Space Lower Bounds for Dynamic Filters and Value-Dynamic Retrieval

William Kuszmaul
Harvard University
Cambridge, USA
william.kuszmaul@gmail.com

Stefan Walzer
KIT
Karlsruhe, Germany
stefan.walzer@kit.edu

ABSTRACT

A *filter* is a data structure that answers approximate-membership queries on a set S of n elements, with a false-positive rate of ϵ. A filter is said to be *dynamic* if it supports insertions/deletions to the set S, subject to a capacity constraint of n.

This paper considers the space requirement of filters, regardless of running time. It has been known for decades that static filters have optimal space $n \log \epsilon^{-1} + O(1)$ expected bits, and that dynamic filters can be implemented in space $n \log \epsilon^{-1} + \Theta(n)$ bits. We prove that this $\Theta(n)$-bit gap is fundamental: any dynamic filter must use $n \log \epsilon^{-1} + \Omega(n)$ bits, no matter the choice of ϵ.

Extending our techniques, we are also able to obtain a lower bound for the *value-dynamic* retrieval problem. Here again, we show that there is a $\Theta(n)$-bit gap between the optimal static and (value-)dynamic solutions.

CCS CONCEPTS

• **Theory of computation → Data structures design and analysis**; *Cell probe models and lower bounds.*

KEYWORDS

Space lower bound, Approximate membership data structure, Bloom filter, Retrieval data structure

ACM Reference Format:
William Kuszmaul and Stefan Walzer. 2024. Space Lower Bounds for Dynamic Filters and Value-Dynamic Retrieval. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing (STOC '24), June 24–28, 2024, Vancouver, BC, Canada.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3618260.3649649

1 INTRODUCTION

In recent decades, filters have emerged as one of the most widely used data structures in computer science (see, e.g., the many surveys on the topic [1, 8, 9, 30, 36, 42, 43, 45, 46]). They allow for space-constrained applications to have something 'similar' to a hash table, but that uses much less space.

Formally, a *filter* solves the *approximate set membership problem*. Given a universe U of size u and an input set $S \subseteq U$ of some size n, the data structure D must implement a function $\texttt{query}_D : U \to \{\text{true}, \text{false}\}$ such that

$$\begin{aligned} \Pr[\texttt{query}_D(x) = \text{true}] &= 1 && \text{if } x \in S \\ \Pr[\texttt{query}_D(x) = \text{false}] &\geq 1 - \epsilon && \text{otherwise.} \end{aligned}$$

Whereas queries to items in S are exact (they always return true), queries to elements *not* in S are approximate (they *usually* return false). The parameter ϵ is called the *false positive rate*. A filter is said to be *dynamic* if it supports not just queries, but also insertions and deletions to S. In this case, n serves as a *maximum capacity*, upper bounding the size of S at any given moment.

It has been known for decades that the optimal expected space complexity of a *static* filter is $n \log \epsilon^{-1} + O(1)$ bits [10, 17, 44] where log denotes the binary logarithm throughout this paper. It has also been known for decades how to construct *dynamic* filters using $n \log \epsilon^{-1} + \Theta(n)$ bits [2, 7, 10, 37]. It has remained open, however, whether this $\Theta(n)$-bit gap is fundamental: do dynamic filters need to store $n \log \epsilon^{-1} + \Omega(n)$ bits?

Tight bounds on incremental filters. In a 2010 breakthrough, Lovett and Porat [29] established a lower bound for *incremental* filters: any filter supporting `query` and `insert` must use $n \log \epsilon^{-1} + nf(\epsilon)$ bits, for some positive non-increasing function $f : (0, 1) \to \mathbb{R}^+$. As an immediate corollary, they concluded that in the case where $\epsilon = \Theta(1)$, the optimal space complexity for an incremental (or for a dynamic) filter is $n \log \epsilon^{-1} + \Theta(n)$ bits.

For $\epsilon = o(1)$, the lower bound deteriorates to $n \log \epsilon^{-1} + o(n)$ bits. This might seem like a technical oddity, but we show in Section 5 that it is unavoidable: there is a simple incremental filter that, when $\epsilon = o(1)$, uses $n \log \epsilon^{-1} + o(n)$ bits.

Tight bounds on dynamic filters. Thus, the unresolved case is that of dynamic filters with $\epsilon = o(1)$. Is it possible to achieve $n \log \epsilon^{-1} + o(n)$ bits in this setting, or is there an unavoidable $\Omega(n)$-bit cost to dynamism?

The main result of this paper is to resolve this question by giving a lower bound of $n \log \epsilon^{-1} + \Omega(n)$ bits on the size of any dynamic filter (Section 3). The lower bound is information-theoretic, so it applies regardless of time efficiency.

More precisely, we show that for any $\epsilon = o(1)$, and a universe of size $|U| = \omega(\epsilon^{-1} n)$, dynamic filters must use expected space

$$n \log \epsilon^{-1} + \frac{n}{2} \log \frac{4}{\sqrt{17}-1} - o(n) \geq n \log \epsilon^{-1} + 0.35 \cdot n - o(n)$$

bits. Combining this with Lovett and Porat's lower bound for $\epsilon = \Theta(1)$ gives an $n \log \epsilon^{-1} + \Omega(n)$ lower bound for all ϵ.

Our lower bound is achieved through a series of communication-protocol arguments. We classify filters based on what we call their *churn rate*, and we show that no matter what the churn rate of a filter is, there exists a one-way communication protocol that uses the filter in order to encode $n \log \epsilon^{-1} + \Omega(n)$ bits of information.

STOC '24, June 24–28, 2024, Vancouver, BC, Canada

ACM ISBN 979-8-4007-0383-6/24/06
https://doi.org/10.1145/3618260.3649649

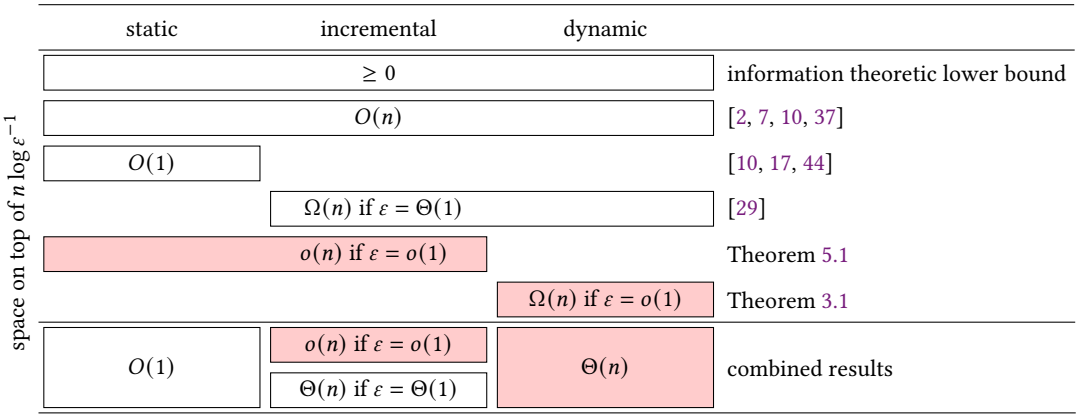

Figure 1: Summary of space bounds for static, incremental, and dynamic filters. New results are highlighted in red.

We remark that this is a quite different approach from the previous lower bound [29], which was achieved by performing a covering-set argument in order to compare the number of filter states on sets of size n to the number of filter states on sets of size $k < n$ for some carefully chosen k.

A second lower bound: the cost of value-dynamic retrieval. Extending our techniques, we are able to obtain a lower bound for a second well-studied problem as well.

Given a set $S \subseteq U$ of n elements from some universe U, and given a function $f : S \to [2^b]$, a *retrieval data structure* for f is any data structure D satisfying

$$D[x] = f(x) \text{ for all } x \in S, \text{ and } \quad D[x] \in [2^b] \quad \text{for all } x \in U \setminus S,$$

where $D[x]$ denotes the value obtained by querying key x.[1]

The retrieval problem is well understood in two settings. In the fully static setting, the optimal expected space is known to be $nb + o(n)$ bits [17, 19, 44]; and in the fully dynamic setting, where both S and f can be updated, the optimal expected space becomes $\Theta(nb + n \log \log(u/n))$ bits [11, 15, 35]. However, the intermediate setting of *value dynamism*, where the set S is static but the function f can be updated, has remained more enigmatic. One can straightforwardly obtain a solution using $nb + n \log e + o(n)$ bits, with high probability in n, using minimal perfect hashing [23, 26, 32], but it is not known whether $nb + o(n)$ bits might be possible.

