Dai, Z., Shrivastava, A., Reviriego, P. & Hernandez, J. A. (2022, septiembre). Optimizing Learned Bloom Filters: How Much Should Be Learned? IEEE Embedded Systems Letters, 14(3), 123-126.

DOI: 10.1109/les.2022.3156019

Optimizing Learned Bloom Filters: How Much Should Be Learned?

Zhenwei Dai[1], Anshumali Shrivastava[1], Pedro Reviriego[2], José Alberto Hernández[2]

***Abstract*—The Learned Bloom Filter (LBF) combines a machine learning model (learner) with a traditional Bloom filter to improve the false positive rate (FPR) that can be achieved for a given memory budget. The LBF has recently been generalized by making use of the full spectrum of the learner's prediction score. However, in all those designs, the machine learning model is fixed. In this paper, for the first time, the design of learned Bloom filters is proposed and evaluated by considering the machine learning model as one of the variables in the process. In detail, for a given memory budget, several LBFs are constructed using different machine learning models and the one with the lowest false positive rate is selected. We demonstrate that our approach can achieve much better performance than existing LBF designs providing reductions of the FPR of up to 90% in some settings.**

***Index Terms*—Learned Bloom Filters; Networking; Machine Learning; URL classification.**

I. Introduction

Bloom filters are widely used in software defined networking for example to manage flow tables [1], to classify packets [2] or to reduce the forwarding state [3]. Bloom filters can efficiently implement approximate membership checking and many variants and optimizations have been proposed over the years [4].

A recent use of machine learning is to optimize the implementation of Bloom filters. In particular, the Learned Bloom Filter (LBF) was introduced in [5]. The LBF combines a machine learning model with a traditional Bloom filter and is able to significantly reduce the total memory required to achieve a given false positive rate in many practical settings. The initial LBF has been subsequently extended by using additional Bloom Filters that are checked before [6] or after the machine learning model [7], [8] and further reduce the false positive rate. Also, schemes to support the use of the LBF in streaming applications on which elements are inserted and removed dynamically have been recently proposed [9].

An interesting observation that to the best of our knowledge has not been made before, is that in both the original LBF design and in its extensions and optimizations, the machine learning model is given and fixed. This means that the learned Bloom filters can potentially be further optimized by considering the machine learning model as another variable in the design process. In more detail, for example, different values of the parameters of the learner can be used to build different LBF instances with the same memory budget and the one with the lowest FPR can be selected.

In this paper, we make the following contributions:

1) Explore the use of the learner as a variable in the design of LBFs and evaluate its performance.
2) Show that the proposed approach can reduce significantly the FPR for different datasets and machine learning models with reductions of up to 90%.

The rest of the paper is organized as follows. In section II, Bloom filters and learned Bloom filters are discussed to provide the background needed for the rest of the paper. Section III, introduces and discusses the optimization of learned Bloom filters by considering the machine learning model as a variable in the process. The proposed approach is evaluated in section IV for a practical case study to show its potential benefits. Finally, the paper ends with the conclusion and some ideas for future work in section V.

II. Bloom Filters and Learned Bloom Filters

This section covers the background of Bloom filters and learned Bloom filters.

A. Bloom Filter (BF)

Bloom filters (BFs) were introduced more than fifty years ago to provide approximate set representation with fast membership testing and small memory footprint [10]. BFs have been widely applied in different fields such as computing and networking [11]. Though previous studies have provided the theoretical limit of BF [12], the optimization and application of BF remains an active research area [13].

A Bloom filter is a bit array of size M initialized with zero. To compress a set of keys, S, to the bit array, every key x is mapped to k different buckets using k independent hash functions, $h_1, h_2, ..., h_k$. To insert the key x, the bit values of $h_i(x)$ are set to one. Conversely, to check whether a query q is in the set, we return positive if all the bit values of $h_i(x)$ are set to one. Otherwise a negative is returned. The design of BF ensures zero false negative rate (FNR). However, BF may generate some false positives. It happens when all the hashing locations of x, $h_i(x)$, collide with the keys.

