CSC2525H Project Report
Expandable Filters: To Grow or not to Grow

Navid Eslami

April 5, 2026

This project explores how to leveraged rejuvenation operations in expandable filters (InfiniFilter and Aleph filter) to achieve a better memory footprint while providing a constant false positive rate using fingerprint widening. After describing a general and theoretically grounded solution, we implement our ideas on top of Aleph filter and demonstrate via empirical measurements how they save significant amounts of memory (up to 1.3× in our evaluations), even when as little as a 15% of the keys are rejuvenated before each expansion.

Introduction

Rejuvenation operations extend old/shortened fingerprints by rehashing the original key to improve their False Positive Rate (FPR) with no extra memory cost. Their opportunistic nature, however, means that their benefit is largely workload-dependent, as extending fingerprints requires a query to an existing key. On the other hand, to stabilize the FPR, InfiniFilter and Aleph filter both utilize a *Widening Regime* to allocate wider slots during each expansion, create longer fingerprints for new keys, and ultimately achieve a constant false positive rate bound, regardless of the dataset size. This regime becomes wasteful, however, if many of the keys are rejuvenated at expansion time, as the wider slots serve to counteract the shorter fingerprints, which do not exist anymore in this case.

The following research question then arises: "*Is it possible to design an expandable filter that widens its slots at each expansion just enough so that it (1) reaps the benefits of rejuvenations as much as possible, (2) uses the least amount of memory, and (3) provides a constant FPR as close as possible to the initial FPR across expansions, all at the same time?*"

We devise an *Adaptive Expansion Scheme* that achieves the above three design goals. Our scheme keeps track of a histogram of fingerprint counts, specifying how many fingerprints of a specific length are are the filter. Using this histogram, it can then precisely determine its FPR at any point. It then makes this FPR value follow a *Converging Sequence of FPRs*, allowing it to maintain a constant FPR guarantee across expansions. It achieves an improved memory footprint by following this sequence of FPRs as slowly as possible.

Solution

One might approach this problem from an *Online Algorithms with Adversaries* perspective, but it quickly becomes apparent that the adversary can only help the algorithm by doing rejuvenations. Thus, the pathological worst-case FPR happens when there are no rejuvenations whatsoever. Our solution must ideally maintain the old FPR guarantees (i.e., asymptotically constant FPR) in this case while also benefiting from the contributions of an "Altruistic Adversary" that rejuvenates the fingerprints.

Bounding Worst-Case FPR

To ensure a worst-case guarantee, we consider a sequence of FPRs $\epsilon_0, \epsilon_1, \epsilon_2, \ldots$ that we plan to mimic with our filter. The idea is as follows: *if we can enforce this sequence of FPRs as the filter's FPR for all expansions, we have designed a filter that achieves the asymptotic FPR behavior of this sequence in the worst-case.*

Note that this limiting behavior of the sequence can range from being a constant $\epsilon^* = \lim_{n\to+\infty} \epsilon_n$ to being an arbitrary function $f(n)$ where $\lim_{n\to+\infty}|\epsilon_n - f(n)|= 0$. One such FPR sequence, presented in the InfiniFilter paper, is $\epsilon_n = \sum_{i=1}^{n} \frac{\epsilon_0}{i^2}$. By virtue of the Basel problem, we have that $\lim_{n\to+\infty} \epsilon_n = \frac{\epsilon_0 \pi^2}{6}$. Thus, this sequence provides a limiting FPR guarantee that is only a constant factor above the initial FPR.

We now prove Lemma 1:

Lemma 1. *Given an expandable filter with an FPR of ϵ just before expansion, if we increase its fingerprint size to f bits, its worst-case FPR at the next expansion becomes $\epsilon + 2^{-f-1}$. We assume that an expansion doubles the size of the filter.*

Proof. In the worst-case scenario, no rejuvenations happen until the next expansion, implying that only the newly inserted keys have f-bit fingerprints. These new keys comprise only half of the fingerprints of the filter, and the old fingerprints comprise the other half. However, the old fingerprints had to sacrifice their least significant bit, doubling their FPR. A non-existent key at the moment of the next expansion will thus map to one of the new fingerprints with a probability of $\frac{1}{2}$, in which case it will result in a false positive with probability 2^{-f}. It may also map to one of the old fingerprints with a probability of $\frac{1}{2}$ and will thus be a false positive with a probability of at most 2ϵ. Therefore, the overall new FPR must be $\epsilon' = \frac{1}{2} \cdot 2\epsilon + \frac{1}{2} \cdot 2^{-f} = \epsilon + 2^{-f-1}$. □

One can see from the proof of this lemma that the sequence of FPRs must be increasing, no matter what fingerprint size we choose at expansion time. Furthermore, it aids in upper bounding the new fingerprint size at each expansion by simply comparing the current FPR to the next FPR from the sequence.