In Section 4, we prove that any value-dynamic retrieval data structure must use, in expectation, at least $nb + \Omega(n)$ bits of space. Thus, once again there is an information-theoretic $\Theta(n)$ bit "cost of dynamism". Concretely, under the assumption that the universe U has size at least $|U| \geq \text{poly}(n)$, we give a lower bound of $nb + \frac{1}{2} \log(5/4) - o(n) \geq nb + 0.16n - o(n)$ for any $b \geq 1$, and a lower bound of $nb + \frac{1}{2} \log(3/2) - o(n) \geq nb + 0.29n - o(n)$ for any $b = \omega(1)$.

Beating minimal perfect hashing. The fact that $nb + \Omega(n)$ bits are necessary does not necessarily mean that the basic minimal-perfect hashing solution is optimal. And, in fact, in Section 6, we show that for $b = O(1)$ it is not: we give a simple alternative solution that achieves space $nb + n \log e - n \cdot g(b) + o(n)$ bits, with high probability in n, for some positive function g. For $b = O(1)$, this beats the bound achieved by minimal perfect hashing by $\Omega(n)$ bits. For $b = 1$, specifically, we show that $g(b) \geq 0.334$.

Related work on filters and retrieval. Our contributions, along with related work, are summarized in Figures 1 and 2.

The earliest work on filters was by Burton H. Bloom in 1970 [14], who introduced an incremental filter using space approximately $1.44n \log \epsilon^{-1}$ bits. A few years later, in 1978, Carter, Floyd, Gill, Markowsky, and Wegman [10] established a tight bound of $n \log \epsilon^{-1} + O(1)$ bits[2] on the optimal expected space-usage of a static filter, and gave a dynamic filter using $n \log \lceil \epsilon^{-1} \rceil + 2n$ bits. Their basic framework for constructing a dynamic filter, in which they construct a succinct hash table storing a random $\log n + \lceil \log \epsilon^{-1} \rceil$-bit hash of each key, would go on to be used in almost all subsequent work on dynamic filters. Interestingly, when $\epsilon = o(1)$, this approach necessarily uses space $n \log \epsilon^{-1} + \Omega(n) - o(n)$ bits, since when $\epsilon = o(1)$, the information-theoretic lower bound for storing a set of $(1 - o(1))n$ items from a universe of size $n\epsilon^{-1}$ is $\log \binom{n\epsilon^{-1}}{n} \geq n \log \epsilon^{-1} + n \log e - o(n)$ bits. Our lower bound confirms that this type of bound is optimal, although the exact constant in the linear term remains open.

In recent decades, it has been shown how to construct space-efficient filters that are also highly time-efficient. For static filters, it is known how to achieve $O(1)$-time queries while using $n \log \epsilon^{-1} + o(n)$ bits of space [17, 44]. For dynamic filters, so long as $\log \epsilon^{-1} \leq \log n / (\log \log \cdots \log n)$ (for any constant number of logarithms), it is known how to achieve $O(1)$-time operations using $n \log \epsilon^{-1} + O(n)$ bits of space [2, 7, 37]. For smaller ϵ, one can achieve the same space bound, but with $O(\log^* n)$-time insertions/deletions [7].

There has also been a great deal of work on practical filters. Besides the Bloom filter and its many variants [1, 30], other popular designs include the quotient filter [6, 13, 24, 37, 40, 41], the cuckoo filter [12, 22, 31, 38], and static xor-based filters [20, 25]. Filters and their applications have been the subject of many surveys, see e.g., [1, 8, 9, 30, 36, 42, 43, 45, 46].

[1] We remark that retrieval data structures for $f : S \to [r]$ also make sense if r is not a power of two. Our focus on $r = 2^b$ is mostly for convenience.

[2] Technically, their bound considers only power-of-two values of ϵ^{-1}, but their arguments naturally extend to non-powers-of-two as well.

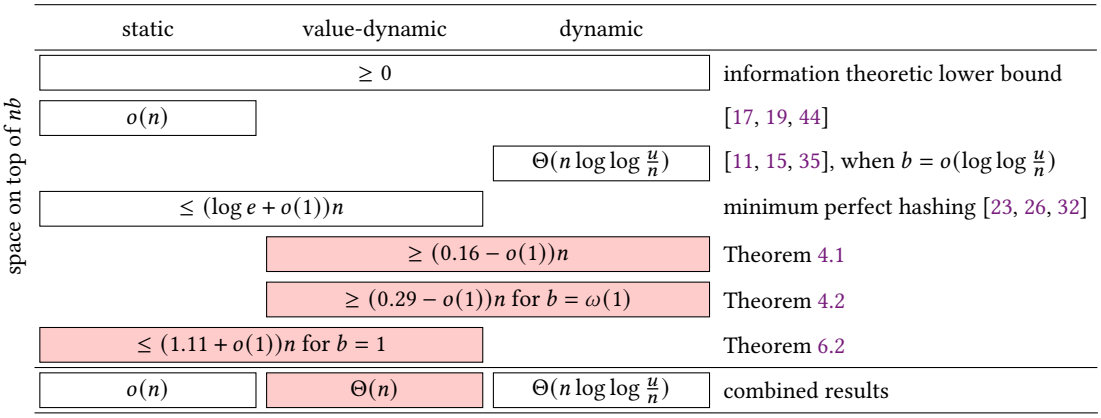

Figure 2: Summary of space bounds for retrieval data structures in the static, value-dynamic, and dynamic settings. New results are highlighted in red. For the (fully) dynamic setting, we restrict to $b = o(\log\log \frac{u}{n})$.

One natural extension of incremental filters is to allow for n to increase arbitrarily over time. Assuming ϵ is an inverse power of two, the optimal space for this version of the problem has been shown to be $(1 + o(1))\left(n \log \epsilon^{-1} + n \log\log n\right) + O(n)$ bits [28, 39]. Other extensions of the problem include adaptive filters [4, 5, 27, 34], which use an external-memory dictionary on S in order to 'fix' false positives so adaptive query sequences get strong guarantees; and learning-augmented filters [33], which use an auxiliary machine learning model to improve performance.

The earliest work on the retrieval problem was in the context of minimal perfect hash functions [23, 26, 32]. Hagerup and Tholey [26], in particular, showed how to efficiently construct constant-time minimal perfect hash functions using $n \log e + o(n)$ bits (with high probability); this, in turn, implies a constant time solution to the (static) retrieval problem using $n \log e + nb + o(n)$ bits.

The observation that static retrieval can be solved with $bn + o(n)$ bits appears to have been made independently by two different sets of authors, first by Dietzfelbinger and Pagh [17], and then by Porat [44] who gave a solution with $O(1)$-time queries. (See, also, follow-up experimental work by Aumüller et al [3].)

Tight bounds of $\Theta(nb + n \log\log(u/n))$ (with high probability) for the fully-dynamic retrieval problem were achieved in a series of three papers by Chazelle, Kilian, Rubinfeld, Tal [11]; Mortensen, Pagh, and Pătraşcu [35]; and Demaine, Meyer auf der Heide, Pagh, and Pătraşcu [15]. Here, again, $O(1)$-time operations are possible [15]. Prior to our work, the only known bounds for value-dynamic retrieval were those achieved by minimal perfect hash functions. It was unknown whether any of the static $nb + o(n)$ constructions could potentially be extended to this setting.

Finally, it is worth briefly remarking on the relationship between filters and retrieval data structures. In the static setting, there is actually a direct reduction between the problems: any retrieval data structure that uses space $f(n, b)$ bits can be directly transformed into a filter that uses $f(n, \log \epsilon^{-1})$ bits. In fact, this has been the primary path taken to construct efficient static filters in both theory [17, 44] and practice [21, 25].

The dynamic versions of the problems studied here (supporting key insertions/deletions for filters; and value updates for retrieval) do not appear to have any formal relationship. They do, however, share an unusual attribute, namely, that they are dynamic data structures that store only incomplete information about the underlying set S of elements being referenced. In both cases, the challenge in establishing a space lower bound is to somehow show that the data structure must unavoidably store $\Omega(n)$ bits of information about S, and that otherwise dynamism becomes impossible.