In expectation, the FPR can be approximated when M is large by:

$$\mathbb{E}(FPR) \approx \left(1 - e^{\frac{k \cdot n}{M}}\right)^k. \quad (1)$$

Eq. 1 suggests that the $\mathbb{E}(FPR)$ of BF depends on the ratio $\frac{n}{M}$. Inserting more keys into BF introduces a worse FPR. Given a target FPR level, eq. 1 can be used to determine the memory budget.

[1]Z. Dai and A. Shrivastava are with the Dept. of Statistics and Dept. of Computer Science, Rice {zhenwei.dai, anshumali}@rice.edu

[2]P. Reviriego and J.A. Hernández are with Universidad Carlos III de Madrid, Leganés 28911, {revirieg, jahgutie}@it.uc3m.es

B. Learned Bloom Filter (LBF)

Learned Bloom filter (LBF) incorporates a classification model to the BF. It can achieve a lower FPR by reducing the number of keys inserted into the BF. First, LBF trains a classification model on the available data to determine whether the given query x is positive or not based on the observed features. Then, LBF selects a threshold, T, where the query x is identified to be positive if $f(x) > T$. Otherwise, query x is passed to a backup filter to check its membership as shown. Similar to the standard BF, LBF also has no false negatives. The false positives of LBF can be either caused by the false positives of the classification model ($f(x|x \notin S) \geq T$) or that of the backup Bloom filter. If the learner classifies most of the keys accurately, LBF can insert fewer keys into the backup filter, which achieves a better trade-off and helps reducing the overall FPR. This initial design can be optimized by having a filter before and after the learned model as proposed in the sandwiched learned Bloom filter [6].

C. Adaptive Learned Bloom Filter (Ada-BF)

Adaptive learned Bloom filter (Ada-BF) improves the LBF by making use of the full spectrum of the prediction scores. LBF partitions the prediction score space into two regions, $f(x) \geq T$ and $f(x) < T$. When $f(x) \geq T$, the membership is fully determined by the learner (zero hash function is used). While $f(x) < T$, the membership of x is further determined using k hash functions by the backup filter. Ada-BF partitions the score space into multiple regions, and varies number of hash functions in different regions. Hence, Ada-BF achieves different FPRs across regions. Generally, most of the non-keys fall in the low score regions but only a few of them fall in the high score regions. And the keys have an opposite trend. Hence, in the low score regions, Ada-BF tends to use more hash functions to achieve a smaller FPR (for most the non-keys). While in the high regions, Ada-BF uses fewer hash functions and tolerates a higher FPR. By tuning the number of hash functions adaptively to the score distribution, Ada-BF could achieve a lower overall FPR compared to the LBF [7].

D. Partitioned Learned Bloom Filter (PLBF)

The partitioned learned Bloom filter [8] partitions the score space into multiple regions like the Ada-LBF and uses different Bloom filters for each of those regions. The main contributions of PLBF is to generalize Ada-LBF by using independent Bloom filters for each region rather than just varying the number of hash functions and to formulate the derivation of the regions and filter parameters as an optimization problem and providing and analytical solution. As a result, PLBF is expected to outperform the Ada-LBF in terms of FPR for a given memory budget.

III. Optimizing LBFs

The current methods to design LBFs to minimize the FPR for a given memory budget consider the following problem: *(P1) given a memory budget B, a learned model, and a target LBF architecture with parameters A, find the values of A that minimize the FPR of the LBF.* Therefore, the learned model is fixed and not part of the design process. Instead, the proposed approach to optimize LBFs can be formally formulated as this alternative problem: *(P2) given a memory budget B, a target learner algorithm with parameters P, and a target LBF architecture with parameters A, find the values of P and A that minimize the FPR of the LBF.* For example, we have a random forest algorithm whose parameters are the number of trees and leaves and an Ada-LBF whose parameters are the score regions, the number of hash functions for each region and the size of the backup Bloom filter and we want to determine the settings for all those parameters that minimize the FPR while using a memory smaller than a target size.