Thus, using Lemma 1 and assuming that we can compute the filter's FPR and its worst-case FPR before the next expansion (when its fingerprints are truncated to some fixed and known length), we come up with Algorithm 1:

Algorithm 1 Adaptive expansions by following a converging FPR sequence.

Given FPR sequence $\epsilon_0, \epsilon_1, \epsilon_2, \ldots$.
Assume that the filter is expanding.
$\epsilon \leftarrow$ current FPR of the filter.
$n \leftarrow \min_n(n \mid \epsilon \le \epsilon_n)$ ▷ The step of the sequence to follow.
$f_{\text{new}} \leftarrow \max(f_{\text{old}}, \lceil -\log_2(\epsilon_{n+1} - \epsilon) - 1 \rceil)$
▷ The new fingerprint size is the maximum of the old fingerprint size and the result of Lemma 1.
$\epsilon' \leftarrow$ Worst-case FPR before the next expansion, if all fingerprints are truncated to at most f_{new} bits.
while $\epsilon' < \epsilon_{n+1}$ **do** ▷ While still in a "safe" FPR range.
  $f_{\text{new}} \leftarrow f_{\text{new}} - 1$ ▷ Truncate the fingerprints further to save memory.
  $\epsilon' \leftarrow$ Worst-case FPR before the next expansion, if all fingerprints are truncated to at most f_{new} bits.
end while
Use $f_{\text{new}} + 1$ bits for the new slots of the filter, i.e., also make room for the unary age counter.

Algorithm 1 works as follows when there are no rejuvenations: starting with a filter with an FPR of ϵ_0, it widens the slots of the filter during the k-th expansion according to Lemma 1 so that the resulting filter has a worst-case FPR of at most ϵ_{k+1} just before the $(k+1)$-th expansion. It then truncates the fingerprints of the filter as much as possible while maintaining the ϵ_{k+1} upper bound on the worst-case FPR. Thus, since the filter closely mimics the FPR sequence $\epsilon_0, \epsilon_1, \epsilon_2, \ldots$ and by Lemma 1, it exhibits the same limiting FPR behavior as this sequence. In the case of the example sequence introduced earlier, this translates to a constant FPR across all expansions.

Utilizing Rejuvenations

Notice that Algorithm 1 did not assume that there are no rejuvenations. If rejuvenations are present, they affect the current FPR of the filter and thus will change how Algorithm 1 chooses n and how much it truncates the fingerprints of the filter. For example, if all of the fingerprints in the filter are rejuvenated before expansion, the filter will achieve an FPR $\epsilon \le \epsilon_0$. Therefore, Algorithm 1 will truncate the fingerprints and shrink the slot widths to the point that the result reflects a filter with an FPR of $\epsilon' \le \epsilon_1$ after the expansion.

Considering the FPR sequence of the Basel problem in this scenario, the filter will have fingerprints of length $\lceil -\log_2(\epsilon_1 - \epsilon) - 1 \rceil$, which is very close to the initial memory consumption of the filter.

Notice that throughout the lifetime of the filter, Algorithm 1 always follows the FPR sequence $\epsilon_0, \epsilon_1, \epsilon_2, \ldots$. This means that if rejuvenations stop happening, the filter can continue assigning the FPR of the sequence to the filter, allowing its FPR to achieve the asymptotic properties of the sequence.

Thus, if the filter can determine its FPR and its worst-case FPR before the next expansion, Algorithm 1, along with Lemma 1, allows us to fully take advantage of rejuvenations to minimize the memory consumption of the filter while also providing the same constant FPR guarantee as before.

Bounding the Filter's FPR

We can easily upper bound the filter's FPR using a histogram c_i. c_i represents the number of fingerprints in the filter with length i for all $0 \leq i \leq f$, where f is the current maximum fingerprint size. Given this histogram, and assuming that the filter has 2^k slots, one can see that a non-existent key may map to the canonical slot of a fingerprint of length i and match its fingerprint value with a probability of at most $\frac{1}{2^k} \cdot 2^{-i}$. Applying a union bound on these probabilities, one can see that the filter's actual FPR ϵ must be at most $2^{-k} \cdot \sum_{i=0}^{f} c_i \cdot 2^{-i}$.