2 DEFINITIONS AND PRELIMINARIES

Computational model. Not much hinges on how precisely we model randomized data structures. Recall that we neglect running times and are primarily interested in lower bounds. We may generously assume that during all of its operations, the data structure has access to a read-only tape containing an infinite sequence of random bits that does not count towards the space consumption. The data structure might use this tape to implement, for instance, fully random hash functions.[3]

Formal definition of the dynamic filter problem. Let $n \in \mathbb{N}$, $\epsilon \in (0, 1)$, and $U = \{1, 2, \ldots, u\}$ be parameters denoting the capacity, false-positive rate, and universe for a filter, respectively. A dynamic filter is a data structure D representing a dynamic set S. The data structure D supports three operations: $\texttt{query}_D$, $\texttt{insert}_D$, $\texttt{delete}_D$.

The $\texttt{insert}_D(x)$ operation takes as input some $x \in U \setminus S$ and results in a modified data structure D' representing $S' = S \cup \{x\}$. The $\texttt{delete}_D(x)$ operation takes as input some $x \in S$ and results in a modified data structure D' representing $S' = S \setminus \{x\}$. The user is permitted to perform an arbitrary (oblivious) sequence of queries, insertions, and deletions, subject to the constraints that the represented set S has size at most n at all times, and that every insertion/deletion is legal (i.e., insertions are on elements not in S, and deletions are on elements in S). The $\texttt{query}_D$ operation satisfies $\Pr[\texttt{query}_D(x) = \text{true}] = 1$ for $x \in S$ and $\Pr[\texttt{query}_D(x) = \text{false}] \geq 1 - \epsilon$ for $x \in U \setminus S$. Note that in the latter case the probability is over the randomness in D (x is not assumed to be a random element of $U \setminus S$) and that query results within the same sequence

[3] It is worth noting that all of the upper bounds that we discuss can be implemented with $o(n)$ random bits, so long as the universe-size u is not too large, say $2^{o(n)}$.

of operations may be correlated (for instance, repeatedly querying the same element can not be used to reduce the error probability).

Critically, the operations on D must be performed based only on the information stored in D, the value x of the item being inserted/deleted, and any random bits that the data structure relies on. No additional information about S is available. This is what makes the dynamic filter problem interesting: we wish to approximate a dynamically changing set, without actually storing the set.[4]

Formal definition of the value-dynamic retrieval problem. Let $n \in \mathbb{N}$, $b \in \mathbb{N}$, and $U = \{1, 2, \ldots, u\}$. Given a set S and a function $f : S \to [2^b]$, a retrieval data structure D is said to encode f if $D[x] = f(x)$ for all $x \in S$. Note that D is permitted to return any value in $[2^b]$ when answering queries $D[x]$ for $x \notin S$. The retrieval data structure is said to be value-dynamic if it further supports an operation $\text{update}_D(x, k)$ that, given $x \in S$ and $k \in [2^b]$, produces a data structure D' representing $f' : S \to [2^b]$ with $f'(x) = k$ and $f'(x') = f(x')$ for all other $x' \in S$. The $\text{update}_D(x, k)$ operation is only required to exhibit well-defined behavior if $x \in S$.

Once again, the operations on D must be performed based only on the provided arguments and the information stored in D (along with any random bits that the data structure is using), and not based on any additional information about S. The difficulty is that $\text{update}_D(x, k)$ must preserve $D[x']$ for all $x' \in S \setminus \{x\}$ without necessarily having access to the set S itself. This is why value-dynamic retrieval solutions cannot necessarily be constructed directly from static solutions.

Information-theoretic preliminaries. The following considerations will be useful in both lower bounds.

Let U be the universe. For any $U' \subseteq U$, define $\text{density}(U') = |U'|/|U|$. Given $0 \le a \le |U|$ let enc_a and dec_a be functions such that for any set $U' \subseteq U$ and any $A \subseteq U'$ of size a we have that $\mathsf{dec}_a(\mathsf{enc}_a(A, U'), U') = A$ where $\mathsf{enc}_a(A, U')$ is a $\lceil \log \binom{|U'|}{a} \rceil$-bit string. For instance, $\mathsf{enc}_a(A, U')$ could return the rank of A in a canonical ordering of all a-element subsets of U'.

For any $A \subseteq U$, we call $\text{info}(A) = \log \binom{|U|}{|A|}$ to be the *information content* of A. The following lemma quantifies how much of $\text{info}(A)$ is known to us if we know some set U' satisfying $A \subseteq U' \subseteq U$.

Lemma 2.1. *Let $a \le |U|$. For any random sets U' and A satisfying $A \subseteq U' \subseteq U$ and $|A| = a$, we have that*

(i) $|\mathsf{enc}_a(A, U')| \le \text{info}(A) - a \log \frac{1}{\text{density}(U')} + 1$ *and*

(ii) $\mathbb{E}\left[|\mathsf{enc}_a(A, U')|\right] \le \text{info}(A) - a \log \frac{1}{\mathbb{E}[\text{density}(U')]} + 1.$

Proof. Note that (ii) follows from (i) by Jensen's inequality because log is a concave function and because $\text{info}(A) = \binom{|U|}{a}$ is not actually random. To see (i), note for any $a \le u' \le u$ we have

$$\binom{u'}{a} = \binom{u}{a} \cdot \frac{u'}{u} \cdot \frac{u'-1}{u-1} \cdot \ldots \cdot \frac{u'-a+1}{u-a+1} \le \binom{u}{a}\left(\frac{u'}{u}\right)^a.$$

By taking the log and applying to the sizes of A, U and U' we get

$$\log \binom{|U'|}{|A|} \le \log \binom{|U|}{|A|} + a \log \text{density}(U').$$

After taking care of "$\lceil \cdot \rceil$" with a "+1", we obtain the claim. □

We will also use the following standard lemma about the information content of a random variable.

Lemma 2.2. *Suppose that there is a one-way communication protocol in which Alice is able to send a uniformly random element $v \in V$ from some universe V to Bob. Then the expected length of Alice's message to Bob is at least $\log |V|$ bits.*

3 A LOWER BOUND FOR DYNAMIC FILTERS

Let $\mathcal{D}$ be an algorithm for implementing dynamic filters with capacity n, false-positive rate $\epsilon = o(1) \cap 2^{-o(n)}$, and expected space at most $n \log \epsilon^{-1} + nr$ bits. All of the filters in this section will be implemented using $\mathcal{D}$. Our goal is to prove that r must be $\Omega(1)$.

Theorem 3.1. *Let U be a universe of size $\omega(n\epsilon^{-1})$. Consider any dynamic filter with universe U, capacity n, and false-positive rate $\epsilon = o(1) \cap 2^{-o(n)}$. If the expected space usage is bounded by $n \log \epsilon^{-1} + nr + o(1)$ bits for some r, then $r \ge \log \frac{4}{\sqrt{17}-1} - o(1) > 0.35 - o(1)$.*

3.1 Intuition of the Proof

Assume that we have a filter with space requirement $n \log \varepsilon^{-1}$. Such a filter can be used to encode a set $A \subset U$ of size n *indirectly* with optimal space efficiency: First store a filter D_A for A, then specify A as a subset of the positive elements U_A of the filter (i.e., $U_A = \{x \in U \mid \mathsf{query}_{D_A}(x) = \text{true}\}$). The $\log \varepsilon^{-1}$ bits per element that are spent for storing D_A are saved in the second step by Lemma 2.1 because $\mathbb{E}[\text{density}(U_A)]$ is roughly ε. When encoding *two* sets A_1, A_2 the same idea works: First store a filter for $A = A_1 \cup A_2$, then specify both sets as subsets of U_A.

Now assume for contradiction that the considered filter is also dynamic. This will allow us to store sets with even better (and thus impossible) space efficiency.

Our setup involves two disjoint sets A, B of n elements, each partitioned into two sets $A_1 \cup A_2 = A$ and $B_1 \cup B_2 = B$. We consider a filter D_A for A and a filter D_B for B where the latter arises by dynamic operations from the former as follows. Starting from D_A we delete A_1 and insert B_1, which gives us a filter D_{mid} for $A_2 \cup B_1$. Then we delete A_2 and insert B_2 to arrive at D_B. Let U_A, U_{mid} and U_B be the positive elements of the respective filters. We consider two cases.

In the "low churn" case, the sets of positive elements only change a little in each step: At least a 2/3-fraction of the elements in U_A are also in U_{mid} and at least a 2/3-fraction of the elements from U_{mid} are also in U_B. Then at least a 1/3-fraction of the elements from U_A are also in U_B. An optimal way to encode A and B would be to encode D_A and D_B and then encode A and B relative to U_A and U_B. However, we can save space for B: Since A and B are disjoint, at most an ε-fraction of the elements from B are contained in U_A (in expectation). Therefore, we know that most of B is contained in $U_B \setminus U_A$, which is at least a 1/3-fraction smaller than U_B. This allows us to save space—which is impossible.