As discussed in the previous section, a learned Bloom filter is formed by a machine learning model and one or more backup filters. Therefore in the design of all the learned Bloom filters, the memory budget is divided in two parts, one for the Bloom filters and the other for the machine learning model. To achieve the lowest FPR for a given memory budget, we have to explore the optimal memory allocation to both parts. In the case of BF, the relationship between memory and performance (expected FPR) is well understood and can be evaluated by a simple closed form formula. Instead, for the machine learning models, the relation between memory and classification accuracy depends on the data set and algorithm used and can not be easily modeled. Intuitively, allocating more memory to the machine learning model improves the accuracy, but also reduces the memory available for the backup filter. Therefore, finding the learner that achieves the best trade-off between accuracy and model size is not straightforward.

The size of the learner depends both on the model structure and some hyper-parameters. For example, let us consider a Random Forest (RF) classifier. For a RF, the learner's size depends on the number of decision trees and the size of each of tree. Therefore, by selecting different number of trees and tree sizes, models that require different memory can be constructed. Similarly for a Support Vector Machine, the number of support vectors determines the amount of memory needed and for a neural network, the number of neurons and connections between them. Therefore, for each model structure, it is possible to provide implementations with different memory usage and accuracy by varying the model hyper-parameters. For the learned Bloom filters, given the machine learning model, there are still some hyper-parameters to tune in order to achieve the optimal performance, i.e. the threshold T in LBF.

Determining analytically the parameters of the machine learning model that will result in the LBF implementation with the lowest FPR is a challenging theoretical problem. Therefore, to show the potential benefits of the proposed approach, we use a simpler experimental method to find the best configuration by trying different choices of the hyper-parameters of the machine learning model, and testing the FPR achieved by the learned Bloom filters under each hyper-parameter setup. This allows us to evaluate the potential of considering the machine learning model as part of the LBF design process. The development of a theoretical framework to determine the best configuration is left for future work.

IV. Evaluation

To illustrate the potential benefits of the proposed approach, we test the FPR of three different learned Bloom filters, LBF, Ada-BF and Partitioned LBF, under several machine learning models: Random Forest (RF), Support Vector Machines (SVM) and Neural Networks (NNs) with a single hidden layer. Our experiments are performed on the malicious URLs and malware datasets used in [7]. The malicious URLs dataset includes 485,730 unique URLs, where 16.47% of them are tagged as malicious. We extracted 17 lexical features to the train of learner. The malware dataset includes 41,323 benign files and 96,724 viral files. As in [7], we also train the models using the well-known Python implementation, Scikit-Learn. By varying the number of leaves and number of classification trees, number of support vectors and number of neurons in the hidden layer we evaluate the effect of the classification models on the overall FPR achieved by the learned Bloom filters.

The three models are trained using 30% of the samples. 1) To train random forest models with different accuracy, we vary the number of trees from 6 to 15 and the number of leaves from 2 to 20; 2) For the SVM models, we raise the penalty weight C from 0.01×2^5 to 0.01×2^{20}. The scale of penalty weight is negatively correlated with the number of support vectors and model size; 3) The NNs are tuned through increasing the hidden layer dimension from 30 to 310.

To have a fair comparison of Ada-BF, PLBF and LBF under different learner setups, we fix the total memory budget = 400K, 600K 800K for the URL data set and of 400K, 500K and 600K for the malware dataset. The backup filter size equals to the memory budget minus the learner size. We randomly choose 30% samples to tune the hyper-parameters of LBF, PLBF and Ada-BF. Since the learned Bloom filters have no false negatives, the performance of learned Bloom filters is measured by the FPR.