Moreover, this histogram also aids in calculating the worst-case FPR before the next expansion. Assuming that the filter chooses a new fingerprint size of f_{new} for the filter during the current expansion, the old fingerprints that were longer than f_{new} get truncated to this length, and all the $N \leq 2^k$ newly inserted keys will also have this length. Therefore, following the same argument as before, we can derive an upper bound of

$$2^{-k-1} \cdot \left[\sum_{i=0}^{f_{\text{new}}-1} c_i \cdot 2^{-i} + 2^{-f_{\text{new}}} \cdot (2^k + \sum_{i=f_{\text{new}}}^{f} c_i) \right]$$

on the worst-case FPR of the filter before the next expansion. Note that we assume the second summation evaluates to zero when $f_{\text{new}} > f$.

Thus, we plug these expressions into Algorithm 1 to complete its functionality and allow it to derive conservative FPR measurements and thus conservative fingerprint lengths during expansions. Note that this histogram c_i can be easily maintained by incrementing and decrementing the corresponding counters during insertions, deletions, rejuvenations, and expansions.

Evaluation

We evaluate our newly developed adaptive expansions algorithm and show that it creates filters with up to 1.5× smaller memory footprints, even when as little as 15% of the keys are rejuvenated before each expansion.

Setup

We use an HP 15-ef1xxx laptop running Fedora 39 on an AMD Ryzen 7 4700U CPU with 8 cores and threads. It has a 256KiB L1 instruction cache and 256KiB, 4MiB, and 8MiB L1, L2, and L3 data caches, respectively. It also has 16GiBs of DRAM, as well as a 512GiB SSD. We do not utilize the SSD in our evaluations.

We implement our ideas on top of Aleph filter in Java and compare it to the Fixed-Width and Widening regimes of Aleph filter. We compile and run all baselines using JDK/JRE 17. The adaptive expansion scheme and the Widening regime both use the sequence $\epsilon_n = \epsilon_0 \cdot \sum_{i=1}^{n} \frac{1}{i^2}$ as their FPR sequences. The updated codebase of Aleph filter is accessible from the `AlephFilter` directory in this report's archive.

Workloads

As we are working with point filters based on Quotient filters, the key and query distribution do not impact the performance or FPR of the structure. Therefore, we are free to choose the workload that is most

convenient to implement.

In all our experiments, we instantiate filters with 12-bit fingerprints and 2^{12} slots. We then progressively insert keys into each filter, starting from 0 and going up, until the filter expands 12 times. This procedure results in a total of $2^{24} \approx 1.7 \cdot 10^7$ inserts into the filter. We measure and record insertion throughput before each expansion of the filter.

Before each expansion, some fraction $\alpha \in [0, 1]$ of the fingerprints in the filter are rejuvenated by taking a sample of size $\alpha \cdot N$ of the N keys inserted into the filter without replacement. The fraction of rejuvenated keys α is chosen on a per-experiment basis. Similarly to inserts, we measure and record rejuvenation throughput before each filter expansion.

To measure the FPR and query throughput of the filter, we execute 1M non-existent queries on the filter before each expansion.

Experiment 1: Consistent Rejuvenations

In this experiment, we fix the rejuvenation fraction α to be a constant across all expansions. This experiment simulates the natural scenario where queries come into the system and concurrently rejuvenate a fraction α of the keys before each expansion.

Figure 1 showcases the results of this experiment. The dotted lines denote no rejuvenations, i.e., $\alpha = 0$, dash-dotted lines correspond to the case of $\alpha = 5\%$, dashed lines denote the case of $\alpha = 10\%$, and solid lines denote the case where $\alpha = 15\%$.

One can see that when there is no rejuvenation, our adaptive expansion scheme behaves similarly to the Widening regime. It achieves a very close FPR to it and uses up to 14% less memory. The reason is that the Widening regime widens its fingerprints too much and provides an overkill FPR. However, adaptive expansions cause the filter to widen its slots just enough and follow the same FPR sequence as the Widening regime. The Fixed-Width regime exhibits logarithmically increasing FPRs with respect to the dataset size.