In the "high churn" case, on the other hand, U_A and U_{mid} are dissimilar, sharing at most a 2/3-fraction of their elements. In this case we will encode A_1, A_2, B_1. A space optimal way to do so would be to encode D_A, encode A_1, A_2 relative to U_A and encode B_1 directly. However, we can do better for A_2. Once D_A, A_1 and B_1 are known,

[4] In many real-world applications of filters [5], the user keeps a full copy of S elsewhere in external memory. This copy is typically accessed by the user to (1) verify positive queries; (2) verify that insertions are valid; and (3) verify that deletions are valid.

a decoder can also compute D_{mid} and hence U_{mid}. Since A_2 is contained in $U_A \cap U_{\text{mid}}$, which is at least a 1/3-fraction smaller than U_A, we can save space when encoding A_2—which is impossible.

This high-level intuition ignores several important considerations. The churn between D_A and D_{mid} need not be the same as the churn between D_{mid} and D_B. We can solve this by performing a many-round experiment in which we construct data structures $D_1, D_2, \ldots$ by repeatedly removing $n/2$ old elements and then inserting $n/2$ new elements. We look at the expected average churn across all the rounds, and then we pick a random round at which to perform the above thought experiment. Another issue is that, in the low-churn case, the fraction of B that is contained in U_A is ϵ in expectation, but not deterministically. We can solve this by adding an if-statement to our construction: only try to save space in the (likely) scenario that $B \cap U_A \le \sqrt{\epsilon}|B|$, and otherwise, just make sure we are using a construction that does not *waste* space. With these modifications to the construction (and some carefulness about details) we will be able to obtain our full lower bound.

3.2 Formal Proof

Let $U = \{1, \ldots, u\}$ be the universe, let $m = \omega(1)$ be even, and let $A_0, A_1, \ldots, A_{m+1} \subseteq U$ be random disjoint sets U consisting of $n/2$ elements each (this uses $u = \omega(n\epsilon^{-1})$). We may assume without loss of generality that $|U| = u$ is polynomial in n, since if u is larger then we can artificially restrict our proof to a subset of U. We can also assume without loss of generality that $r \le O(1)$, since otherwise Theorem 3.1 is trivial.

Whenever we discuss inserting or deleting a *set* of elements into a filter (rather than a single element), it is implicit that the elements are inserted or deleted in order from smallest to largest.

Our proof will make use of the following construction. Build filter D_1 by starting with an empty filter, inserting A_0, and inserting A_1. Then, for each $i \in \{2, \ldots, m+1\}$, construct D_i from D_{i-1} by first deleting A_{i-2} and then inserting A_i. Define $S_i = A_{i-1} \cup A_i$ to be the set of elements that D_i is a filter for. Define $U_i \subseteq U$ to be the set of elements u such that $\texttt{query}(D_i, u) = \text{true}$. Note that $S_i \subseteq U_i$ deterministically, and that, by construction, each U_i satisfies $\mathbb{E}[\text{density}(U_i)] \le (\epsilon u + n)/u = \epsilon + o(\epsilon)$.

Define the *churn rate* of $\mathcal{D}$ to be

$$\kappa = \epsilon^{-1}\mathbb{E}\left[\frac{1}{m}\sum_{i=1}^{m}\text{density}(U_{i+1} \setminus U_i)\right].$$

Note that $\kappa = O(1)$. The churn rate will appear in our proof in slightly varying guises as captured in the following lemma.

Lemma 3.2.

(i) The reverse churn rate $\kappa' := \varepsilon^{-1}\mathbb{E}\left[\frac{1}{m}\sum_{i=1}^{m}\text{density}(U_i \setminus U_{i+1})\right]$ *satisfies* $|\kappa' - \kappa| = o(1)$.

(ii) If t is chosen uniformly at random from $\{1, \ldots, m\}$ then $\mathbb{E}[\text{density}(U_{t+1} \setminus U_t)] = \varepsilon\kappa$.

(iii) If t is chosen uniformly at random from $\{1+a, \ldots, m-b\}$ for $a, b \in O(1)$ then $\mathbb{E}[\text{density}(U_{t+1} \setminus U_t)] \le \varepsilon\kappa \cdot (1 + o(1))$.

(iv) If t is chosen uniformly at random from $\{1+a, \ldots, m-b\}$ for $a, b \in O(1)$ then $\mathbb{E}[\text{density}(U_t \setminus U_{t+1})] \le \varepsilon\kappa' \cdot (1 + o(1))$.

Proof. A telescoping argument and $\mathbb{E}[|U_i|/u] = \varepsilon + o(\varepsilon)$ gives

$$\begin{aligned}
|\kappa - \kappa'| &= \left|\varepsilon^{-1}\mathbb{E}\Big[\frac{1}{m}\sum_{i=1}^{m}\frac{|U_{i+1} \setminus U_i|}{u}\Big] - \varepsilon^{-1}\mathbb{E}\Big[\frac{1}{m}\sum_{i=1}^{m}\frac{|U_i \setminus U_{i+1}|}{u}\Big]\right| \\
&= \frac{1}{\varepsilon m u}\left|\mathbb{E}\Big[\sum_{i=1}^{m}|U_{i+1} \setminus U_i| - |U_i \setminus U_{i+1}|\Big]\right| \\
&= \frac{1}{\varepsilon m u}\left|\mathbb{E}\Big[\sum_{i=1}^{m}|U_{i+1}| - |U_i|\Big]\right| = \frac{1}{\varepsilon m u}\left|\mathbb{E}\Big[|U_{m+1}| - |U_1|\Big]\right| \\
&\le \frac{(\varepsilon + o(\varepsilon))u}{\varepsilon m u} = O(\tfrac{1}{m}) = o(1)
\end{aligned}$$

where the last step uses $m = \omega(1)$. This proves (i).

The law of total probability gives

$$\mathbb{E}[\text{density}(U_{t+1} \setminus U_t)] = \sum_{i=1+a}^{m-b}\tfrac{1}{m-a-b}\mathbb{E}[\text{density}(U_{i+1} \setminus U_i)].$$

For $a = b = 0$ this equals $\varepsilon\kappa$ by definition of κ, proving (ii). To obtain (iii), note that the missing summands work in our favor and the modified normalization factor corresponds to the error term. The same argument applies in the reverse setting, giving us (iv). □

Depending on whether κ is small or large, we can obtain a lower bound on r via one of the following propositions that together imply Theorem 3.1.

Proposition 3.3 (Low-Churn Bound).
$r \ge \log\frac{1}{2\kappa + o(1)} - o(1)$.

Proposition 3.4 (High-Churn Bound).
$r \ge \frac{1}{2}\log\frac{1}{1-\kappa+o(1)} - o(1)$.

Proof of Theorem 3.1. Since the bound from Proposition 3.3 is decreasing in κ and the bound from Proposition 3.4 is increasing in κ they together imply $r \ge \log\frac{1}{2x} - o(1)$, where $x \in (0,1)$ is the solution to $\log\frac{1}{2x} = \frac{1}{2}\log\frac{1}{1-x}$. This gives $x = (\sqrt{17}-1)/8$, implying that $r \ge \log\frac{4}{\sqrt{17}-1} - o(1) \ge 0.35 - o(1)$. This is precisely the claim of Theorem 3.1. □

3.3 Proof of Proposition 3.3

We begin by considering the case where $\mathcal{D}$ has low churn. Consider the following one-way communication protocol. Alice sends Bob a message constructed using Algorithm 1, and Bob decodes the message using Algorithm 2.

We begin by proving a lower bound on the expected number of bits in Alice's message to Bob. We do this by examining the information that Bob is able to reconstruct using the message.

Lemma 3.5. *The expected length of Alice's message to Bob is at least* $\text{info}(S_{t-2}) + \text{info}(S_t) - o(n)$ *bits.*

Proof. Given Alice's message, Bob can use Algorithm 2 to recover the pair (S_{t-2}, S_t). We now consider its distribution.