The size of the models increases linearly with the number of trees and leaves, support vectors, and neurons. Instead, the marginal increment of the model accuracy decreases. Hence, when the model size becomes larger, it may not be beneficial to keep investing more memory on the learner to improve the model accuracy. The results for the RF are summarized in Figures 1 and 2. We can observe that the FPR decreases sharply as we start to increase the number of leaves. But after the number of leaves is larger than 4, the FPR starts to increase slowly though the FPR has some small fluctuations. Therefore, to optimize the performance of the learned Bloom filters, we may not choose the classification model with best prediction accuracy, suggesting the importance of jointly optimizing the learner and Bloom filters. Compared to the design in [7] where the number of trees is 10 and number of leaves is 20, jointly optimizing the Ada-BF, LBF, and PLBF reduces the FPR by around 70% under different memory budgets for the URL dataset, and up to 90% for the malware dataset (Table I). This is a remarkable improvement for real applications.

The results for SVMs and NNs are summarized in Figures 3 and 4 for the URL dataset. It can be seen that depending on the memory budget and learning filter type, the lowest FPR is obtained for different hyperparameter values in both the SVM and the NN. This confirms that considering the machine learning model as part of the learned Bloom filter design would reduce the FPR for a given memory budget. In the case of the NNs, there is a trend to increase the FPR as the number of neurons in the hidden layer increases suggesting that there is no benefit in investing more memory on the NN. For the SVM, the lowest FPRs are typically achieved by values in the middle of the penalty weight range explored.

V. Conclusions and Future Work

Learned Bloom filters that combine a machine learning model with Bloom filters have shown significant reductions in false positive rate over traditional Bloom filters. In this paper, we proposed to use the machine learning model as a variable in the design showing that it can further reduce the false positive rate of learned Bloom filters. Our experiments suggest that tuning the learner can significantly reduce the FPR and improves the learned Bloom filters significantly. Our findings provide a strong motivation to further study the optimization of learned Bloom filters considering the learner model as one of the design elements. Future work can for example explore how to find the best parameters for the learned model without exhaustively testing all the possible combinations and to formalize the optimization problem.

Acknowledgment

The authors would like to acknowledge the support of the EU H2020 project PIMCITY (grant no. H2020-871370) to the development of this work.

References

[1] R. Challa, Y. Lee, and H. Choo, "Intelligent eviction strategy for efficient flow table management in openflow switches," in *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, 2016, pp. 312–318.

[2] P. Reviriego, J. Martínez, D. Larrabeiti, and S. Pontarelli, "Cuckoo filters and bloom filters: Comparison and application to packet classification," *IEEE Transactions on Network and Service Management*, pp. 1–1, 2020.

[3] A. Craig, B. Nandy, and I. Lambadaris, "Forwarding state reduction for multi-tree multicast in software defined networks using bloom filters," in *IEEE International Conference on Communications (ICC)*, 2019.

[4] L. Luo, D. Guo, R. T. B. Ma, O. Rottenstreich, and X. Luo, "Optimizing bloom filter: Challenges, solutions, and comparisons," *IEEE Communications Surveys Tutorials*, vol. 21, no. 2, pp. 1912–1949, 2019.

[5] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis, "The case for learned index structures," in *Proceedings of the 2018 International Conference on Management of Data*, ser. SIGMOD '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 489–504.

[6] M. Mitzenmacher, "A model for learned bloom filters, and optimizing by sandwiching," *arXiv*, 2019.

[7] Z. Dai and A. Shrivastava, "Adaptive learned bloom filter (ada-bf): Efficient utilization of the classifier," *arXiv*, 2019.

[8] K. Vaidya, E. Knorr, T. Kraska, and M. Mitzenmacher, "Partitioned learned bloom filter," *arXiv*, 2020.

