approach is presented in Figure 2. The figure depicts the three embedding modules, namely, packet embedding, flow embedding, and trace embedding, along with the downstream task module. The embedding modules have a hierarchical dependency and the embedding process starts on packet level. Our

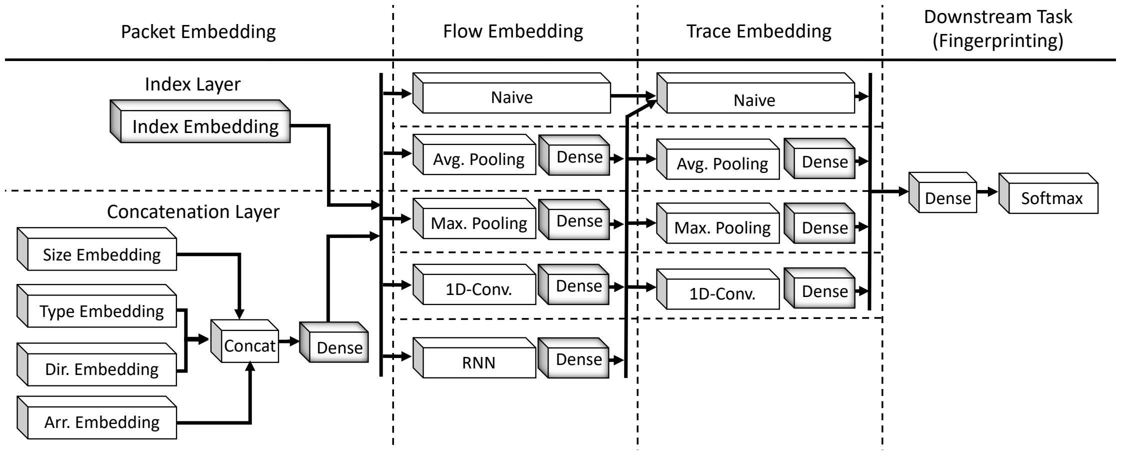

Fig. 2: Neural network architecture of the hierarchical embedding approach.

idea is based on the assumption that hierarchical embeddings best imitate the communication structures of networks [15]. In a preprocessing step, relevant information from encrypted network packet headers is extracted. Then, by passing this information through the packet embedding module we obtain embeddings on packet level. These packet embeddings are then passed on to the flow embedding module, where flow embeddings are learnt based on the packet embeddings. Finally, the flow embeddings again are aggregated to trace embeddings to represent a group of flows. Dashed horizontal lines indicate alternatives for learning embeddings which means, e.g., that either the Index Layer or the Concatenation Layer can be used for packet embedding. Shaded blocks highlight the modules where the individual embeddings are generated. All kinds of embeddings are learned end-to-end, which means that the embeddings are learnt along with the weights required for the downstream task. In this work, we evaluate the impact on fingerprinting accuracy when using the various embeddings.

In the following, the preprocessing of network data and the individual modules of our architecture are discussed in detail.

A. Preprocessing

Given a PCAP file, we perform a packet and flow preprocessing. This includes the extraction of IP packet attributes, e.g., packet size and packet direction, and indexing the packets based on their attributes. Additionally, we extract flow-based information like flow IP addresses and ports, as well as the unique flow tuples. We consider flows identical when they share the same source and destination IP address, the same source and destination port, and the same protocol.

Based on this information the input data is built for our model. For each unique packet tuple and each unique flow tuple, we define a vocabulary, which maps the unique tuples to integers. Finally, we define the maximum number of flows and the maximum number of packets, which we consider for the embedding. To keep the input size concise, we utilize only the top ten flows, sorted by the total download volumes in the trace, and the first 500 packets of each flow to build the input. Runs with less than ten flows or flows with less than 500 packets are padded with zeros, where zero corresponds to the padding token in our vocabulary. We further remove anomalous traces with an insufficient amount of flows (< 5). We chose these numbers to trade off heavy flows with many packets and light flows with a few packets. This is supposed to avoid many flows with too much padding, but also to preserve the characteristic patterns of a network trace .

We then stack these data to obtain input of shape (FN, PN, AN), where FN corresponds to the maximum number of flows, PN is the maximum number of packets, and AN represents the number of packet attributes.

