2021 24th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS) | 978-1-6654-3595-6/20/$31.00 ©2021 IEEE | DOI: 10.1109/DDECS52668.2021.9417068

Efficient Acceleration of Decision Tree Algorithms for Encrypted Network Traffic Analysis

Roman Vrána
Faculty of Information Technology
Brno University of Technology
Božetěchova 1/2 612 66 Brno, Czech Republic
ivrana@fit.vutbr.cz

Jan Kořenek
Faculty of Information Technology
Brno University of Technology
Božetěchova 1/2 612 66 Brno, Czech Republic
korenek@fit.vutbr.cz

***Abstract*—Network traffic analysis and deep packet inspection are time-consuming tasks, which current processors can not handle at 100 Gbps speed. Therefore security systems need fast packet processing with hardware acceleration. With the growing of encrypted network traffic, it is necessary to extend Intrusion Detection Systems (IDSes) and other security tools by new detection methods. Security tools started to use classifiers trained by machine learning techniques based on decision trees. Random Forest, Compact Random Forest and AdaBoost provide excellent result in network traffic analysis. Unfortunately, hardware architectures for these machine learning techniques need high utilisation of on-chip memory and logic resources. Therefore we propose several optimisations of highly pipelined architecture for acceleration of machine learning techniques based on decision trees. The optimisations use the various encoding of a feature vector to reduce hardware resources. Due to the proposed optimisations, it was possible to reduce LUTs by 70.5 % for HTTP brute force attack detection and BRAMs by 50 % for application protocol identification. Both with only negligible impact on classifiers' accuracy. Moreover, proposed optimisations reduce wires and multiplexors in the processing pipeline, positively affecting the proposed architecture's maximal achievable frequency.**

***Index Terms*—acceleration, network, threat, detection, decision tree,**

I. Introduction

The ever-increasing speed of network lines needs more and more effort to increase the speed of networking systems. As every packet can arrive every five ns on 100 Gb line, high-speed packet processing is necessary. The problem is critical, especially in network security systems, where every packet drop directly impacts network threat detection.

With encrypted network data we cannot rely only on the pattern matching in Intrusion Detection Systems (IDS). IDS systems have to be enhanced by utilising statistical information such as packet lengths or inter-packet gaps. Anderson [1] has shown how machine learning can be used to analyse encrypted network traffic. He was able to achieve very high precision in the detection and identification of network threats. The machine learning techniques have been successfully used for other use-cases such as classifying application protocols [7], identifying DNS over HTTPS [6] or detect SSH brute-force attacks [2].

978-1-6654-3595-6/21/$31.00 ©2021 IEEE

Although deep neural networks are top-rated in various fields, encrypted traffic analysis is usually based on decision tree-based classifiers such as Random Forest (RF) or AdaBoost. These classifiers are highly accurate, have deterministic behaviour, and can be easily analysed because they use a set of parallel decision trees. This also makes designing a hardware accelerator easier since the tree structure is relatively simple to map into hardware.

Several architectures have been proposed for mapping decision tree-based classifiers to hardware [3]–[5]. These architectures unfold nodes of decision trees to pipeline stages to create deep pipeline and achieve high throughput. As the feature vector is transferred on large data bus and features are selected in every pipeline stage by large multiplexers, the architectures need many hardware resources.

We focused on designing a hardware architecture that would allow for an online analysis of encrypted network traffic using a set of decision tree classifiers. The proposed architecture uses a single processing pipeline for every decision tree. It reduces the size of the feature vector to decrease hardware resources and reduce routing problems. The proposed architecture is evaluated on three different classifiers, which are used [2], [6], [7] for the analysis of encrypted network traffic. It was possible for all classifiers to significantly reduce hardware resources with only negligible impact on precision.

II. Related Work

The first mapping of RF classifier to FPGA [5] uses a deep processing pipeline, where each stage represents one level of a decision tree and has a memory block store all nodes of the same tree level. The pipeline depth and the number of parallel memories correspond to the decision tree depth. As the nodes are stored in memories, the architecture significantly reduces resource requirements compared to directly mapping the tree structure to the FPGA logic. Direct mapping also has frequency issues due to the routing problems caused by the number of parallel trees connected to the one feature vector.