As α increases to 15%, one can see that the adaptive expansion scheme gets closer and closer to maintaining its initial FPR and memory footprint. The Widening regime, on the other hand, uses the same memory it would have used when $\alpha = 0$ and achieves an extremely low FPR, which may very well be overkill for the application. Lastly, even though the Fixed-Width regime maintains its original memory footprint, it has a much worse FPR than the adaptive scheme that increases similarly to a super-constant function.

Notice that when $\alpha = 15\%$, the adaptive expansion scheme periodically switches between using 21 and 23 bits per key, creating a saw-tooth pattern. This is due to the adaptive regime provisioning enough memory to perform well in the worst-case scenario. However, this provisioning is overkill in this workload since many of the fingerprints are rejuvenated. After this overkill behavior occurs, the filter realizes it can shrink its fingerprints to achieve the same FPR a before with less memory.

We also note from Figure 1 that all these filters have similar insertion/rejuvenation/query throughput. Therefore, we can conclude that even when a small fraction of the keys are rejuvenated, the adaptive regime can effectively achieve a smaller memory footprint by up to $\frac{30}{23}\times \approx 1.3\times$ compared to the Widening regime while providing the same constant FPR guarantee across expansions.

Experiment 2: Sudden Rejuvenations

In this experiment, we showcase how the adaptive expansion scheme can dial back its memory consumption if some fraction α of the keys are suddenly rejuvenated. We thus execute the same workloads as Experiment 1 but only rejuvenate from the 6th expansion onward.

Figure 2 shows that before any rejuvenations occur, the adaptive expansion scheme maintains its constant FPR guarantee similarly to the Widening regime, while the Fixed-Width regime exhibits logarithmically increasing FPRs. However, as soon as the rejuvenations start, the adaptive expansion scheme achieves an FPR and memory footprint close to its initial FPR and memory footprint. In contrast, the Widening regime achieves an overkill FPR with a much higher memory footprint in comparison, and the Fixed-Width regime achieves a smaller FPR that is larger than the FPR achieved by the adaptive expansion scheme.

Similarly to Experiment 1, Figure 2 shows that all the filters exhibit similar insertion/rejuvenation/query throughputs. We can thus conclude that the adaptive expansion scheme can fully take advantage of reju-