B. Packet Encoding

We define a packet as the four tuple $P = \{S, T, D, A\}$, where S represents the packet size, T represents the packet type, D represents the packet direction, and A represents the arrival time of a packet within a flow.

Size The packet size S is extracted from the IP header and is limited from 0 to 1500 bytes (larger packets are clipped to 1500 bytes). Packets with size 0 are usually connection-oriented control packets containing for example TCP flags.

Type A packet is described by its purpose, which we denote with the type T. A type hereby includes the protocol of a packet along with its enabled flags, in particular we distinguish only between TCP, UDP/QUIC, and DNS as they are easily distinguishable. For TCP segments, the type includes the TCP flags concatenated in canonicalized form, e.g., $\_F\_ACK|FIN$ denotes a TCP packet with active ACK and FIN flags. UDP packets do not have any flags and are thus represented by the empty symbol. Also, DNS packets can only be queries or responses and we thus distinguish them with $\_DNS\_Q$ and $\_DNS\_R$.

Direction The direction D of a packet states whether the packet is an upload or download packet from the perspective of the host and is represented by -1 and 1, respectively.

Arrival Time The relative arrival time A denotes the position, or more specifically the normalized time of the packet in a flow relative to the first and last packet. This means that the arrival time ranges from 0 (first packet) to 1 (last packet) and that for example a packet at 0.5 corresponds to a packet occurring at half of the flow duration. To keep the space of arrival times concise, we round the normalized arrival time to the second decimal, resulting in 101 possible values.

C. Packet Embedding

In this work, we test two different layers for learning packet embeddings, which we call Index Layer (IL) and Concatenation Layer (CL) in the following. Both layers are presented in Figure 2 and differ in the following way.

Index Layer With the IL we consider each packet P (defined by the four tuple $\{S, T, D, A\}$) as an instance and the goal is to learn an embedding for each unique instance of P. The IL is realized by mapping the packet encodings P to a corresponding vocabulary, where each unique packet P is represented by an integer. This vocabulary is then used in a lookup table to learn an embedding for each packet along with the downstream task. One disadvantage with this approach is that it usually generalizes poorly, since there are no entries for unknown combinations in the vocabulary. We will discuss this problem later in Section V.

Concatenation Layer In contrast, the CL is supposed to learn specific embeddings for each attribute (S, T, D, A) which compose a packet P, before concatenating the learnt embeddings and passing them through a dense layer to obtain the packet embedding for P. For each attribute, we also utilize lookup tables to learn the embeddings for each potential attribute instance. The length of the four embeddings and the number of units of the final dense layer are hyperparameters.

D. Flow Embedding

Based on these packet embeddings, the flow embeddings are generated. The input shape for learning a flow embedding is now (PN, PE), where PN represents the maximum number of packets in a flow and PE represents the dimensionality of the learnt packet embeddings.

There exist various possibilities to train a flow embedding from this input. Here, the main goal is to aggregate packet embeddings to a flow embedding, i.e., we have to reduce the dimensionality by aggregating over the dimension PN. However, during the aggregation we need to preserve the characteristics of the packet streams as best as possible to avoid information loss. The aggregation can be achieved for example with pooling strategies, convolutional neural networks (CNNs), recurrent neural networks (RNNs), or simply concatenating the vectors. We thus test 1D average/maximum pooling, 1D CNNs, and RNNs as flow embedding modules.

Naive The simplest way, however, would be to flatten all packet embeddings, thus using the concatenated packet embeddings without aggregation as flow embedding, and then passing it to the next level of the hierarchy. Note that when we use the naive approach for flow embeddings, we automatically also take the naive approach for trace embeddings.

Pooling For the pooling mechanisms, we aggregate the packet embeddings of a flow by maximizing or averaging over all packet embeddings of a flow.

CNN As is common practice, we can utilize a 1D CNN to process time series data. Therefore, we consider each packet as a timestep and each packet embedding dimension as a channel. This output is then again passed through a 1D maximum pooling layer to provide the embedding.

RNN Finally, we utilize recurrent neural networks, in particular Gated Recurrent Unit, where we also consider each packet as a timestep and where the learnt packet embeddings correspond to the features of a timestep.

After obtaining our flow embedding FE by performing one of the steps above for each of the ten largest flows within a trace, we obtain data of the form (FN, FE) and then pass it on towards the trace embedding module. Note that FE can have a different embedding dimension than PE.