Despite pipeline structure and nodes stored in memory blocks, the architecture in [5] still utilises many hardware resources and has problems with routing. A large feature vector causes routing issues and increases hardware resources utilisation because of large multiplexers and many flip-flop registers in every pipeline stage.

Prasanna et al. [4] proposed a discretisation technique by mapping the input feature values into intervals denoted by thresholds taken from the original decision tree. This approach results in an encoder for each feature, which reduces memory requirements and comparators' size in the decision tree pipeline. However, if the classifier has a tree with small depth, the utilisation of hardware resources is increased. To alleviate this potential issue, Prasanna [4] introduces approaches integrating a preprocessing step into the classifier tree to reduce resource utilisation at the cost of requiring two comparisons to fit the input value into the interval. However, two comparisons in every pipeline stage directly impact the throughput because two comparisons can stop the pipeline and reduce the processing speed. To avoid pipeline blocking, we choose encode features before entering the decision tree pipeline. Moreover, this optimisation still uses a wide data bus to pass features over the pipeline stages and thus need many flip-flop registers. The big feature vector bus directly impacts multiplexers, routing resources, and system frequency similarly as in the original architecture [5].

III. Design Optimizations

In our work, we focus on optimising the architecture described in [5] while using the *memory-centric* solution described [3]. Figure 1 shows an overview of the classifier with the input pipeline. The upper part of the figure shows a schematic of a decision tree which we then map to the pipeline architecture below. Each level of the tree is mapped to a single step of the pipeline. Memory component MEM holds all thresholds in the decision tree level, index of the feature used in the comparison, and the classification result. In every pipeline stage, first data are read from memory and then passed to the CMP block. The CMP block then selects the feature and performs the comparison which decides how the next step in the pipeline will perform.

The main issue of the described architecture is the large width of the feature vector which in turn may result in complicated routing. This issue stems from using either a large number of input features or using values that are too precise and require fine resolution which can also increase the complexity of comparators. The feature count can be reduced relatively easily by omitting those features that are not significant enough in the decision process. To reduce the comparator complexity, we can use fixed-point arithmetic with proper setting instead of floating point arithmetic. This significantly reduces resources needed for the comparator logic with negligible impact on accuracy. To reduce the size of the feature vector however, we need to implement an optimization, that preserves the accuracy as much as possible. We explore two possible approaches to achieve this goal.

Second significant factor influencing the amount of resources used is depth of the tree since each level exponentially increases the amount of nodes and thus number of items stored. This would lead to a large consumption of BRAMs

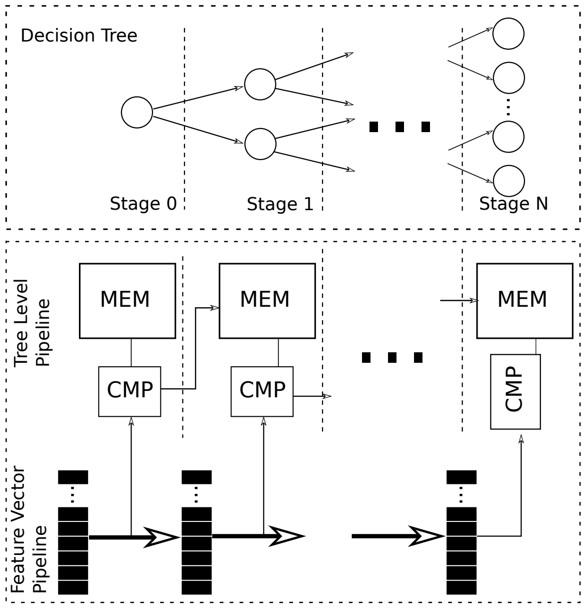

Fig. 1. Mapping of a decision tree into the pipeline

while many of them would not be utilised properly. To reduce this issue, the proposed design starts using BRAMs only when they can be utilised at 50% of their capacity at least. Until then, the required memory blocks are implemented in logic.

