Shedding Light in the Tunnel: Counting Flows in Encrypted Network Traffic

Fares Meghdouri, Félix Iglesias Vázquez and Tanja Zseby
Technische Universität Wien
firstname.lastname@tuwien.ac.at

***Abstract*—Network traffic analysis helps experts to understand the behavior of communication networks. By exploiting information from packet headers and payloads, ML has been also widely applied to extend the capabilities of traditional statistical approaches. However, modern traffic encryption hampers network traffic analysis, especially in the case of VPNs when multiple flows are encrypted and transported concurrently. An initial step toward extracting information from encrypted traffic is to discover the number of flows that coexist in an aggregated and encrypted data stream. In this paper we propose a technique for disclosing the number of flows aggregated and sent together via encrypted tunnels. We use LSTM cells to learn the relation between the number of flows and the different temporal combinations from available attributes regardless of encryption. Results indicate that predicting the number of flows in encrypted tunnels with a relatively small error is surprisingly possible, providing the basis for a wide range of future research.**

***Index Terms*—network traffic, network flows, lstm**

I. Introduction

In recent years, Network Traffic Analysis (NTA) has helped to maintain robust and secure infrastructures. Because of the heterogeneity and increasing amount of today's Internet traffic, more complex yet lightweight analytical tools are required. Furthermore, traffic encryption has made the task more difficult, demanding methods that deliver satisfactory performances with less data or even with only meta-data, which implies additional challenges for Intrusion Detection System (IDS) developers. While traffic encryption protects users' privacy, it also makes intrusion detection, traffic monitoring, and quality control more difficult. On the other hand, monitoring network resources, usage and traffic in general is a critical step for any security infrastructure. One of the preliminary tasks to achieve the aforementioned goals consists on gathering statistics about traffic passing through a specific observation point and disclose the number of active flows. Having access to this information allow estimating the number of connections, hosts, or the network size in general. However, the latter is difficult due to the fact that encrypted traffic is often routed through encrypted tunnels in which the original Internet Protocol (IP) packet is encapsulated with a pre-header at aggregation points. For instance, when using Internet Protocol Security (IPsec) in tunnel gateway-to-gateway mode, all traffic leaving a local network has the IP address of the network gateway, making it difficult to separate or count the number of active flows once in the IPsec tunnel.

Previous research indicates that the flow count is required for a variety of applications. Some examples are:

- Monitoring resource usage for network management.
- Detection of specific types of attacks, e.g., Distributed Denial Of Service (D-DOS) and anomalies in communication patterns.
- Checking the state of sensors or IoT devices, for example in smart grids or smart production environments.
- De-anonymizing traffic to determine the number of communication parties.

In this paper, we present an architecture that uses Long Short-Term Memory (LSTM) cells [1] based on their attention principle to solve the problem of counting flows. The architecture uses packet features that are accessible even when traffic is encrypted, namely: *packet size* (PktSize), *inter-arrival time* (IAT) between packets, and *packet direction* (Direction). Subsequently, our approach is able to predict the correct number of concurrent flows in Virtual Private Networks (VPNs) tunnels.

We summarize the contributions of our paper as follows:

- We propose a lightweight architecture capable of accurately guessing the number of flows in encrypted tunnels.
- We analyze data from two consecutive years and show that many traffic characteristics did not significantly change over this time frame. Subsequent experiments show that, the proposed technique is therefore time-transferable (only requiring eventual fine tuning).
- We show that our proposal is robust against possible adversarial attacks that aim at influencing model predictions by manipulating certain traffic features.

The rest of the paper is organized as follows. In Section II, we discuss similar works and current challenges in encrypted traffic analysis. In Section III, we elaborate on the task of counting network flows and discuss the proposed solution. In Section IV we explain the methodology and describe the experimental test-bed. Later on, we show results and discuss them in Section V. Finally, the paper concludes in Section VI.

II. Related Work and Motivation

"Flow counting" was introduced as a problem to count non-encrypted flows in high-speed links, deal with the growing size of hashing tables and perform fast look-ups while capturing further traffic.

Kim and O'Hallaron [2] propose a timestamp-based solution, which shows high performance and low error. Similarly, Sanjuas-Cuxart et al. [3] propose an alternative solution based on Bitmaps [4] and use a sliding window to count flows in non-encrypted traffic over high-speed links. Alternatively, Hohn et al. [5] use sampling and estimation to count the number of flows as a viable approach if per-packet inspection is not feasible or requires specialized hardware.