Let $V_{k,s}$ be the set of all k-tuples of pairwise disjoint sets of size s from U. By definition the sequence $(A_0, \ldots, A_{m+1})$ is a uniformly random element of $V_{m+2,n/2}$. Since t is selected independently of $(A_0, \ldots, A_{m+1})$ the sequence $(A_{t-3}, A_{t-2}, A_{t-1}, A_t)$ is a uniformly random element of $V_{4,n/2}$. This makes $(S_{t-2} = A_{t-3} \cup A_{t-2}, S_t = A_{t-1} \cup A_t)$ a uniformly random element of $V_{2,n}$. By Lemma 2.2

Algorithm 1 ENCODE-LOWCHURNFILTER

sample disjoint $\frac{n}{2}$ element sets $A_0, \ldots, A_{m+1} \subseteq U$
compute D_i, S_i and U_i for all $i \in \{1, \ldots, m+1\}$
sample $t \in \{3, \ldots, m+1\}$ uniformly at random
if $|S_{t-2} \cap U_t| \le \sqrt{\epsilon}|S_{t-2}| \wedge |S_t \cap U_{t-2}| \le \sqrt{\epsilon}|S_t|$ **then**
  let $S_{t-2} \cap U_t \subseteq X \subseteq S_{t-2}$ with $|X| = \sqrt{\epsilon}|S_{t-2}|$
  let $S_t \cap U_{t-2} \subseteq \overline{X} \subseteq S_t$ with $|\overline{X}| = \sqrt{\epsilon}|S_t|$
  $Y \leftarrow S_{t-2} \setminus X$
  $\overline{Y} \leftarrow S_t \setminus \overline{X}$
  $\mathcal{X} \leftarrow \mathsf{enc}_{\sqrt{\epsilon}n}(X, U)$
  $\mathcal{Y} \leftarrow \mathsf{enc}_{(1-\sqrt{\epsilon})n}(Y, U_{t-2} \setminus U_t)$
  $\overline{\mathcal{X}} \leftarrow \mathsf{enc}_{\sqrt{\epsilon}n}(\overline{X}, U)$
  $\overline{\mathcal{Y}} \leftarrow \mathsf{enc}_{(1-\sqrt{\epsilon})n}(\overline{Y}, U_t \setminus U_{t-2})$
  return $(\text{true}, (D_{t-2}, D_t, \mathcal{X}, \mathcal{Y}, \overline{\mathcal{X}}, \overline{\mathcal{Y}}))$
else
  $\mathcal{S}_{t-2} \leftarrow \mathsf{enc}_n(S_{t-2}, U)$
  $\mathcal{S}_t \leftarrow \mathsf{enc}_n(S_t, U)$
  return $(\text{false}, (\mathcal{S}_{t-2}, \mathcal{S}_t))$

Algorithm 2 DECODE-LOWCHURNFILTER$(E, \mathcal{M})$

if $E = \text{true}$ **then**
  $(D_{t-2}, D_t, \mathcal{X}, \mathcal{Y}, \overline{\mathcal{X}}, \overline{\mathcal{Y}}) \leftarrow \mathcal{M}$
  $U_{t-2} \leftarrow \{x \in U \mid \mathsf{query}(x, D_{t-2}) = \text{true}\}$
  $U_t \leftarrow \{x \in U \mid \mathsf{query}(x, D_t) = \text{true}\}$
  $X \leftarrow \mathsf{dec}_{\sqrt{\epsilon}n}(\mathcal{X}, U)$
  $Y \leftarrow \mathsf{dec}_{(1-\sqrt{\epsilon})n}(\mathcal{Y}, U_{t-2} \setminus U_t)$
  $S_{t-2} \leftarrow X \cup Y$
  $\overline{X} \leftarrow \mathsf{dec}_{\sqrt{\epsilon}n}(\overline{\mathcal{X}}, U)$
  $\overline{Y} \leftarrow \mathsf{dec}_{(1-\sqrt{\epsilon})n}(\overline{\mathcal{Y}}, U_t \setminus U_{t-2})$
  $S_t \leftarrow \overline{X} \cup \overline{Y}$
  return (S_{t-2}, S_t)
else
  $(\mathcal{S}_{t-2}, \mathcal{S}_t) \leftarrow \mathcal{M}$
  $S_{t-2} \leftarrow \mathsf{dec}_n(\mathcal{S}_{t-2}, U)$
  $S_t \leftarrow \mathsf{dec}_n(\mathcal{S}_t, U)$
  return (S_{t-2}, S_t)

Alice's message has expected length at least $\log |V_{2,n}|$, which we can bound as follows using $u = \omega(n)$ for the "$\ge$"

$$\log |V_{2,n}| = \log\left(\tbinom{u}{n} \cdot \tbinom{u-n}{n}\right) = \log \tbinom{u}{n} + \log \tbinom{u-n}{n}$$
$$\ge 2 \log \tbinom{u}{n} - o(n) = \text{info}(S_t) + \text{info}(S_{t-2}) - o(n). \quad \square$$

Define E to be the event that $|S_{t-2} \cap U_t| \le \sqrt{\epsilon}|S_{t-2}|$ and $|S_t \cap U_{t-2}| \le \sqrt{\epsilon}|S_t|$. An important fact that we will make use of is that E is likely to occur.

LEMMA 3.6. $\Pr[E] = 1 - O(\sqrt{\epsilon})$.

PROOF. This uses only the randomness in the data structures $(D_i)_{1 \le i \le m+1}$, i.e. we may assume the sequence $(S_i)_{1 \le i \le m+1}$ to be fixed. Since S_{t-2} and S_t are disjoint, we know that each $x \in S_{t-2}$ satisfies $\Pr[\mathsf{query}(x, D_t) = \text{true}] \le \epsilon$. Thus $\mathbb{E}[|S_{t-2} \cap U_t|] \le \epsilon |S_{t-2}|$, so by Markov's inequality,

$$\Pr[|S_{t-2} \cap U_t| \ge \sqrt{\epsilon}|S_{t-2}|] \le O(\sqrt{\epsilon}).$$

The symmetric argument gives $\Pr[|S_t \cap U_{t-2}| \ge \sqrt{\epsilon}|S_t|] = O(\sqrt{\epsilon})$ as well, which completes the proof. $\square$

The following fact will allow us to condense $o(1)$-notation:

LEMMA 3.7. $(1 - o(1)) \log \frac{1}{a+o(1)} \ge \log \frac{1}{a+o(1)}$ *for any* $a > 0$.

PROOF. We can express $(1-o(1)) \log \frac{1}{a+o(1)}$ as $\log \frac{1}{(a+o(1))^{1-o(1)}}$. If $a = o(1)$, then $(a+o(1))^{1-o(1)} \le o(1) \le a+o(1)$. If $a = \Theta(1)$, then $(a+o(1))^{1+o(1)} = a+o(1)$ because the derivative of $x \mapsto (a+o(1))^x$ is bounded by an absolute constant for x in the vicinity of 1. Thus, either way, we have the desired result. $\square$

Our next task is to prove an upper bound on the expected size of Alice's message. The most interesting step bounds the expected sizes of $\mathcal{Y}$ and $\overline{\mathcal{Y}}$, which are only generated if E occurs.

LEMMA 3.8. *We have*
$\mathbb{E}[|\mathcal{Y}| + |\overline{\mathcal{Y}}| \mid E] \le 2\,\text{info}(Y) - 2n \log \frac{1}{2\kappa+o(1)} - 2n \log \epsilon^{-1} + o(n)$.

PROOF. To bound $\mathbb{E}[|\mathcal{Y}| \mid E]$, we can apply Lemma 2.1:

$$\mathbb{E}[|\mathcal{Y}| \mid E] = \mathbb{E}[|\mathsf{enc}_{(1-\sqrt{\epsilon})n}(Y, U_{t-2} \setminus U_t)| \mid E]$$
$$\le \text{info}(Y) - |Y| \log \frac{1}{\mathbb{E}[\text{density}(U_{t-2} \setminus U_t) \mid E]} + O(1).$$

Expanding the expected-value term,

$$\begin{aligned}
\mathbb{E}[\,\text{density}(U_{t-2} \setminus U_t) \mid E] &\le \frac{\mathbb{E}[\text{density}(U_{t-2} \setminus U_t)]}{\Pr[E]} \\
&\le (1 + o(1))\mathbb{E}[\text{density}(U_{t-2} \setminus U_t)] \\
&\le (1 + o(1))\mathbb{E}[\text{density}(U_{t-2} \setminus U_{t-1}) + \text{density}(U_{t-1} \setminus U_t)] \\
&\le (1 + o(1)) \cdot 2\epsilon\kappa' \cdot (1 + o(1)) \qquad \text{(by Lemma 3.2)} \\
&\le 2(\kappa' + o(1)) \cdot \epsilon,
\end{aligned}$$

where the final inequality uses $\kappa' = O(1)$. Thus we have that

$$\begin{aligned}
\mathbb{E}[|\mathcal{Y}| \mid E] &\le \text{info}(Y) - |Y| \log \tfrac{1}{2(\kappa'+o(1))\epsilon} + O(1) \\
&\le \text{info}(Y) - |Y| \log \tfrac{1}{2\kappa'+o(1)} - |Y| \log \epsilon^{-1} + O(1) \\
&\le \text{info}(Y) - (n - \sqrt{\epsilon}n) \log \tfrac{1}{2\kappa'+o(1)} - (n - \sqrt{\epsilon}n) \log \epsilon^{-1} + O(1) \\
&\le \text{info}(Y) - n(1 - \sqrt{\epsilon}) \log \tfrac{1}{2\kappa'+o(1)} - n \log \epsilon^{-1} + o(n) \\
&\le \text{info}(Y) - n \log \tfrac{1}{2\kappa'+o(1)} - n \log \epsilon^{-1} + o(n). \qquad \text{(by Lemma 3.7)}
\end{aligned}$$