[9] Q. Liu, L. Zheng, Y. Shen, and L. Chen, "Stable learned bloom filters for data streams," *Proc. VLDB Endow.*, vol. 13, no. 12, p. 2355–2367, Jul. 2020. [Online]. Available: https://doi.org/10.14778/3407790.3407830

[10] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, p. 422–426, Jul. 1970. [Online]. Available: https://doi.org/10.1145/362686.362692

[11] S. Tarkoma, C. E. Rothenberg, and E. Lagerspetz, "Theory and practice of bloom filters for distributed systems," *IEEE Communications Surveys Tutorials*, vol. 14, no. 1, pp. 131–155, 2012.

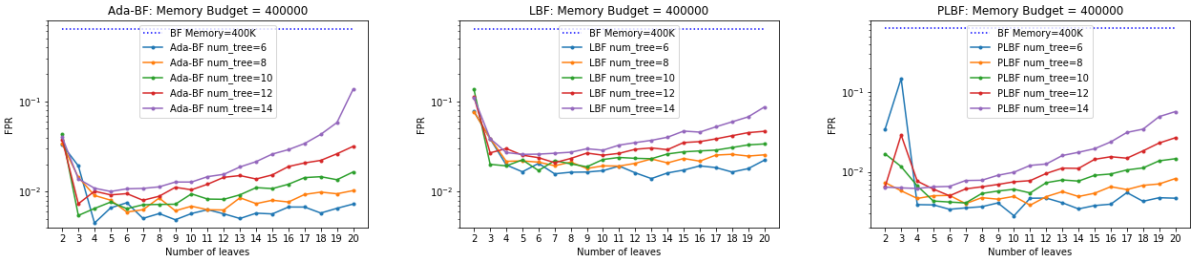
[12] L. Carter, R. Floyd, J. Gill, G. Markowsky, and M. Wegman, "Exact and approximate membership testers," in *Proceedings of the tenth annual ACM symposium on Theory of computing*. ACM, 1978, pp. 59–65.

[13] P. Reviriego and O. Rottenstreich, "The tandem counting bloom filter - it takes two counters to tango," *IEEE/ACM Transactions on Networking*, vol. 27, no. 6, pp. 2252–2265, 2019.

URL	Ada-BF		LBF		PLBF	
memory	MS (T, L)	opt FPR ([7])	MS (T, L)	opt FPR ([7])	MS (T, L)	opt FPR
M=400K	52.7K (6, 4)	0.450% (1.655%)	113.6K (6, 13)	1.400% (3.384%)	93.3K (6, 10)	0.281% (1.460%)
M=600K	52.7K (6, 4)	0.087% (0.244%)	52.7K (6, 4)	0.280% (0.963%)	120.3K (6, 14)	0.050% (0.200%)
M=800K	133.8K (6, 16)	0.022% (0.041%)	52.7K (6, 4)	0.083% (0.250%)	79.7K (6, 8)	0.013% (0.040%)

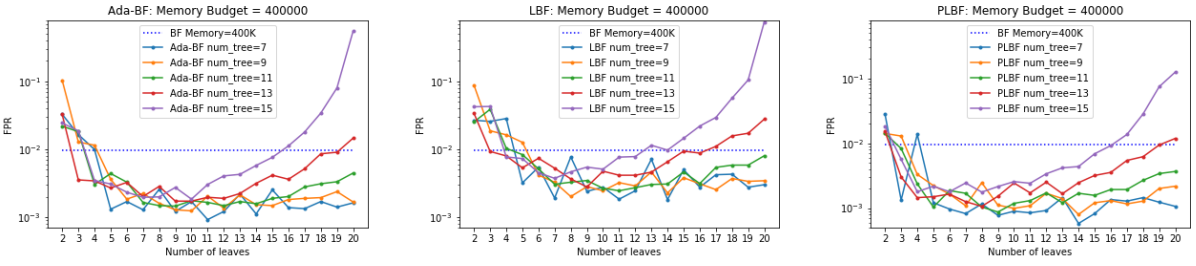
Malware	Ada-BF		LBF		PLBF	
memory	MS (T, L)	opt FPR ([7])	MS (T, L)	opt FPR ([7])	MS (T, L)	opt FPR
M=400K	115.3K (7, 11)	0.092% (0.308%)	138.9K (7, 14)	0.181% (0.728%)	138.9K (7, 14)	0.058% (0.217%)
M=500K	154.7K (7, 16)	0.017% (0.162%)	115.3K (7, 11)	0.097% (0.453%)	138.9K (7, 14)	0.039% (0.094%)
M=600K	236.9K (9, 20)	0.010% (0.056%)	154.7K (7, 16)	0.060% (0.281%)	115.3K (7, 11)	0.015% (0.070%)