The proposed design uses 32-bit fixed-point arithmetic feature thresholds to minimize the loss of accuracy as much as possible when replacing the floating-point arithmetic. This significantly reduces the resources required to implement the decision block. The width of the fixed-point value however becomes an issue when we add the information about the class and the feature index. The resulting item may be wider than one row of the memory. This would require either to store the data in multiple consecutive rows or using memory configuration with support for wider data. The first approach would require accessing the memory multiple times to load the item thus increasing latency. The other approach would increase on-chip memory usage while not being able to efficiently utilize the item width.

When classifying a sample with a decision tree in general, we are not looking for the exact value of a selected feature. Instead, we are trying to fit the feature into a certain range. This allows us to devise a mapping that would provide a comparable result while requiring less resources and/or improve the on-chip memory utilisation. In our work we will explore two approaches for reducing the width of the feature vector as well as size of the data stored in every tree level.

The first proposed approach is trying to reduce the resources by lowering the resolution of used features. To decrease resolution, we only pick from each feature the first-N most significant bits. This simple step results in a narrower pipeline since only the step before entering the classifier needs to be implemented using the full width. All following steps will use the reduced width. The reduction must take the precision of the original input into account. Otherwise, the loss of resolution will lead to a significant loss of accuracy.

In the second approach, we use thresholds' values from the original classifier to map the input vector to integer arithmetic. The thresholds serve as upper bounds of intervals. For every input feature value, we have to find a corresponding interval. The approach requires the implementation of a tree structure similar to the classifier to encode the input vector. The encoder's output is a new input vector where each item is an index of the interval. Intervals allow us to reduce the vector's size more precisely than the previous approach at the cost of a more complex encoder.

IV. Results

We evaluated the throughput and utilisation of hardware resources for various techniques of mapping decision tree-based classifiers to the high-speed hardware architecture. The proposed hardware architecture uses unrolling the decision tree into the deep pipeline [5] together with two optimisations to encode floating-point input values into integers. These optimisations decrease mapping complexity and reduce hardware resources. We look at the estimated resources required to implement a classifier with proposed optimisations and compare it with the original pipelined architecture. We use a Xilinx Kintex-7 device as a target technology to provide numbers of utilised LUTs and BRAMs. For the evaluation, we selected 3 different IDS use-cases: (i) DNS-over-HTTPS detection, (ii) HTTPS login brute-force detection and (iii) traffic classification. In all cases, we're aiming to achieve an accuracy comparable to related work aimed at described cases. The reference classifier in terms of resources spent is an implementation based on design described in [5].

A. Discretisation by sorting to buckets

The first approach to further reduce on-chip resources is to take the input values and reduce the bitwidth of the input vector by taking only first N bits of the each feature. This relatively simple approach helps reduce the width of the pipeline as well as memory requirements. The effect however introduces a loss in resolution of the feature value which can lead to significant reduction in accuracy. Graphs in figure 2 demonstrate the impact of the approach using different bitwidth. From the results it can be clearly observed that we can achieve accuracy that's comparable to the original solution while using noticeably less resources. The only requirement is that the reduction must not reduce the resolution too significantly. The impact of this can be observed in 2-A when we use only 8 bits for the features and in 2-B when using 8 or 16 bits per feature. However with proper setting the loss in accuracy is negligible while noticeably reducing the resources used.

B. Discretisation by sorting into intervals

Second optimisation approach uses data from the original classifier to create a discretisation tree for each feature. This allows for a more accurate conversion while significantly reducing resources needed to implement the classifier. Graphs in figure 3 demonstrate the impact of the conversion in all presented use cases. As we can see, the discretisation tree allows to reduce the input vector even further at the cost of using a little bit more resources to encode the initial values. These resources may actually exceed the unoptimised design, either when the classifier is small (small depth of decision tree) as is the case for 3-A or when the encoder requires a deep pipeline as in 3-C. In 3-B we can observe a reduction in resources while preserving the accuracy.