The analysis of $\mathbb{E}[|\overline{\mathcal{Y}}| \mid E]$ is similar except that the roles of S_{t-2} and S_t are reversed, the roles of U_{t-2} and U_t are reversed, and the role of κ' is replaced with the role of κ. We therefore have that

$$\mathbb{E}[|\overline{\mathcal{Y}}| \mid E] \le \text{info}(\overline{Y}) - n \log \tfrac{1}{2\kappa+o(1)} - n \log \epsilon^{-1} + o(n).$$

Since $\text{info}(Y) = \text{info}(\overline{Y})$ and $|\kappa - \kappa'| \le o(1)$ (by Lemma 3.2), we can bound

$$\mathbb{E}[|\mathcal{Y}| + |\overline{\mathcal{Y}}| \mid E] \le 2\,\text{info}(Y) - 2n \log \tfrac{1}{2\kappa+o(1)} - 2n \log \epsilon^{-1} + o(n). \quad \square$$

Finally, we prove an upper bound on the size of Alice's message.

Lemma 3.9. *The expected length of Alice's message is at most*

$$\text{info}(S_{t-2}) + \text{info}(S_t) + 2nr - n\log\tfrac{1}{2\kappa+o(1)} + o(n).$$

Proof. The cost for storing the Boolean value and the expected overhead due to storing several variable-length bitstrings in a single bitstring is $o(n)$ bits.[5] The (expected) lengths of the possible components of the tuple are

$$\begin{aligned}
|\mathcal{S}_{t-2}| + |\mathcal{S}_t| &= 2\lceil\text{info}(S_t)\rceil \text{ (note that } |S_{t-2}| = |S_t|)\\
\mathbb{E}[|D_t| \mid E] &\le \frac{\mathbb{E}[D_t]}{\Pr[E]} \le (1+O(\sqrt{\epsilon}))\mathbb{E}[D_t] \le \mathbb{E}[D_t] + o(n)\\
&\le n\log\varepsilon^{-1} + nr + o(n) \quad \text{(ditto for } D_{t-2})\\
|\mathcal{X}| + |\overline{\mathcal{X}}| &\le 2\lceil\text{info}(X)\rceil\\
\mathbb{E}[|\mathcal{Y}| + |\overline{\mathcal{Y}}| \mid E] &\le 2\,\text{info}(Y) - 2n\log\tfrac{1}{2\kappa+o(1)} - 2n\log\epsilon^{-1} + o(n)\\
&\qquad \text{((from Lemma 3.8))}
\end{aligned}$$

Moreover, since $|X| \le O(\sqrt{\epsilon}|S_{t-2}|) = o(|S_{t-2}|)$ and since X and Y are disjoint sets satisfying $X \cup Y = S_{t-2}$, we have that

$$\begin{aligned}
\text{info}(X) + \text{info}(Y) &= \text{info}(S_{t-2}) + o(|S_{t-2}|)\\
&= \text{info}(S_{t-2}) + o(n) = \text{info}(S_t) + o(n).
\end{aligned}$$

Summing the message sizes and applying the insight just given, Alice's message has expected length at most

$$\begin{aligned}
&o(n) + (1-\Pr[E])\cdot(|\mathcal{S}_{t-2}| + |\mathcal{S}_t|)\\
&\quad + \Pr[E]\cdot\mathbb{E}[|D_{t-2}| + |D_t| + |\mathcal{X}| + |\overline{\mathcal{X}}| + |\mathcal{Y}| + |\overline{\mathcal{Y}}| \mid E]\\
&= o(n) + (1-\Pr[E])\cdot 2\lceil\text{info}(S_t)\rceil + \Pr[E]\cdot(2n\log\varepsilon^{-1}\\
&\quad + 2nr + 2\,\text{info}(X) + 2\,\text{info}(Y) - 2n\log\tfrac{1}{2\kappa+o(1)} - 2n\log\epsilon^{-1})\\
&\le o(n) + (1-\Pr[E])\cdot 2\,\text{info}(S_t)\\
&\quad + \Pr[E]\cdot\left(2nr + 2\,\text{info}(S_t) - 2n\log\tfrac{1}{2\kappa+o(1)}\right)\\
&\le o(n) + 2\,\text{info}(S_t) + 2nr - 2n\cdot\Pr[E]\cdot\log\tfrac{1}{2\kappa+o(1)}.
\end{aligned}$$

As $\Pr[E] = 1 - o(1)$, we can apply Lemma 3.7 to bound the above quantity by

$$2\,\text{info}(S_t) + 2nr - 2n\log\tfrac{1}{2\kappa+o(1)} + o(n). \qquad \square$$

Finally, we can prove Proposition 3.3:

Proof of Proposition 3.3. Combined, Lemmas 3.5 and 3.9 give us the inequality

$$\begin{aligned}
&\text{info}(S_{t-2}) + \text{info}(S_t) + 2nr - 2n\log\tfrac{1}{2\kappa+o(1)} + o(n)\\
&\ge \text{info}(S_{t-2}) + \text{info}(S_t) - O(1).
\end{aligned}$$

It follows that $r \ge \log\frac{1}{2\kappa+o(1)} - o(1)$, as desired. $\square$

3.4 Proof of Proposition 3.4

Next, we consider the case where $\mathcal{D}$ has high churn. Consider the following one-way communication protocol. Alice sends Bob a message constructed using Algorithm 3, and Bob decodes the message using Algorithm 4.

[5] This uses that $u = \text{poly}(n\epsilon^{-1}) = 2^{o(n)}$. This means that all relevant bitstrings have sub-exponential expected size, and thus that their *lengths* can be encoded in $o(n)$ expected bits.

Algorithm 3 Encode-HighChurnFilter

sample disjoint $\frac{n}{2}$ element sets $A_0, \ldots, A_{m+1} \subseteq U$
compute D_i, S_i and U_i for all $i \in \{1, \ldots, m+1\}$
sample $t \in \{1, \ldots, m\}$ uniformly at random
$\mathcal{A}_{t-1} \leftarrow \mathsf{enc}_{n/2}(A_{t-1}, U_t)$
$\mathcal{A}_{t+1} \leftarrow \mathsf{enc}_{n/2}(A_{t+1}, U)$
$\mathcal{A}_t \leftarrow \mathsf{enc}_{n/2}(A_t, U_t \cap U_{t+1})$
return $(D_t, \mathcal{A}_{t-1}, \mathcal{A}_t, \mathcal{A}_{t+1})$

Algorithm 4 Decode-HighChurnFilter$(D_t, \mathcal{A}_{t-1}, \mathcal{A}_t, \mathcal{A}_{t+1})$

$U_t \leftarrow \{x \in U \mid \mathsf{query}(x, D_t) = \text{true}\}$
$A_{t-1} \leftarrow \mathsf{dec}_{n/2}(\mathcal{A}_{t-1}, U_t)$
$A_{t+1} \leftarrow \mathsf{dec}_{n/2}(\mathcal{A}_{t+1}, U)$
compute D_{t+1} by deleting A_{t-1} from D_t, and inserting A_{t+1}
$U_{t+1} \leftarrow \{x \in U \mid \mathsf{query}(x, D_{t+1}) = \text{true}\}$
$A_t \leftarrow \mathsf{dec}_{n/2}(\mathcal{A}_t, U_t \cap U_{t+1})$
return (A_{t-1}, A_t, A_{t+1})

As in the previous section, our approach will be to prove lower and upper bounds on the length of Alice's message, where the lower bound comes from examining the information that Bob is able to decode based on the message, and where the upper bound is derived by directly analyzing the components of the message.