However, tunneling makes the previous approaches not suitable. This is due to the fact that flows are aggregated together when leaving a local network or even a host, making counting the number of flows by using flow-keys for example no longer applicable since the same key can encapsulate a variable number of flows.

Nevertheless, *knowing the number of flows* is a prerequisite assumed in many applications related to network traffic analysis. For instance, Li et al. [6] propose a solution to identify Youtube flows in encrypted traffic. The number of flows is hence needed to differentiate between video and audio flows. This is particularly determining when QUIC is used and both traffic is sent over the same end-to-end connection.

Bar-Yanai et al. [7] present an encrypted traffic classification scheme that classifies network applications using clustering. Considering that authors define the number of centroids as a multiple of the number of protocols and/or applications, the number of flows is therefore an optimal estimate of the latter. Otherwise, if the number of flows is unknown, the number of clusters must be roughly guessed by means of iterative internal validation techniques, which are hardly feasible in actual implementations.

Shahbar et al. [8] present a method for identifying "Anonymity" flows in encrypted traffic at the packet-, flow-, and user-levels. In their experiments, authors collect flows that are aggregated by host, meaning that the number of flows is needed to identify "Anonymity" flows for users that handle multiple network services simultaneously on a single host.

He et al. [9] propose a method for identifying mobile applications in encrypted traffic (Domain Name System (DNS), Hypertext Transfer Protocol, (HTTP), Secure Sockets Layer (SSL) and Transport Layer Security (TLS)). The authors show that different applications generate traffic with different patterns, but also remark that encryption causes problems when two applications are simultaneously used. If the number of flows is known, traffic can be cleaned by removing interfering applications, therefore overcoming the mentioned limitation.

Overall, many proposed network traffic analysis methods require the number of concurrent flows either as a parameter in the analysis framework or indirectly to clean captured traffic beforehand and remove interfering flows [10]–[13].

III. Counting Flows

As discussed in Section I, traffic encryption makes traditional data extraction methods hardly feasible. In this paper, we face the case of IPsec tunnels that is increasingly common in Internet traffic. Here, all packets from a network are aggregated at the gateway, typically a router, which encapsulates, signs[1] and encrypts all packets before sending them over the Internet to the receiving gateway, thereby constructing a tunnel. Since packets inside the tunnel have the same origin and destination header, determining how many flows are transported cannot be done using traditional flow-key extraction methods. Hence, we are forced to use the external network information that is still available regardless encryption, in our case: packet size, packet direction, and times between consecutive packets. The following (reasonable) requirements are assumed in our proposal:

1) Gateways introduce minimal queuing delay. Otherwise, inter-arrival times would be affected.
2) Gateways encapsulate packets with fixed pre- and post-headers (either ESP or AH), increasing the size of all packets in the same way.
3) Gateways do not introduce application-based payload padding.

Note that when dealing with the inter-arrival time between packets, we consider two attributes: one that represents the time between packets regardless of their direction and a second one that only represents the time between packets of the same direction i.e., the time to the last packet with the same direction. The reasoning above is derived from the observation that packets of the same direction reflect typically the behavior of a single source, whereas when considering both directions more uncertainty is introduced because two consecutive packets of different directions represent the interaction of multiple sources and may have an arbitrary inter-arrival time (e.g. server response time).

A. Proposed Solution

The complexity of counting flows is highly dependent on the number of multiplexed flows, the nature of each flow, and the mechanism used to aggregate the flow (i.e. software running on gateways). Furthermore, because packets are generated based on user interaction, the resulting traffic usually does not exhibit stationary time patterns. The most important symbols used throughout the paper are summarized in table I.

Symbol	Description
N	number of packets in a sequence
M	number of sequences in a dataset
Δt	time between two consecutive packets
s	one sequence of packets
Γ	set observed Δt's
$y_{i,j}$	actual count for the i^{th} sequence and j^{th} packet
$\widehat{y}_{i,j}$	LSTM output for the i^{th} sequence and j^{th} packet
$\Delta\widehat{y}_{i,j}$	change in the LSTM output

TABLE I: Notation.

The example depicted in the upper part of Figure 1 shows two hosts sending packets on a regular basis and then aggregated in a tunnel. This example could represent two regular sensor reportings in a M2M setting. Sensors send packets with fixed time periods of Δt_1 and Δt_2 respectively. Since they are mixed within the tunnel, observing the final transmission does not reveal information about the number of sensors or the individual periods. However, it is possible to guess flows

[1]The gateway adds a message authentication code which is, strictly speaking, not a signature.

whenever Δt_1 and Δt_2 are known a-priori. Let **s** be a sequence of N packets in a tunnel, we can iterate over **s** and store the observed delays in a finite set Γ. If we initialize Γ with delays Δt_1 and Δt_2, and increase the number of counted flows only when spotting delays that are not already in Γ, we are able to roughly guess the number of flows, in this case, it would be 2. This manual approach would work whenever we assume that flows exhibit certain characteristics and stationary delays.