E. Trace Embedding

As a last step, the flow embeddings are transformed to trace embeddings. Again, we have to reduce the dimensionality by one and can thus apply similar techniques as above.

Naive All flow embeddings are flattened as before without aggregation and then passed forward to the downstream task.

Pooling To obtain the trace embedding using pooling, we again average or maximize all flow embeddings of a trace.

CNN We use a 1D CNN, where FN and FE represent the timesteps and features, respectively.

F. Downstream Task

In the downstream task, the embeddings are used as input and the actual website fingerprinting is performed. Website fingerprinting is a multi-class classification task, which means that the model must predict which website is accessed within the encrypted network trace. In this work, we consider 500 websites as targets, so the output layer for this task is a dense layer with 500 neurons. This is followed by a softmax activation, which computes the probability of a sample belonging to a website. In addition to the output layer, we use an additional dense layer of arbitrary size, which is responsible for extracting the relevant features from the passed embeddings beforehand.

IV. Dataset

Our dataset was collected to gather web performance metrics on different devices over multiple websites while experiencing varying network conditions. In a measurement testbed, WebpageTest (WPT)[1] was used to visit the top 500 Alexa websites with the Chrome browser. Three devices, namely a smartphone (Google Pixel 2XL), a tablet (Samsung Galaxy Tab S5e), and a desktop computer, were used as clients, each of which connected to the Internet through a separate computer to act as a network emulator and traffic capture device. Every page was visited at least ten times on each device for a total of 13 different network conditions, either applying an additional one-way delay of 10ms, 50ms, 100ms or 200ms, adding a packet loss 0.1%, 0.5%, 1%, 2% or 10%, or limiting the throughput to 0.5, 2, 5 or 10 MBit/s, resulting

[1]https://www.webpagetest.org/

Hyperparameters	Options
Optimizer	Adam
Layer Activation	[ReLU, Tanh]
Packet Embedding Dimension	[16, 32]
Flow Embedding Dimension	[32,64]
Trace Embedding Dimension	[64, 128]
Pooling Sizes	[4x4, 6x6]
Kernel Sizes	[4x4, 6x6]
Filters	[16, 32]
Stride	1
Recurrent Units	32
Downstream Units	1024
Batch Size	[16, 32]
Epochs	30

TABLE I: Overview on tested hyperparameters.

in a dataset containing over 200.000 runs. This dataset is highly challenging in that it requires the trained models to not only identify a broad range of diverse websites, such as *youtube.com* and *amazon.com*, but also to distinguish between very similar websites, such as *google.com* and *google.es*. After performing the packet and flow preprocessing for our dataset (described in Section III), we observe 739435 different packet instances and 512 different packet types in total for our dataset.

V. Evaluation

Now, the performance of our novel embedding approach for website fingerprinting is evaluated and the influence of the embedding space on the model's performance is investigated.

Tensorflow and Keras have been used for all evaluations. The data was split into training, validation, and test set, whereby the validation set and test set contained at least one sample of each website and test condition, i.e., we consider a closed world scenario, in which the accessed websites are known beforehand. After cleaning the data and removing broken runs, the training set comprises 175904 samples, the validation set contains 17312 samples, and the test set consists of 18304 samples distributed over 500 websites.

Using categorical cross entropy as loss function, we train each model combination of our hierarchical embedding approach for a maximum of 30 epochs. We also add early stopping with a patience of five epochs to reduce training time.

A. Hyperparameter Tuning

For the various models, we perform hyperparameter tuning using grid search implemented in Optuna[2]. Table I presents an overview on the tested hyperparameters. We use the the Adam optimizer with a learning rate of 1e-4. We test two different embedding dimensions for each level on the hierarchy and several pooling, kernel, and filter sizes for the pooling and convolutional mechanisms to consider different scales. The downstream units, i.e., the number of units of the last hidden layer, were fixed at 1024. The batch size is either 16 or 32.