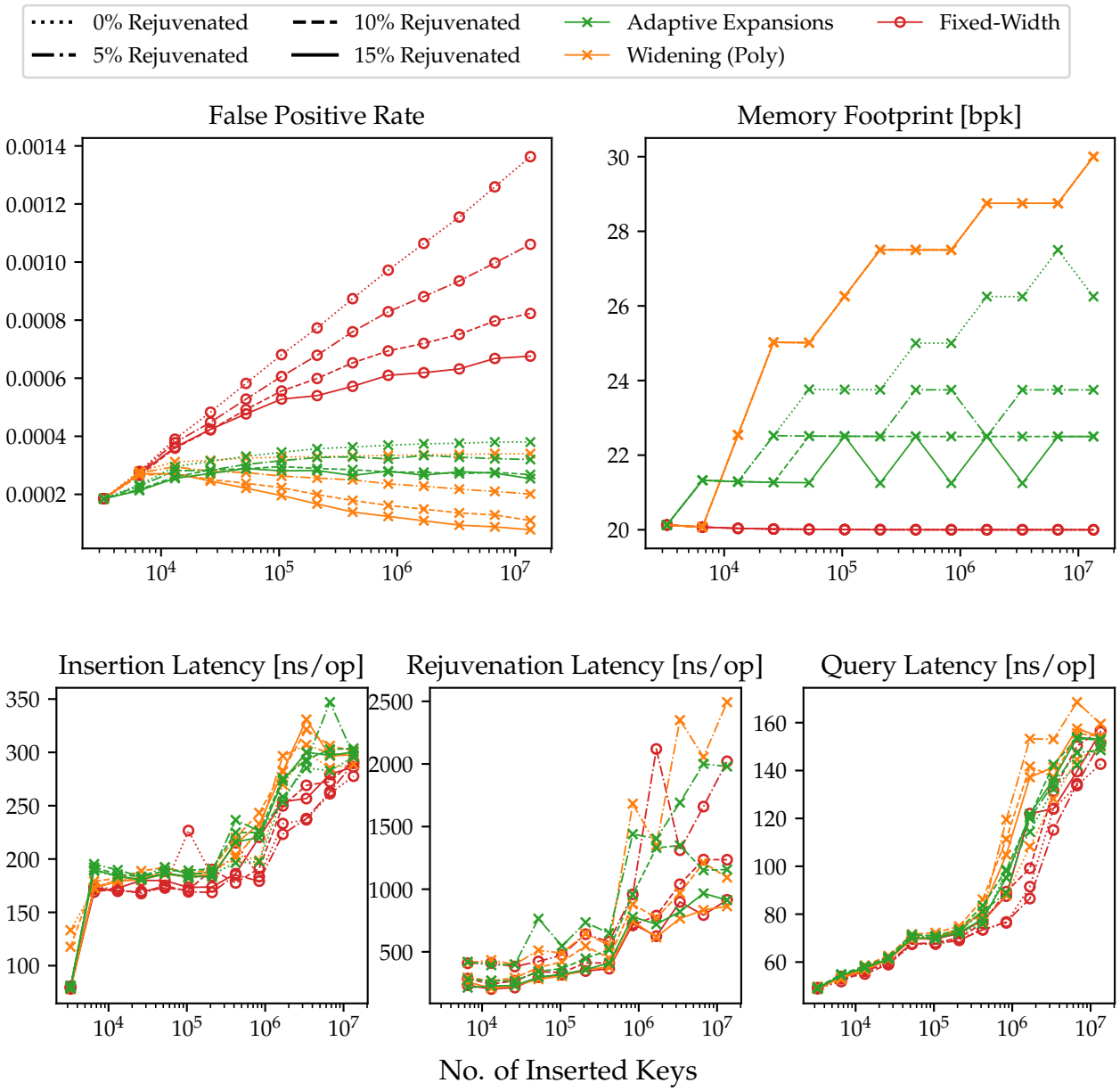

Figure 1: FPR, memory footprint, and insertion/rejuvenation/query latencies measured for different fractions of rejuvenations α occurring at all points in time. The adaptive expansion scheme achieves superior performance in all scenarios, beating both the Widening and Fixed-Width regimes.