Lemma 3.10. *The expected length of Alice's message to Bob is at least* $\text{info}(A_{t-1}) + \text{info}(A_t) + \text{info}(A_{t+1}) - o(n)$ *bits.*

Proof. Given Alice's message, Bob can use Algorithm 4 to recover A_{t-1}, A_t, and A_{t+1}. As in Lemma 3.5, the tuple (A_{t-1}, A_t, A_{t+1}) is uniformly random in the set of tuples (A, B, C) where A, B, C are disjoint size-$n/2$ subsets of U. It follows by Lemma 2.2 that Alice's message has expected length at least

$$\begin{aligned}
&\log\left(\binom{u}{n/2}\cdot\binom{u-n/2}{n/2}\cdot\binom{u-n}{n/2}\right)\\
&= \log\binom{u}{n/2} + \log\binom{u-n/2}{n/2} + \log\binom{u-n}{n/2}\\
&\ge 3\log\binom{u}{n/2} - o(n) \qquad \text{(for } u = \omega(n))\\
&= \text{info}(A_{t-1}) + \text{info}(A_t) + \text{info}(A_{t+1}) - o(n). \qquad \square
\end{aligned}$$

Lemma 3.11. *The expected length of Alice's message is at most*

$$\text{info}(A_{t-1}) + \text{info}(A_t) + \text{info}(A_{t+1}) + nr - \frac{n}{2}\cdot\log\tfrac{1}{1-\kappa+o(1)} + o(n).$$

Proof. Message D_t has expected size at most $n\log\epsilon^{-1} + rn + o(n)$ bits. By Lemma 2.1, we can argue for $\mathcal{A}_{t-1}$ that

$$\begin{aligned}
\mathbb{E}[|\mathcal{A}_{t-1}|] &\le \text{info}(A_{t-1}) - \tfrac{n}{2}\log\tfrac{1}{\mathbb{E}[\text{density}(U_t)]} + O(1)\\
&\le \text{info}(A_{t-1}) - \tfrac{n}{2}\log\tfrac{1}{(1+o(1))\epsilon} + O(1)\\
&\le \text{info}(A_{t-1}) - \tfrac{n}{2}\log\epsilon^{-1} + o(n).
\end{aligned}$$

Message $\mathcal{A}_{t+1}$ has length $\lceil \text{info}(A_{t+1}) \rceil$. And, again applying Lemma 2.1, $\mathcal{A}_t$ has expected length

$$\begin{aligned}
\mathbb{E}[|\mathcal{A}_t|] &\le \text{info}(A_t) - \tfrac{n}{2}\log \tfrac{1}{\mathbb{E}[\text{density}(U_t \cap U_{t+1})]} + O(1) \\
&\le \text{info}(A_t) - \tfrac{n}{2}\log \tfrac{1}{\mathbb{E}[\text{density}(U_t)] - \mathbb{E}[\text{density}(U_t \setminus U_{t+1})]} + O(1) \\
&\le \text{info}(A_t) - \tfrac{n}{2}\log \tfrac{1}{(1+o(1))\epsilon - \kappa\epsilon} + O(1) \qquad \text{(by Lem. 3.2)} \\
&\le \text{info}(A_t) - \tfrac{n}{2}\log \epsilon^{-1} - \tfrac{n}{2}\log \tfrac{1}{1-\kappa+o(1)} + O(1).
\end{aligned}$$

Putting the pieces together, the total expected length of the tuple $(D_t, \mathcal{A}_{t-1}, \mathcal{A}_t, \mathcal{A}_{t+1})$ is at most[6]

$$\begin{aligned}
&n\log\epsilon^{-1} + rn + \text{info}(A_{t-1}) - \tfrac{n}{2}\log\epsilon^{-1} \\
&\quad + \text{info}(A_{t+1}) + \text{info}(A_t) - \tfrac{n}{2}\log\epsilon^{-1} - \tfrac{n}{2}\log\tfrac{1}{1-\kappa+o(1)} + o(n) \\
&= \text{info}(A_{t-1}) + \text{info}(A_{t+1}) + \text{info}(A_t) \\
&\quad + rn - \tfrac{n}{2}\log\tfrac{1}{1-\kappa+o(1)} + o(n). \qquad \square
\end{aligned}$$

Finally, we can prove Proposition 3.4.

Proof of Proposition 3.4. Combining Lemmas 3.10 and 3.11, we get the inequality

$$\begin{aligned}
\text{info}(A_{t-1}) + \text{info}(A_{t+1}) + \text{info}(A_t) + rn - \tfrac{n}{2}\log\tfrac{1}{1-\kappa+o(1)} + o(n) \\
\ge\ \text{info}(A_{t-1}) + \text{info}(A_{t+1}) + \text{info}(A_t) - o(n),
\end{aligned}$$

which implies $r \ge \frac{1}{2}\log\frac{1}{1-\kappa+o(1)} - o(1)$, as desired. $\square$

4 A LOWER BOUND FOR VALUE-DYNAMIC RETRIEVAL

Let $\mathcal{D}$ be an algorithm for implementing a value-dynamic retrieval data structure that stores b-bit values for n elements from a universe U of size $\omega(n)$. Suppose that $\mathcal{D}$ uses expected space at most $nb + nr$ bits for some r. All of the retrieval data structures in this section will be implemented using $\mathcal{D}$. Our goal is to prove that $r = \Omega(1)$.

Theorem 4.1. *Let U be a universe of size $\omega(n)$. Consider a value-dynamic retrieval data structure that stores b-bit values for n elements from U. If the data structure achieves an expected space bound of at most $nb + nr + o(n)$ bits for some r, then $r \ge (\log\frac{5}{4})/2 - o(1) \ge 0.16 - o(1)$.*

We will also prove a stronger version of the theorem for $b = \omega(1)$:

Theorem 4.2. *Let U be a universe of size $\omega(n)$. Consider a value-dynamic retrieval data structure that stores b-bit values for n elements from U. If $b = \omega(1)$, and if the data structure achieves an expected space bound of at most $nb + nr + o(n)$ bits for some r, then $r \ge (\log\frac{3}{2})/2 - o(1) \ge 0.29 - o(1)$.*

Let $U = \{1, \dots, u\}$ denote the universe. We assume without loss of generality that n is even and that $u = |U|$ is polynomial in n, since if u is larger we can restrict ourselves to a subset of it. Let (S_0, S_1) be a uniformly random pair of disjoint $\frac{n}{2}$-element subsets of U. Let $S = S_0 \cup S_1$. Moreover let $(v_x^{(0)}, v_x^{(1)})$ be a uniformly random pair of distinct values from $[2^b]$, chosen independently for each $x \in S$.

[6]Note that, because Alice's message is a tuple with variable-length components, Alice must also encode the lengths of the components; this only adds $o(n)$ additional bits in expectation, so it does not affect our bound.

In general, whenever we discuss updating the values for a set of elements (rather than for a single element), it will be implicit that the elements are updated in order from smallest to largest.

Our proof will make use of the following construction. Initialize an instance D_1 of $\mathcal{D}$ so that D_1 stores $D_1[x] = v_x^{(0)}$ for each $x \in S$. Then, for each $i \in \{2, \dots, n+1\}$, construct D_i from D_{i-1} by updating elements $x \in S_{i \bmod 2}$ to have values $D_i[x] = v_x^{(\lfloor i/2 \rfloor \bmod 2)}$. This means that the function encoded by D_i repeats every four steps.

Once again, we define a notion of *churn*, this time based on the values that change between two data structures D and D'. Define

$$\text{Churn}(D, D') = \{x \in U \mid D[x] \ne D'[x]\},$$

and define the *churn rate* of $\mathcal{D}$ to be

$$\kappa = \mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n} \text{density}(\text{Churn}(D_i, D_{i+1}))\right].$$

Depending on whether κ is small or large, we can obtain a lower bound on r via one of the following propositions.