B. Performance Evaluation

Table II depicts the results for our performance evaluation of the hierarchical embedding approach. We utilize accuracy, i.e., the amount of correctly identified websites, on the test set as performance metric. We show only the results of the best performing model with respect to the hyperparameters. The most relevant accuracies, which are discussed in the following, are illustrated in bold. It can be observed that the best performance for each packet embedding layer was obtained when using the naive approach for both flow embedding and trace embedding. Both the IL and the CL offered an almost perfect accuracy of 0.998 and 0.997, respectively, thus misclassifying only up to 55 out of the 18304 samples of the test set. The second best results for both IL and CL were obtained when using CNNs for computing the flow embeddings and then naively flattening the trace embedding. The performance of the IL dropped only slightly from 0.998 to 0.995, while the performance of the CL did not change at all. For both layers, these results indicate that CNNs are suitable solutions for flow embeddings, while the usage of pooling mechanisms and RNNs stronger decrease the classifier's performance for this task due to information loss. Further, we observe a performance drop when additionally utilizing more sophisticated trace embeddings. For both IL and CL, the combination of CNN as flow embedding and as trace embedding shows very good results, again suffering from a marginal performance drop only.

This shows that both IL and CL are suitable for learning meaningful packet embeddings. From analyzing the sophisticated flow and trace embeddings, we could observe that CNNs are the most suitable for learning the characteristic patterns in encrypted network traffic. Moreover, the usage of RNNs should be avoided for learning embeddings since they are

Packet Embedding	Flow Embedding	Trace Embedding	Test Accuracy
IL	NAIVE	NAIVE	**0.998**
	AVG	NAIVE	0.771
		AVG	0.611
		MAX	0.769
		CNN	0.918
	CNN	NAIVE	**0.995**
		AVG	0.892
		MAX	0.987
		CNN	0.982
	MAX	NAIVE	0.961
		AVG	0.741
		MAX	0.723
		CNN	0.935
	RNN	NAIVE	0.969
		AVG	0.714
		MAX	0.002
		CNN	0.943
CL	NAIVE	NAIVE	**0.997**
	AVG	NAIVE	0.769
		AVG	0.549
		MAX	0.698
		CNN	0.845
	MAX	NAIVE	0.960
		AVG	0.827
		MAX	0.887
		CNN	0.963
	CNN	NAIVE	**0.997**
		AVG	0.829
		MAX	0.963
		CNN	0.981
	RNN	NAIVE	0.863
		AVG	0.802
		MAX	0.791
		CNN	0.942

TABLE II: Best results of performance evaluation.

[2]https://optuna.readthedocs.io/en/stable/index.html

slow to train and thus slow to converge. Finally, the pooling mechanisms (AVG/MAX) proved to perform badly in general.

To sum up, the evaluation showed the potential of embeddings for ML-based tasks in encrypted network traffic, providing a high accuracy of up to 99.8% for the task of website fingerprinting. For this simple use case, packet embeddings and naive flow and trace embeddings were sufficient to obtain high accuracy and it could be observed that accuracy decreased with additional abstraction in form of more sophisticated, aggregating flow and trace embeddings. Nevertheless, flow and trace embeddings may become very relevant for other, more complex use cases, e.g., for identifying malicious flows in a network or predicting the QoE of a multi-flow web application. Another benefit of flow embeddings is that they could possibly be computed independently in deployment scenarios with memory constraints. Thus, further research is necessary on how to best preserve the information on the various levels of the hierarchy.

C. Impact of Embedding Space

One drawback of the IL model is the large space of possible packet manifestations P (characterized by S, T, D, A), which may result in observing unknown packets, i.e., packets without pretrained embedding, during deployment. This problem could be solved by creating a default embedding for unknown values [24], which may not be optimal for many use cases.

With our current implementation, the space of potential packet manifestations is composed the following way. The size S has 1501 manifestations, the direction D has two manifestations, and the arrival time A has 101 manifestations. For the type T, there are nine TCP flags [29], i.e., a maximum of $2^9 = 512$ TCP flag manifestations, two DNS manifestations (request and response), and a single manifestation for UDP packets, which covers the increasingly used QUIC protocol. All in all, this results in a maximum of 515 possible type manifestations for our use case. This number would increase if additional protocols would be considered in other scenarios.

We observe that the packet size S is the limiting factor in this factorial design. Further, the accuracy of the arrival time A likely does not influence the model performance strongly. Therefore, we first strongly reduce the potential packet size manifestations by uniformly binning the packet sizes into eleven manifestations (starting with 0 in steps of 150). Secondly, we reduce the manifestations of A by rounding to the first decimal (not to the second decimal as before), also resulting in eleven manifestations. Subsequently, performing a factorial design for S, T, D, and A results in a maximum of 124630 possible manifestations for P, which is a rather small embedding space. After reducing the embedding space, we observe only 3335 manifestations in our dataset (instead of 739435 before). Similar to the IL, we also limit the vocabulary size of S and A for the CL with the reductions of above.