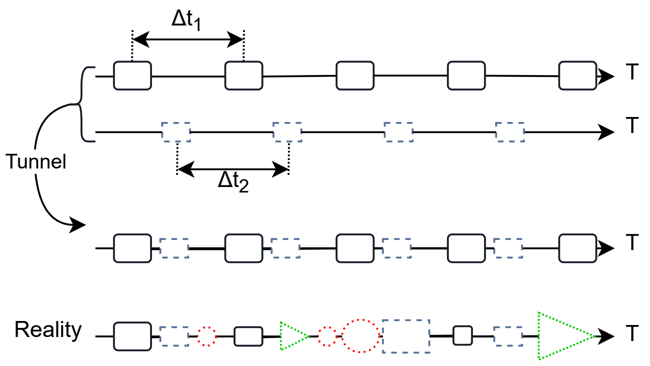


Fig. 1: An example of a simple tunnel with two periodic transmissions (upper part) and a realistic Internet tunnel (bottom part). Different shapes and colors represent packets of different sizes belonging to different applications.

Such rule-based approaches (**if-then-else**) could be enriched with information concerning packet size and packet direction. However, rule-based systems show some drawbacks [14], [15] too:

- Input data must be cautiously cleaned because noise can have a significant impact on a yes/no rule.
- There is no way to prove the absolute correctness of a rule inferred from previous observations, even if they are similar in the future.
- With the accumulation of nested rules, the order becomes important causing the generation of sub-optimal rule trees and branches, the degradation of the learning process itself, and an increasing opacity in the model (understood as a set of rules) interpretation.

The complexity of the above reasoning can quickly become unmanageable, requiring automated models built with regression or rule-induction methods. We opt for an advanced model-learning attention technique that (a) captures patterns instead of rules, and (b) stores and uses only what is required from past events therefore, we opt for LSTMs. After enough training, the model is expected to accurately guessing the number of flows in tunnels in a more complex setup.

B. Managing Complexity and Memory

We treat packet sequences as multidimensional time series and assume the existence of temporal dependencies between them. Recurrent Neural Networks (RNNs) architectures have been extensively used to solve similar problems. Such models use current data samples in addition to old information acquired from previous samples to construct valid predictions at each future time step. However, RNNs are well known to suffer from the vanishing gradients for long sequences causing the model to quickly forget past knowledge [16]. In order to overcome this limitation, we make use of a similar architecture that solves the previous problem: LSTMs.

LSTMs are a type of RNNs that have an enriched internal structure. In addition to the RNN structure, LSTM units contain many inner gates that allow to efficiently keep the past knowledge and use it in combination with future inputs. RNN units usually contain only an input gate that merges together the previous hidden state of the unit with the new input and an output gate. Instead, LSTM units introduces the new concept of the forget gate and the cell state. The forget gate decides whether to keep new knowledge or to “forget” it and the cell state is a combination of the previous cell state, the new input and the forget gate state. The idea behind is that, over time, the unit learns which information to remember and which to forget, this is intuitively similar to what happens in Natural Language Processing (NLP) applications. We configure our architecture so that each time step represents a new captured packet and the multidimensional input consists of the four features discussed above, namely: PktSize, Direction, mixed IAT, and same-direction IAT. Therefore, we construct sequences of length N and dimension 4. The cell output $y_{i,j}$ is simply the estimated number of flows up to the current time step (j) for a given sequence (i).

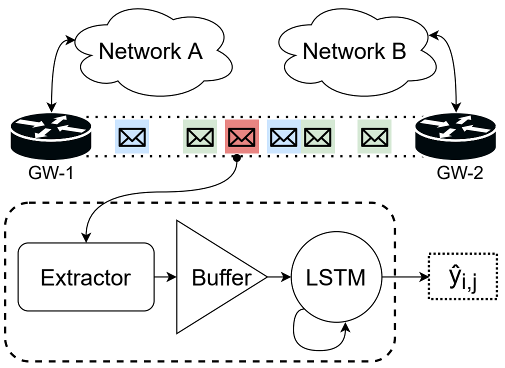


Fig. 2: LSTM-based solution to count flows in a tunnel. The network is fed only with available packet meta-data.