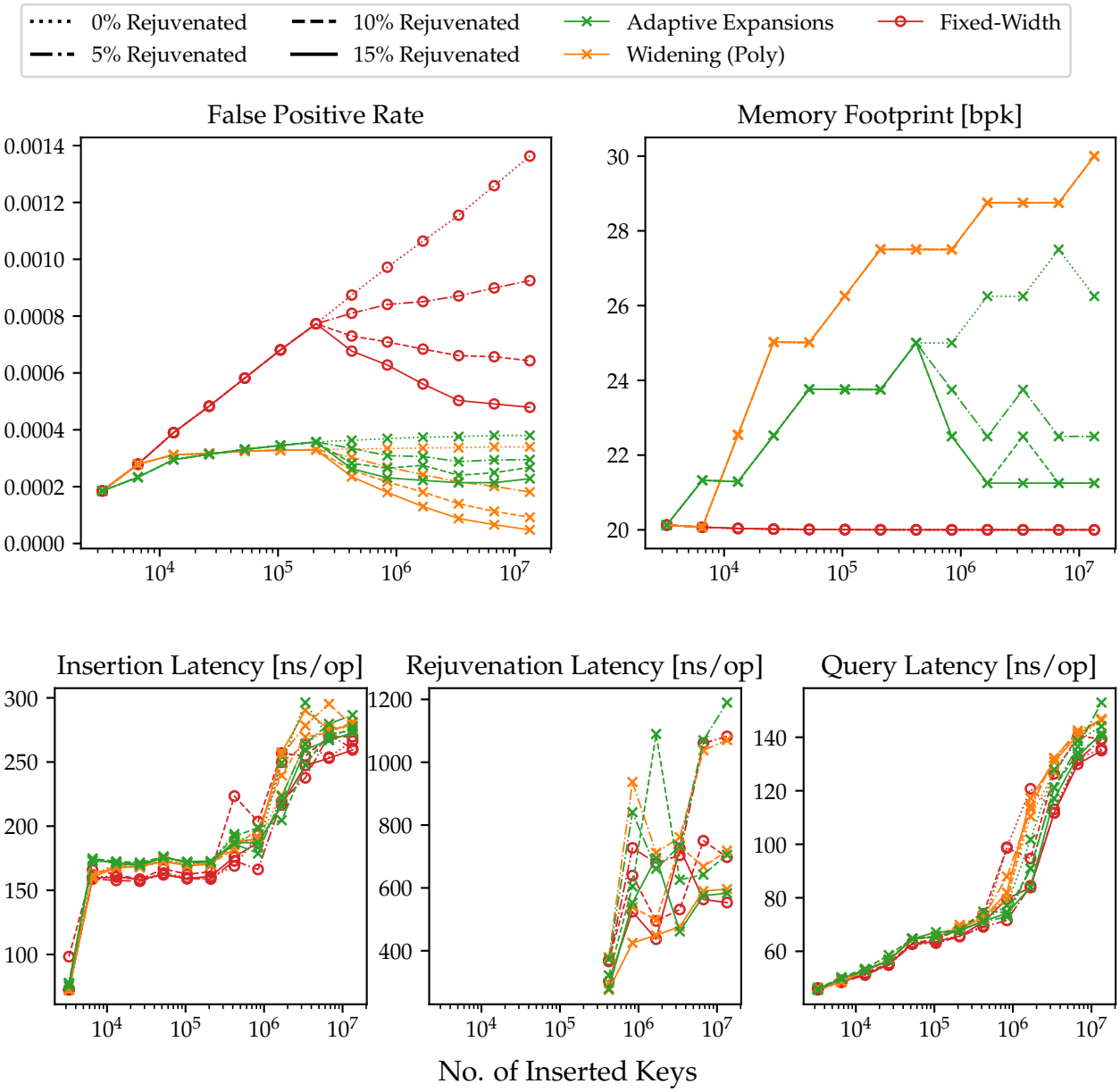

Figure 2: FPR, memory footprint, and insertion/rejuvenation/query latencies measured for different fractions of rejuvenations α when the only the second half of the expansions receive rejuvenations. The adaptive expansion scheme once again achieves superior performance in all scenarios, beating both the Widening and Fixed-Width regimes.

venations, even if they happen with a long delay, and exhibits the good qualities of the Widening and Fixed-Width regimes simultaneously.

Experiment 3: Rejuvenation Cutoff

In this experiment, we demonstrate how the adaptive expansion scheme can gracefully increase its memory footprint when rejuvenations with a fraction of α suddenly stop happening. That is, we execute the same workloads as Experiment 1 but only rejuvenate until 6th expansion.

Figure 3 shows that before the rejuvenations stop, the adaptive expansion scheme maintains an FPR close to its initial FPR, while the Fixed-Width scheme achieves a higher FPR in comparison, and the Widening regime achieves an overkill FPR at the cost of more memory. However, as soon as the rejuvenations stop, the adaptive expansion scheme gracefully starts to behave like the Widening regime and achieves its FPR guarantees at a much lower memory cost. On the other hand, the Widening regime achieves a much better FPR than intended, while the Fixed-Width regime exhibits logarithmically increasing FPRs.

As in the case of Experiments 1 and 2, Figure 3 shows that all filters exhibit similar insertion, rejuvenation, and query throughputs.

We conclude that the adaptive expansion scheme can degrade gracefully and has the good qualities of the Widening and Fixed-Width regimes.

Conclusion and Future Directions

In this report, we designed, implemented, and benchmarked a theoretically sound approach to utilizing rejuvenations in expandable filters to achieve a target FPR behavior across expansions while minimizing the memory footprint of the structure. We analytically showed that a filter can widen its slots such that it mimics a pre-determined sequence of FPRs and thus exhibit the limiting behavior of that sequence. During this mimicking process, it can widen its slots less if many of the fingerprints of the filter are rejuvenated. We empirically showed that even when a small fraction of the fingerprints (around 15%) are rejuvenated before each expansion, the adaptive expansion scheme enables the filter to achieve a constant memory footprint while also maintaining constant FPR guarantees across expansions.

The ideas of this project can be expanded upon in one of the following ways:

- Even though the sequence of reciprocals of squares provides a constant limiting FPR, it is not known whether this series is the best choice available, even though it has an asymptotically optimal memory footprint vs. FPR tradeoff in the worst-case. Can we find a sequence of FPRs with a constant limit that is "better" than this sequence? In what sense is this sequence superior to the reciprocals of squares?

- Even though a pre-defined sequence of FPRs to follow is sufficient to create a filter with a certain limiting FPR behavior, it is a clunky and hard-to-use tuning knob. What is the most intuitive interface to expose to practitioners to allow them to choose their preferred asymptotic FPR vs. memory tradeoff?