C. Final Comparison

Both tested approaches show a noticeable reduction in the resulting classifier's resources while having a negligible impact on the overall performance. All approaches show a significant reduction in LUTs used to implement the architecture. As shown in the L7 classification use-case, the optimisation allows us to save LUTs, FFs and BRAMs. Graphs in figure 4 compare both optimisation approaches to the original classifier without optimizations applied for every described use case. As can be seen from the results, both optimisations reduce resources utilisation compared to the original HW design. The reduction allows us to add or remove additional trees or implement deeper pipeline if needed more easily. In terms of applied optimisations, sorting values into buckets is more efficient for L7 classification since it requires a higher resolution. The tree-based discretisation requires more resources to implement while providing only marginally better results. However, for other use-cases, the tree-based discretisation saves more resources and allows us to implement a narrower pipeline.

V. Conclusions

We introduced optimised hardware architecture for encrypted network traffic analysis in IDS systems. The architecture is designed for Random Forest, AdaBoost and other decision trees based classifiers. High-speed frequency and throughput are achieved by deep processing pipeline. The architecture utilises two types of optimisations focused on the feature vector's data width, which directly impacts the size of multiplexors and flip-flops in every processing pipeline stage and interconnection wires in the processing pipeline. We could reduce by optimisations LUTs by 70.5 % for HTTP brute force attack detection and by 83 % for DNS over HTTPS detection. Moreover, the optimisations reduced LUTs by 24 % and BRAMs by 50 % for the application protocol identification.

As future work, we want to analyse more deeply application identification use-case. To further reduce hardware resources, we want to remove one classifier by a set of smaller binary classifiers.

Acknowledgment

This work was supported by Brno University of Technology project FIT-S-20-6309 and Security Research project VI20192022143 granted by Ministry of Interior of the Czech Republic.