Note that in our proposal the LSTM uses sequences of fixed length. Such sequences represent sets of N packets taken in a sliding window rather than a single infinite sequence[2]. A defined sequence length is required to prevent error propagation for long sequences if the network prediction begins to drift away from the actual number of flows. By having a short sequence length, the cell state is flushed at the beginning of a new sequence and the error is hence minimized. However, extremely short sequences provide minimal information and lead to poor predictions.

The architecture shown in Figure 2 depicts two E2E flows being routed through a tunnel with GW1 and GW2 acting

[2]In the case of an infinite sequence, the cell state is never flushed and predictions are obtained after each time step continuously.

as VPN aggregation gateways. The counting chain is divided into several stages: (1) parsing network packets, (2) extracting and processing of required features, (3) using a buffer to store fixed-length sequences of feature vectors, and (4) step-wise feeding of sequences into the LSTM model. The LSTM updates its state and predicts the number of flows ($\widehat{y}_{i,j}$) at each time step of the current observation window. The proposed approach is able to count the number of passing flows incrementally and keeping track of active flows is hardly feasible since the termination of a flow is usually defined by specific transport layer flags that are not accessible in encryption. Guessing active flows would require flow-duration estimations, which is out of the scope covered in this work.

IV. Experiments

In this section we discuss the experimental methodology and relevant details for reproduciblity. The source code, data, and pre-trained models are available in our repository[3].

A. Data

For our experiments we use real traces captured by the MAWI[4] working group at backbone links. Data collected during June 2020 (Samplepoint-G, average rate: 2.3Gbps with ∼465 million packets) is used for training, validation and testing (Test 1). We then use a second capture from February 2021 (Samplepoint-F, average rate: 420Mbps with ∼83 million packets) for a second test (Test 2) in order to check if a trained model can be used with new traffic observed several months later in a different sampling point. We provide statistics related to traffic characteristics of both sampling points in Figure 3. Direction is *forward* when the packet is sent by the device that started the flow; otherwise, "packet direction"' is *backward*. TCP is obviously the most frequently used protocol in both traces. We show the distribution of various protocols in Table II.

Dataset	TCP	UDP	ICMP	Others
Samplepoint-G	76.7%	15.8%	01.9%	05.6%
Samplepoint-F	61.2%	29.6%	00.6%	08.6%

TABLE II: Protocol distribution in MAWI network traces.

B. Configuration of the Learning Scheme

Our learning prototype consists of a three-cell stacked LSTM network that feed on sequences of a fixed length N. In our experiments, we test different sequence lengths, $N = \{50, 100, 500, 1000\}$. We empirically set 128 neurons for each intermediate layer with *Leaky ReLU* [17] activations and a *sigmoid* activation at the output layer. Dropout is used at each layer with $r = 0.2$ to reduce over-fitting [18]. During training, the Mean Absolute Error (MAE) is used to compute the loss at each element of the sequence. The MAE is used because it is more robust against outliers in comparison to the Mean Squared Error (MSE) which usually aggravate larger losses. An *Adam* optimizer [19] is used to back-propagate the computed loss.

To deal with the long tailed distribution of packet sizes and inter-arrival times (see Figures 3a, 3e and 3f), we apply a log transform (Equation 1), which focus on orders of magnitude and squeezes the attribute ranges.

$$x' = log(1 + x) \tag{1}$$

Afterwards, a z-score normalization is applied to the data to obtain zero-mean, unit-variance distributions which has been shown to be optimal for gradient-based learning [20].

C. Generation the Ground Truth

The training of the LSTM requires prior knowledge of the number of flows, which is unavailable in encrypted tunnels. To solve this problem, we use unencrypted data to construct a ground truth. If the assumptions in Section III are met, using the unencrypted traces should be equivalent to the usage of encrypted traces for the undertaken task of guessing the number of flows with the proposed features.