We then again evaluate the model performance using IL and CL and the compressed embedding space. Therefore, we use the best performing IL and CL models and train the models again for 30 epochs. We observe an accuracy of 99.1% for IL, which is only slightly worse than the model with the original space (99.8%), and an accuracy of 98.7% for CL, which is also slightly worse. This indicates that both the IL and CL with a limited embedding space would also be proper solutions for online deployment. Note, however, that the embedding space compression could have a larger negative effect for other downstream tasks, e.g., where packet sizes play a more important role than for website fingerprinting.

D. Deployment Considerations

When considering to deploy our approach in a productive setting, our embedding approach would additionally require to identify web browsing sessions in order to know which flows belong together to a certain website. As one possible and easy-to-implement mitigation, flows could be grouped together based on client IP address, assuming that clients typically only request one website at a time. This simple heuristic should provide sufficient accuracy most of the time. If a higher precision is required, additional preprocessing steps could be added later, e.g., performing a temporal clustering on large network traces to learn co-occurring flows for a websites.

VI. Conclusion and Outlook

In this paper, we presented a novel concept for learning hierarchical network traffic embeddings to especially consider complex dependencies within encrypted network traffic on different time scales. For this, our approach iteratively aggregates packet embeddings to flow embeddings, and flow embeddings to trace embeddings. We therefore tested two different packet embedding approaches and several flow and trace embedding aggregation techniques. We evaluated its performance for the downstream task of website fingerprinting using 500 different websites as targets.

First, we observed high accuracies of up to 99.8% for specific models. This shows that embeddings for encrypted network traffic provide a promising research direction in general. With respect to the hierarchical design, we showed that the best performance is obtained when using packet embeddings together with naive flow and trace embeddings. Both utilized packet embeddings offered high accuracy, even when the embedding space was compressed strongly. Additional abstractions by more sophisticated flow and trace embeddings seemed to result in information loss, which was detrimental for the website fingerprinting performance, but could nevertheless become very relevant when considering other, more complex downstream tasks.

In future work, we plan to extend our approach of hierarchical network traffic embeddings by additionally considering flow information. Moreover, we want to train and evaluate our approach on larger data sets and for more applications, such as QoE monitoring and intrusion detection. In the best case, a network traffic embedding should also be designed in a way that it can be applied to multiple downstream tasks without having to train a specific network embedding for each downstream task. This again would require pre-trained network traffic embeddings.

References

[1] B. Staehle, M. Hirth, R. Pries, F. Wamser, and D. Staehle, "YoMo: A YouTube Application Comfort Monitoring Tool," in *1st Workshop of Quality of Experience for Multimedia Content Sharing (QoEMCS)*, 2010.

[2] P. Casas, M. Seufert, and R. Schatz, "YOUQMON: A System for On-line Monitoring of YouTube QoE in Operational 3G Networks," *ACM SIGMETRICS Performance Evaluation Review*, vol. 41, no. 2, pp. 44–46, 2013.

[3] M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Hoßfeld, and P. Tran-Gia, "A Survey on Quality of Experience of HTTP Adaptive Streaming," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 469–492, 2015.

[4] Check Point Research, "Cyber Security Report 2020," Check Point Research, Tech. Rep., 2020, accessed: 2021-05-18. [Online]. Available: https://www.checkpoint.com/downloads/resources/cyber-security-report-2020.pdf

[5] N. Sultana, N. Chilamkurti, W. Peng, and R. Alhadad, "Survey on sdn based network intrusion detection system using machine learning approaches," *Peer-to-Peer Networking and Applications*, vol. 12, no. 2, pp. 493–501, 2019.

[6] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, "Survey of intrusion detection systems: techniques, datasets and challenges," *Cybersecurity*, vol. 2, no. 1, pp. 1–22, 2019.

[7] NetMarketShare, "HTTP vs HTTPS," NetMarketShare, Tech. Rep., 2021, accessed: 2021-08-23. [Online]. Available: https://netmarketshare.com/report.aspx?id=https

[8] D. Cerović, V. Del Piccolo, A. Amamou, K. Haddadou, and G. Pujolle, "Fast packet processing: A survey," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 3645–3676, 2018.