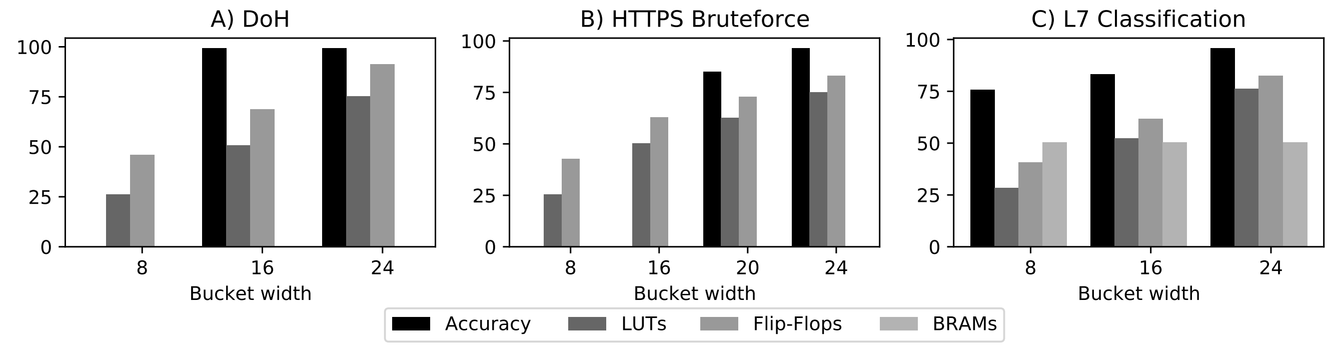

Fig. 2. Accuracy and resources used for bucket optimisation

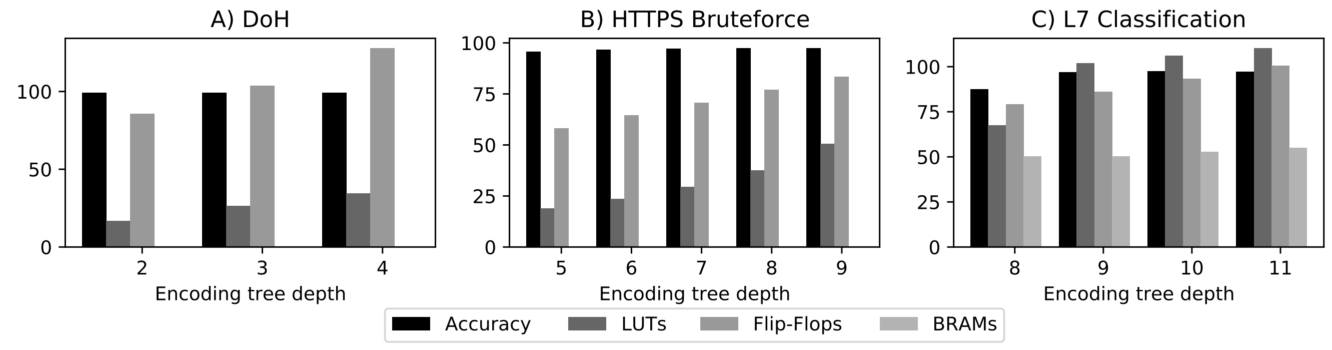

Fig. 3. Accuracy and resources used for tree encoding optimisation

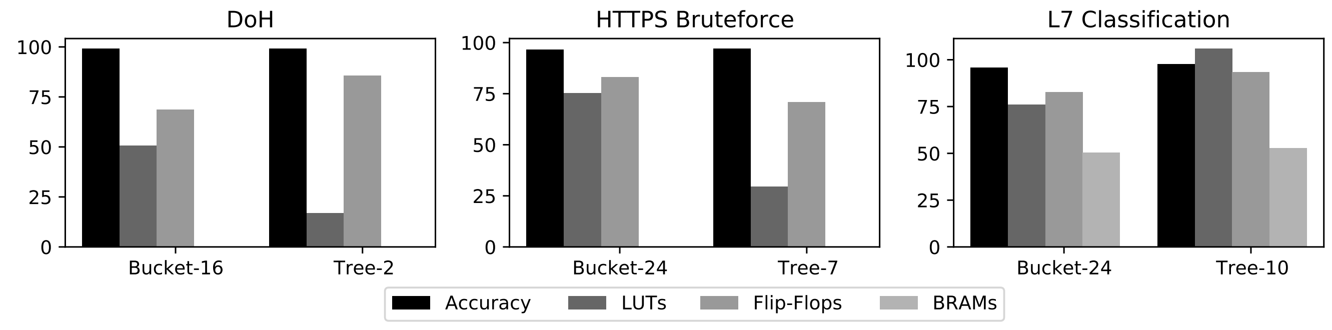

Fig. 4. Comparison of optimisations to the reference classifier

REFERENCES

[1] Blake Anderson and David McGrew. Identifying encrypted malware traffic with contextual flow data. In *Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security*, AISec '16, page 35–46, New York, NY, USA, 2016. Association for Computing Machinery.

[2] Karel Hynek, Tomáš Beneš, Tomáš Čejka, and Hana Kubátová. Refined detection of ssh brute-force attackers using machine learning. In Marko Hölbl, Kai Rannenberg, and Tatjana Welzer, editors, *ICT Systems Security and Privacy Protection*, pages 49–63, Cham, 2020. Springer International Publishing.

[3] Xiang Lin, R.D. Shawn Blanton, and Donald E. Thomas. Random forest architectures on fpga for multiple applications. In *Proceedings of the on Great Lakes Symposium on VLSI 2017*, GLSVLSI '17, page 415–418, New York, NY, USA, 2017. Association for Computing Machinery.

[4] D. Tong, Y. R. Qu, and V. K. Prasanna. Accelerating decision tree based traffic classification on fpga and multicore platforms. *IEEE Transactions on Parallel and Distributed Systems*, 28(11):3046–3059, 2017.

[5] B. Van Essen, C. Macaraeg, M. Gokhale, and R. Prenger. Accelerating a random forest classifier: Multi-core, gp-gpu, or fpga? In *2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines*, pages 232–239, 2012.

[6] Dmitrii Vekshin, Karel Hynek, and Tomas Cejka. Doh insight: Detecting dns over https by machine learning. In *Proceedings of the 15th International Conference on Availability, Reliability and Security*, ARES '20, New York, NY, USA, 2020. Association for Computing Machinery.

[7] Petr Velan, Milan Čermák, Pavel Čeleda, and Martin Drašar. A survey of methods for encrypted traffic classification and analysis. *Netw.*, 25(5):355–374, September 2015.