D. Evaluation Metrics

The model is trained to detect the start of each new flow and increment its output respectively. We use the MAE for measuring the absolute distance of the prediction to the actual value. We additionally use the Mean Absolute Percentage Error (MAPE) for a more intuitive interpretation of results. The MAPE represents a marginal error per sample in %. Given the true prediction $y_{i,j}$ of the the j^{th} timestamp for the i^{th} sequence, and $\widehat{y}_{i,j}$ being the respective LSTM prediction, M the number of sequences in a test set and N the length of the input sequence, the MAPE is computed as follows:

$$MAPE = \frac{1}{M} \sum_{i=0}^{M-1} \left| \frac{\widehat{y}_{i,N} - y_{i,N}}{y_{i,N}} \right| \times 100 \quad [\%] \tag{2}$$

The MAPE allows us to directly interpret error as number of flows in excess or absence. Moreover, it weights results and makes them comparable when facing windows with different number of flows on average. Finally, we define an accuracy score to reflect the counting performance after observing a entire test network trace. This score compares the final number of predicted flows with the real number. Therefore,

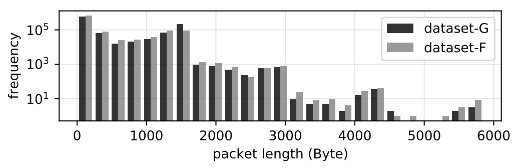
$$Accuracy = \left(1 - \left| \frac{\sum_{i=0}^{M-1} y_{i,N} - \sum_{i=0}^{M-1} \widehat{y}_{i,N}}{\sum_{i=0}^{M-1} y_{i,N}} \right| \right) \times 100 \quad [\%] \tag{3}$$

V. Performance and Discussion

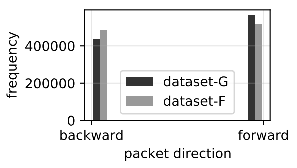
We train multiple vanilla LSTM networks and select the one with the minimum MAE loss for each sequence length (50, 100, 500 and 1000) after a given number of epochs. To obtain interpretable results we denormalize predictions and data for the evaluation.

[3]https://github.com/CN-TU/network-flow-counting

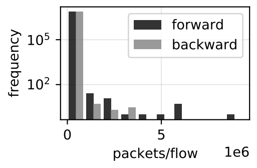
[4]https://mawi.wide.ad.jp/mawi



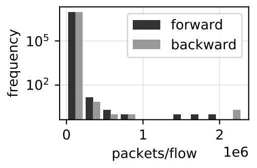
(a) Packet length.



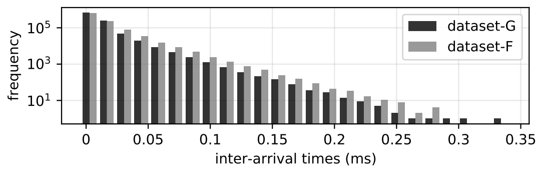
(b) Packet direction.



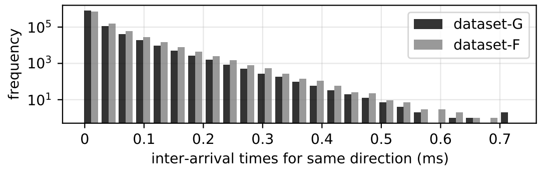
(c) Dataset-G.



(d) Dataset-F.

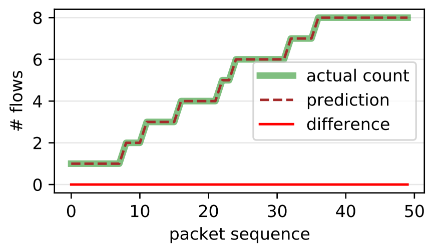


(e) Mixed IATs.

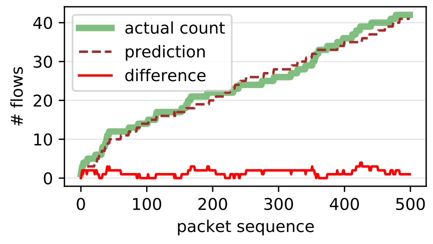


(f) Separate IATs.

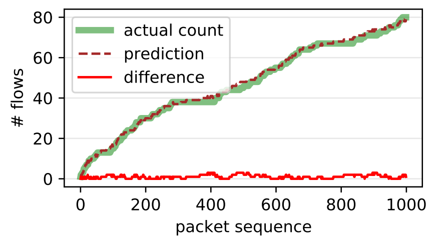
Fig. 3: Histograms showing the distributions of relevant features in the datasets used for evaluation



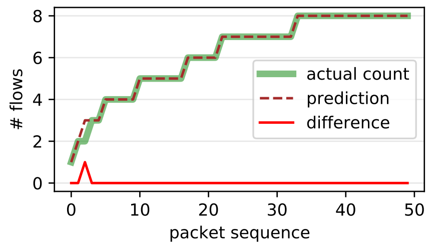
(a) Seq. length: 50 packets.