TABLE I: The size of optimal classification model and comparison of the FPR. The column MS (T, L) gives the size of the learner and the corresponding number of trees and number of leaves. The column opt FPR ([7]) provides the optimal FPR and the FPR achieved under the learner setup in [7].



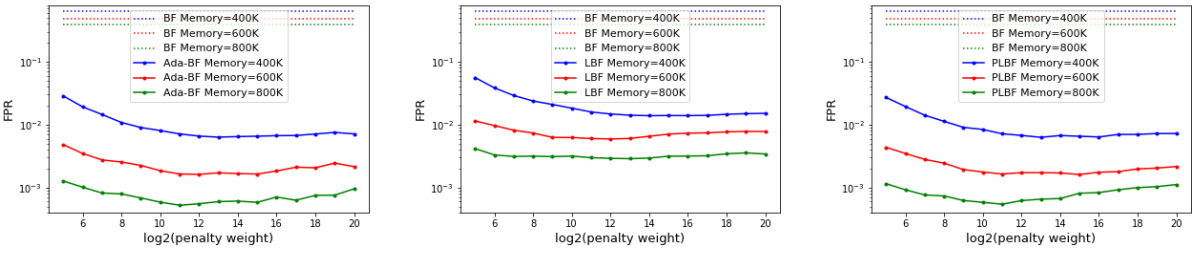
(a) FPR of Ada-BF with different learners (b) FPR of LBF with different learners (c) FPR of PLBF with different learners

Fig. 1: Performance of Ada-BF, LBF, and PLBF with a Random Forest in the malicious URL dataset



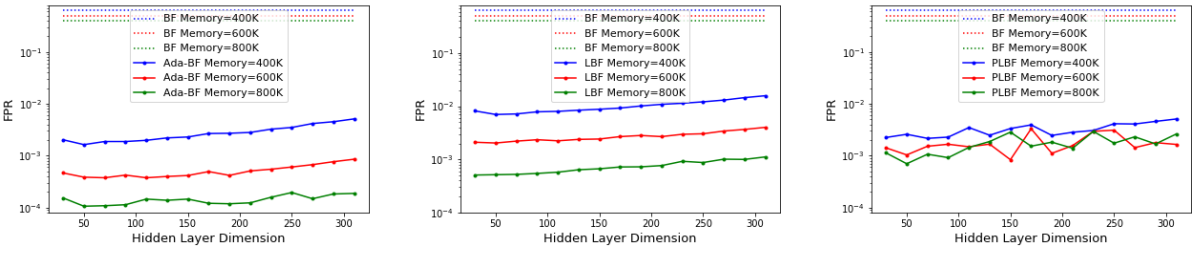
(a) FPR of Ada-BF with different learners (b) FPR of LBF with different learners (c) FPR of PLBF with different learners

Fig. 2: Performance of Ada-BF, LBF, and PLBF with a Random Forest in the malware dataset



(a) FPR of Ada-BF with different learners (b) FPR of LBF with different learners (c) FPR of PLBF with different learners

Fig. 3: Performance of Ada-BF, LBF, and PLBF with a SVM in the malicious URL dataset



(a) FPR of Ada-BF with different learners (b) FPR of LBF with different learners (c) FPR of PLBF with different learners

Fig. 4: Performance of Ada-BF, LBF, and PLBF with a single hidden layer NN in the malicious URL dataset