[9] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[10] R. Boutaba, M. A. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano, and O. M. Caicedo, "A comprehensive survey on machine learning for networking: evolution, applications and research opportunities," *Journal of Internet Services and Applications*, vol. 9, no. 1, pp. 1–99, 2018.

[11] V. Aggarwal, E. Halepovic, J. Pang, S. Venkataraman, and H. Yan, "Prometheus: Toward Quality-of-Experience Estimation for Mobile Apps from Passive Network Measurements," in *Proceedings of the 15th Workshop on Mobile Computing Systems and Applications (HotMobile)*, Santa Barbara, CA, USA, 2014.

[12] G. Dimopoulos, I. Leontiadis, P. Barlet-Ros, and K. Papagiannaki, "Measuring Video QoE from Encrypted Traffic," in *Proceedings of the ACM Internet Measurement Conference (IMC)*, Santa Monica, CA, USA, 2016.

[13] I. Orsolic, D. Pevec, M. Suznjevic, and L. Skorin-Kapov, "A Machine Learning Approach to Classifying YouTube QoE Based on Encrypted Network Traffic," *Multimedia Tools and Applications*, vol. 76, no. 21, pp. 22 267–22 301, 2017.

[14] M. Seufert, P. Casas, N. Wehner, L. Gang, and K. Li, "Stream-based Machine Learning for Real-time QoE Analysis of Encrypted Video Streaming Traffic," in *3rd International Workshop on Quality of Experience Management*, 2019.

[15] A. Feldmann, A. C. Gilbert, and W. Willinger, "Data networks as cascades: Investigating the multifractal nature of internet wan traffic," *ACM SIGCOMM Computer Communication Review*, vol. 28, no. 4, pp. 42–55, 1998.

[16] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, (NAACL)*. Association for Computational Linguistics, 2019, pp. 4171–4186.

[17] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 135–146, 2017.

[18] J. Pennington, R. Socher, and C. D. Manning, "GloVe: Global Vectors for Word Representation," in *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543.

[19] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *arXiv preprint arXiv:1310.4546*, 2013.

[20] E. Choi, M. T. Bahadori, E. Searles, C. Coffey, M. Thompson, J. Bost, J. Tejedor-Sojo, and J. Sun, "Multi-layer representation learning for medical concepts," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 1495–1504.

[21] H. Liang, W. Lei, P. Y. Chan, Z. Yang, M. Sun, and T.-S. Chua, "Pirhdy: Learning pitch-, rhythm-, and dynamics-aware embeddings for symbolic music," in *Proceedings of the 28th ACM International Conference on Multimedia*, 2020, pp. 574–582.

[22] M. Zeng, X. Tan, R. Wang, Z. Ju, T. Qin, and T.-Y. Liu, "Musicbert: Symbolic music understanding with large-scale pre-training," *arXiv preprint arXiv:2106.05630*, 2021.

[23] E. Henry, "Netflow and word2vec -¿ flow2vec," 12 2016. [Online]. Available: https://edhenry.github.io/2016/12/21/Netflow-flow2vec/

[24] M. Ring, A. Dallmann, D. Landes, and A. Hotho, "Ip2vec: Learning similarities between ip addresses," in *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*. IEEE, 2017, pp. 657–666.

[25] F. Du, Y. Zhang, X. Bao, and B. Liu, "FENet: Roles Classification of IP Addresses Using Connection Patterns," in *International Conference on Information and Computer Technologies (ICICT)*, 2019, pp. 158–164.

[26] E. L. Goodman, C. Zimmerman, and C. Hudson, "Packet2vec: Utilizing word2vec for feature extraction in packet data," *arXiv preprint arXiv:2004.14477*, 2020.

[27] M. Mimura and H. Tanaka, "Reading Network Packets as a Natural Language for Intrusion Detection," in *International Conference on Information Security and Cryptology (ICISC)*. Cham: Springer International Publishing, 2018, pp. 339–350.

[28] Q. Le and T. Mikolov, "Distributed Representations of Sentences and Documents," in *International Conference on Machine Learning*, 2014, pp. 1188–1196.

[29] Devo, "Tcp flags," 2019. [Online]. Available: https://docs.devo.com/confluence/ndt/latest/searching-data/building-a-query/operations-reference/packet-group/tcp-flags-tcpflags