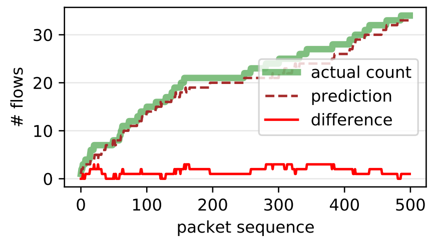
(b) Seq. length: 500 packets.



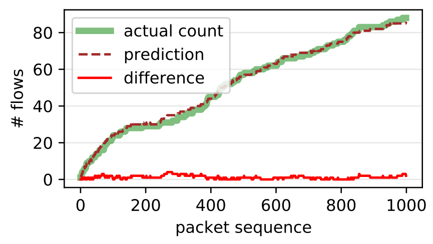
(c) Seq. length: 1000 packets.



(d) Seq. length: 50 packets.

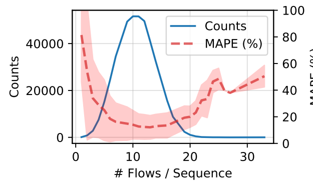


(e) Seq. length: 500 packets.

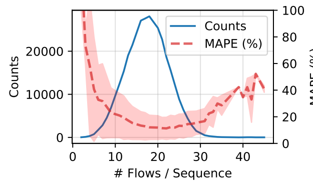


(f) Seq. length: 1000 packets.

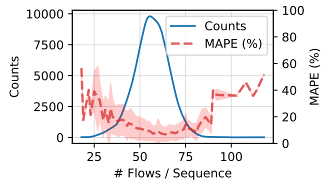
Fig. 4: Random samples chosen to visualize the estimated flow count. The prediction closely follows the ground truth showing minimal error. Note that the number of flows in a sequence is an accumulative measure (i.e., it can only increase).



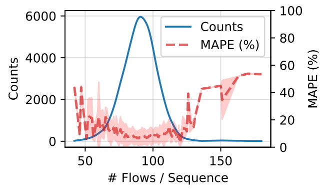
(a) 50 packets.



(b) 100 packets.



(c) 500 packets.



(d) 1000 packets.

Fig. 5: The average MAPE for each "number of flows per sequence" value for various sequence lengths. The count represent the number of sequences from the test set with a given "number of flows per sequence".

Dataset	Sequence	AFS*	MAPE** ↓	MAE ↓	Acc. ↑
Validation	50	10.58	±15.95%	2.42	97.05%
	100	17.54	±14.73%	5.57	97.62%
	500	56.46	±10.23%	7.29	98.05%
	1000	90.47	**±09.06%**	8.18	98.89%
Test 1	50	22.54	±22.61%	5.43	97.05%
	100	41.04	±21.90%	9.41	97.05%
	500	164.05	±18.60%	13.95	97.05%
	1000	291.318	**±17.23%**	18.57	97.05%
Test 2	50	26.14	±23.64%	6.18	76.57%
	100	48.01	±21.00%	10.55	77.16%
	500	195.30	±19.52%	13.46	80.05%
	1000	346.093	**±19.23%**	21.56	82.05%

$*$ AFS: Average # Flows per Sequence.
$**$The MAPE represents the margin regardless of the sequence length or the number of flows in a sequence, e.g., if we consider a dataset with 100k flows and use 500-packet sequences (MAPE=9.06%), we get a margin of ± 9.06k flows i.e., an estimate between 90.94k and 109.06k flows.

TABLE III: Performance evaluation over three datasets. The arrows indicate whether a high or low score is preferable.

A. *Overall Performance*

Figure 4 shows six random sequences of length 50, 500 and 1000 packets. Curves depict the increase in the number of flows in a window as well as the model prediction after each packet. The figure illustrates how the predictions accurately match the ground truth with small or no margin of error (represented by the brown line).

In Figure 5, we show the average MAPE value over all samples in the test set for a given "number of flows per sequence". We analyse this statistic for various sequence lengths and we notice a smaller MAPE at high flow counts; this indicates that the model is trained better for such values because of more training samples (aka sequences), and thus the minimum MAPE is obtained for values where the maximum of the count curve occurs.

We show various error measures discussed in Section IV-D in Table III. The best MAPE (9.06%) is obtained on the validation set for sequences of 1000 packets, which means that the estimation of 90.47 flows (in a 1000 packet sequence) in this case is on average off by approximately 9.19 flows (MAPE $\times$ AFS.). Furthermore, the longer the sequence length is, the smaller the MAPE and the better the Accuracy are. Such behavior can be justified as a consequence of the *reversion to mediocrity* effect; in other words, the more data we have (past knowledge), the more representative learned patterns and statistical estimations become. Besides that, missing a few flows in a long sequence has a smaller effect than missing the same number of flows in a shorter sequence. However, the error naturally increases and the model is less optimal when guessing flows in data from a different time frame (Test 1) or a different year (Test 2).