We believe that the answers to these questions will lead to easily deployable expandable filter designs, significantly improving the development of systems.

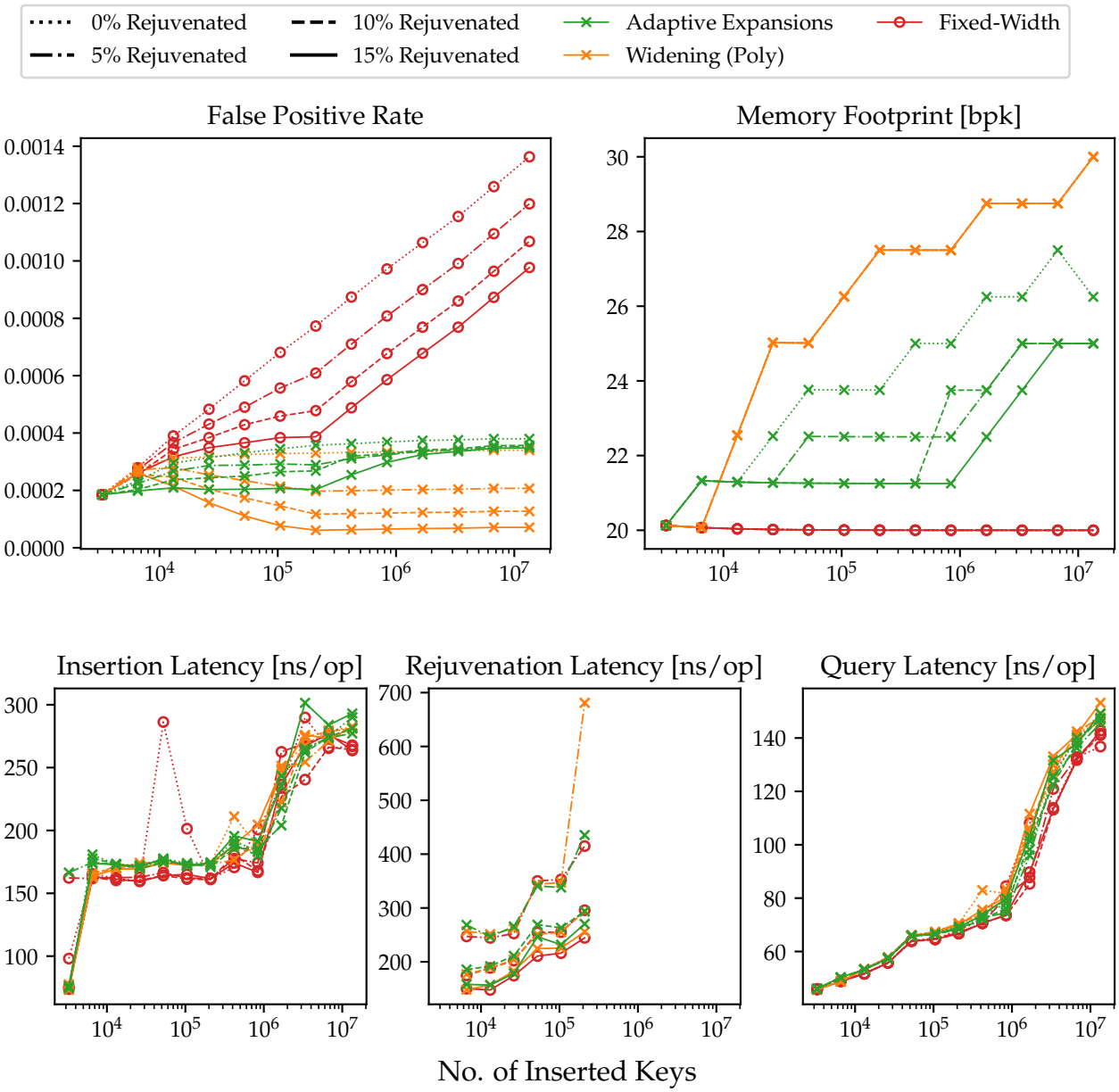

Figure 3: FPR, memory footprint, and insertion/rejuvenation/query latencies measured for different fractions of rejuvenations α when the only the first half of the expansions receive rejuvenations. The adaptive expansion scheme dominates its competitors once again.