The results above indicate that longer sequences are better suited to count flows.

But note that using extremely long sequences can cause error accumulation due to the fact that previous estimations and cell states are used as "inputs" in the LSTM. Moreover, the LSTM model requires a fixed length to delimit a meaningful context scope and can also suffer from exploding/vanishing gradient problems [16]. The study of optimal observation windows is one of the challenges to be tackled in subsequent work.

B. *Sensitivity Analysis*

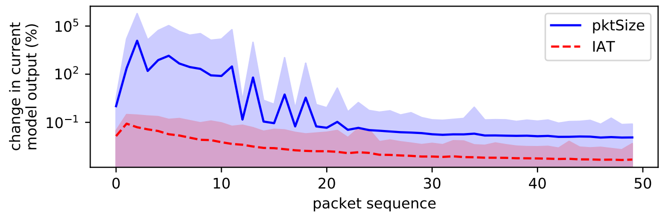
In the design of any Machine Learning (ML)-based predictor for security applications, robustness against adversarial attacks becomes highly relevant. Attackers may be interested in manipulating traffic to conceal their malicious activity or falsify statistics. In this section we test the stability of our proposal against variations of the key features. For this purpose, we carry out sensitivity analysis similar to [21].

Partial Dependence Plots (PDPs) [22] build global model explanations by fixing one feature and altering the rest over all possible combinations. In our case, we slightly change this principle and build a perturbation model that shows the change in the model output when a single feature is manipulated over a defined range. We alter the packet size and the inter-arrival-time up to a maximum of 400% increase (note that an attacker could only increase those values). Because our model deals with sequences rather than a static input and predictions occur after each time step, we define two types of perturbation effects. Let $\mathbf{x}_{i,j}$ be the four-dimensional input of the the i^{th} sequence at the j^{th} time step and $y_{i,j}$ its respective ground truth output, N being the sequence length, and k and p the current sequence and current time step indices respectively, therefore:

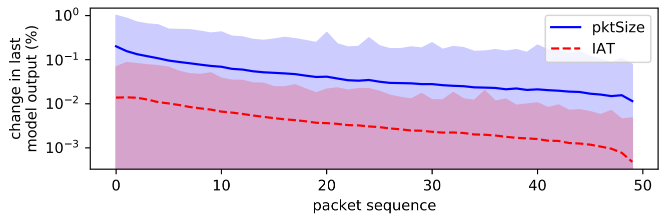
- The Local Perturbation is the amount of change in the LSTM current output ($\Delta\widehat{y}_{k,p}$) caused by variations in the current input ($x_{k,p}$).
- The Global Perturbation is the amount of change in the LSTM last output ($\Delta\widehat{y}_{k,N}$) caused by variations in the current input ($x_{k,p}$).

We show both perturbations ($\Delta y_{i,j}$ and $\Delta y_{i,N}$) for sequences of length $N = 50$ in Figure 6a and Figure 6b. Curves represent average values across the entire dataset (200k sequences), while shaded areas represent standard deviations. Both figures illustrate that the magnitude of the perturbation usually decreases toward the end of the sequence, which can be explained by the nature of the model used. LSTMs tend to retain knowledge from previous time steps, therefore new samples affect the prediction less if the previous knowledge is genuine. In short, the later the disturbances occur, the less effect they have. A second observation is that variations in packet lengths have a larger impact than variations in inter-arrival times. We hypothesize that this is caused by the stronger discriminant power of packet length to characterize network traffic [23], [24]. Finally, we observe that perturbations have a higher impact on the local outputs (Figure 6a) than on the final output (Figure 6b), which suggests that the final prediction at the end of the sequence is more robust against manipulations.

In any case, packet length is difficult to manipulate in encrypted network traffic if the attacker is outside the tunnel



(a) Perturbation effect on current prediction.



(b) Perturbation effect on last prediction.

Fig. 6: Perturbation analysis of the LSTM output for sequences of length 50. At each sample, the packet size and inter-arrival times are manipulated up to 400%.

because IP header checksums of the original packet must be recalculated, which requires decryption first.

VI. Conclusion

Network traffic analysis at network layer level has become a challenging task due to encryption. In extreme cases, such as IPsec, nearly all traditional features used for analysis are encrypted, demanding the development of a new generation of analysis methods. In this work we presented a preliminary manual solution for counting encrypted flows in network tunnels. We then propose a ML-based approach and show that LSTMs can overcome the complexity limitations of manual approaches as they learn multiple patterns while only retaining relevant information from past data.

Our experiments over two backbone datasets with various sequence lengths reveal that the proposed solution performs satisfactorily with $\approx$ 9% error margin. The same model applied on data from a year later still obtains $\approx$ 19% error margin, indicating that trained models are temporally transferable for certain applications with low error, thus requiring only long-term adjustment and fine tuning. Beyond flow counting, future work undertakes using the same LSTMs model, architecture and flow count estimation for solving more complex tasks such as clustering packets based on the flow they belong to.

Acknowledgment

This work was supported by the project MALware cOmmunication in cRitical Infrastructures (MALORI), funded by the Austrian security research program KIRAS of the Federal Ministry for Agriculture, Regions and Tourism (BMLRT) under grant no. 873511.

References

[1] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, 1997.

[2] H.-A. Kim and D. R. O'Hallaron, "Counting network flows in real time," in *GLOBECOM'03. IEEE Global Telecommunications Conference (IEEE Cat. No. 03CH37489)*, IEEE, 2003.

[3] J. Sanjuas-Cuxart, P. Barlet-Ros, and J. Solé-Pareta, "Counting flows over sliding windows in high speed networks," in *International Conference on Research in Networking*, Springer, 2009.

[4] C. Estan, G. Varghese, and M. Fisk, "Bitmap algorithms for counting active flows on high speed links," in *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, 2003.

[5] N. Hohn and D. Veitch, "Inverting sampled traffic," *IEEE/ACM Transactions on Networking*, 2006.

[6] F. Li, J. W. Chung, and M. Claypool, "Silhouette: Identifying youtube video flows from encrypted traffic," in *Proceedings of the 28th ACM SIGMM Workshop on Network and Operating Systems Support for Digital Audio and Video*, 2018.

[7] R. Bar-Yanai, M. Langberg, D. Peleg, and L. Roditty, "Realtime classification for encrypted traffic," in *International Symposium on Experimental Algorithms*, Springer, 2010.

[8] K. Shahbar and A. N. Zincir-Heywood, "How far can we push flow analysis to identify encrypted anonymity network traffic?," in *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*, IEEE, 2018.

[9] G. He, B. Xu, L. Zhang, and H. Zhu, "Mobile app identification for encrypted network flows by traffic correlation," *International Journal of Distributed Sensor Networks*, 2018.

[10] A. Chen, Y. Jin, J. Cao, and L. E. Li, "Tracking long duration flows in network traffic," in *2010 Proceedings IEEE INFOCOM*, IEEE, 2010.

[11] R. Schuster, V. Shmatikov, and E. Tromer, "Beauty and the burst: Remote identification of encrypted video streams," in *26th Security Symposium*, 2017.

[12] R. Dubin, A. Dvir, O. Pele, and O. Hadar, "I know what you saw last minuteencrypted http adaptive video streaming title classification," *IEEE transactions on information forensics and security*, 2017.

[13] J. Datta, N. Kataria, and N. Hubballi, "Network traffic classification in encrypted environment: a case study of google hangout," in *2015 twenty first national conference on communications (NCC)*, IEEE, 2015.

[14] J. W. Grzymala-Busse, "Rule induction," in *Data mining and knowledge discovery handbook*, Springer, 2005.

[15] F. Bergadano, "The problem of induction and machine learning.," in *IJCAI*, 1991.

[16] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *International conference on machine learning*, PMLR, 2013.

[17] A. L. Maas, A. Y. Hannun, A. Y. Ng, *et al.*, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. icml*, Citeseer, 2013.

[18] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, 2014.

[19] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[20] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," in *Neural networks: Tricks of the trade*, Springer, 2012.

[21] A. Hartl, M. Bachl, J. Fabini, and T. Zseby, "Explainability and adversarial robustness for rnns," in *2020 IEEE Sixth International Conference on Big Data Computing Service and Applications (BigDataService)*, IEEE, 2020.

[22] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, 2001.

[23] A. Este, F. Gringoli, and L. Salgarelli, "On the stability of the information carried by traffic flow features at the packet level," *ACM SIGCOMM Computer Communication Review*, 2009.

[24] Y.-s. Lim, H.-c. Kim, J. Jeong, C.-k. Kim, T. T. Kwon, and Y. Choi, "Internet traffic classification demystified: on the sources of the discriminative power," in *Proceedings of the 6th International Conference*, 2010.