Proposition 4.3. *We have $r \ge \frac{1}{2}\log\frac{1}{2\kappa+o(1)} - \frac{1}{2}\log\frac{2^b}{2^b-1} - o(1)$.*

Proposition 4.4. *We have $r \ge \frac{1}{2}\log\frac{1}{1-\kappa} - o(1)$.*

Assuming these propositions, we can prove Theorems 4.1 and 4.2 as follows:

Proof of Theorem 4.1. Proposition 4.3 is weakest for $b = 1$ when it gives $r \ge \frac{1}{2}\log\frac{1}{2\kappa+o(1)} - \frac{1}{2} - o(1)$, which is decreasing in κ. Since the bound from 4.4 is increasing in κ both bounds together imply $r \ge \frac{1}{2}\log\frac{1}{1-x} - o(1)$, where $x \in (0,1)$ is the solution to $\frac{1}{2}\log\frac{1}{2x} - \frac{1}{2} = \frac{1}{2}\log\frac{1}{1-x}$. This gives $x = 1/5$ implying that $r \ge \frac{1}{2}\log\frac{5}{4} - o(1)$, as claimed in Theorem 4.1. $\square$

Proof of Theorem 4.2. For $b = \omega(1)$ the bound from Proposition 4.3 becomes $r \ge \frac{1}{2}\log\frac{1}{2\kappa+o(1)} - o(1)$. Again we combine this bound with the bound from Proposition 4.4. This time the relevant equation is $\frac{1}{2}\log\frac{1}{2x} = \frac{1}{2}\log\frac{1}{1-x}$, the solution is $x = 1/3$ and the resulting bound is $r \ge \frac{1}{2}\log\frac{3}{2}$, as claimed in Theorem 4.2. $\square$

4.1 Proof of Proposition 4.3

We begin by considering the case where $\mathcal{D}$ has a low churn rate κ. Consider the following one-way communication protocol. Alice sends Bob a message constructed using Algorithm 5, and Bob decodes the message using Algorithm 6.

Algorithm 5 Encode-LowChurnRetrieval

sample (S_0, S_1), $S = S_0 \cup S_1$ and $(v_x^{(0)}, v_x^{(1)})_{x \in S}$ as discussed
sample $t \in \{3, 5, \dots, n+1\}$ uniformly at random
$\mathcal{S} \leftarrow \mathsf{enc}_n(S, \text{Churn}(D_{t-2}, D_t))$
$j \leftarrow \lfloor t/2 \rfloor \bmod 2$
return $(j, D_{t-2}, D_t, \mathcal{S})$

We first establish a lower bound on the number of bits in Alice's message to Bob. We do this by examining the information that Bob is able to reconstruct using the message. Note that our bounds are now parameterized by both r and κ.

Algorithm 6 DECODE-LOWCHURNRETRIEVAL$(j, D_{t-2}, D_t, \mathcal{S})$

$S \leftarrow \mathsf{dec}_n(\mathcal{S}, \mathrm{Churn}(D_{t-2}, D_t))$
for each $x \in S$ **do**
  $v_x^{(j)} \leftarrow D_t[x]$
  $v_x^{(1-j)} \leftarrow D_{t-2}[x]$
$V^{(0)} \leftarrow \{(x, v_x^{(0)}) \mid x \in S\}$
$V^{(1)} \leftarrow \{(x, v_x^{(1)}) \mid x \in S\}$
return $(S, V^{(0)}, V^{(1)})$

Lemma 4.5. *The expected length of Alice's message to Bob is at least* $\mathrm{info}(S) + nb + n\log(2^b - 1)$ *bits.*

Proof. Given Alice's message, Bob can use Algorithm 6 to recover S, $V_0 = \{(x, v_x^{(0)}) \mid x \in S\}$ and $V_1 = \{(x, v_x^{(1)}) \mid x \in S\}$. Critically, the tuple (S, V_0, V_1) is uniformly random in the set of tuples (A, f, g) where A is a size-n subset of U, and f, g are functions mapping $S \to [2^b]$ satisfying $f(x) \neq g(x)$ for all $x \in S$. It follows by Lemma 2.2 that Alice's message has expected length at least

$$\log\left(\binom{u}{n} \cdot (2^b)^n \cdot (2^b - 1)^n\right) = \mathrm{info}(S) + nb + n\log(2^b - 1). \quad \square$$

Next, we prove an upper bound on the size of Alice's message.

Lemma 4.6. *The expected length of Alice's message is at most*

$$2n(b + r) + \mathrm{info}(S) - n\log\tfrac{1}{2\kappa + o(1)} + o(n).$$

Proof. The quantities j, D_{t-2}, D_t take $2n(b + r) + o(n)$ bits in expectation. Therefore, it suffices to bound the expected size of $\mathcal{S} = \mathsf{enc}_n(S, \mathrm{Churn}(D_{t-2}, D_t))$ by $\mathrm{info}(S) - n\log\frac{1}{2\kappa+o(1)} + o(n)$.

By Lemma 2.1,

$$\mathbb{E}\left[|\mathsf{enc}_n(S, \mathrm{Churn}(D_{t-2}, D_t))|\right] \leq \mathrm{info}(S) - n\log\frac{1}{\mathbb{E}[\mathrm{density}(\mathrm{Churn}(D_{t-2}, D_t))]} + 1.$$

Notice that $\mathrm{density}(\mathrm{Churn}(D_{t-2}, D_{t-1}))$ is at most

$$\mathrm{density}(\mathrm{Churn}(D_{t-2,t-1})) + \mathrm{density}(\mathrm{Churn}(D_{t-1,t})).$$

Since t is uniformly random in $\{3, 5, 7, \ldots, n+1\}$ and by an argument analogous to those in Lemma 3.2, this quantity has expected value $2\kappa + o(1)$. It follows that

$$\mathrm{info}(S) - n\log\frac{1}{\mathbb{E}[\mathrm{density}(\mathrm{Churn}(D_{t-2}, D_t))]} + 1 \leq \mathrm{info}(S) - n\log\tfrac{1}{2\kappa+o(1)} + o(n),$$

completing the proof.[7] $\square$

Finally, we can prove Proposition 4.3:

[7]Note that, because Alice's message is a tuple with variable-length components, Alice must also encode the lengths of the components; this only adds $o(n)$ additional bits in expectation, so it does not affect our bound.

Proof of Proposition 4.3. Combined, Lemmas 4.5 and 4.6 give us the inequality

$$2n(b + r) + \mathrm{info}(S) - n\log\tfrac{1}{2\kappa+o(1)} + o(n) \geq \mathrm{info}(S) + nb + n\log(2^b - 1) = \mathrm{info}(S) + 2nb - n\log\tfrac{2^b}{2^b-1}.$$

It follows that

$$r \geq \tfrac{1}{2}\log\tfrac{1}{2\kappa+o(1)} - \tfrac{1}{2}\log\tfrac{2^b}{2^b-1} - o(1). \quad \square$$

4.2 Proof of Proposition 4.4

Next, we consider the case where $\mathcal{D}$ has high churn. Consider the following one-way communication protocol. Alice sends Bob a message constructed using Algorithm 7, and Bob decodes the message using Algorithm 8.

Algorithm 7 ENCODE-HIGHCHURNRETRIEVAL

sample (S_0, S_1), $S = S_0 \cup S_1$ and $(v_x^{(0)}, v_x^{(1)})_{x \in S}$ as discussed
sample $t \in \{1, \ldots, n\}$ uniformly at random
$j \leftarrow t \bmod 2$
for each $f : S_j \to [2^b]$ **do**
  obtain D_f by updating D_t so that each $x \in S_j$ has value $f(x)$
$U' \leftarrow \bigcap_{f:S_j \to [2^b]} \left(U \setminus \mathrm{Churn}(D_t, D_f)\right)$
$\mathcal{S}_j \leftarrow \mathsf{enc}_{n/2}(S_j, U)$
$\mathcal{S}_{1-j} \leftarrow \mathsf{enc}_{n/2}(S_{1-j}, U')$
return $(j, D_t, \mathcal{S}_j, \mathcal{S}_{1-j})$

Algorithm 8 DECODE-HIGHCHURNRETRIEVAL$(j, D_t, \mathcal{S}_j, \mathcal{S}_{1-j})$

compute $S_j = \mathsf{dec}_{n/2}(\mathcal{S}_j, U)$
for each $f : S_j \to [2^b]$ **do**
  obtain D_f by updating D_t so that each $x \in S_j$ has value $f(x)$
$U' \leftarrow \bigcap_{f:S_j \to [2^b]} (U \setminus \mathrm{Churn}(D_t, D_f))$
$S_{1-j} \leftarrow \mathsf{dec}_{n/2}(\mathcal{S}_{1-j}, U')$
$V \leftarrow \{(x, D_t[x]) \mid x \in S_0 \cup S_1\}$
return (S_0, S_1, V)

Once again, we begin by proving a lower bound on the expected length of Alice's message.

Lemma 4.7. *The expected length of Alice's message to Bob is at least* $\mathrm{info}(S_0) + \mathrm{info}(S_1) + nb - o(n)$ *bits.*

Proof. Given Alice's message, Bob can use Algorithm 8 to recover S_0, S_1, $V = \{(x, D_t[x]) \mid x \in S_0 \cup S_1\}$. Critically, the tuple (S_0, S_1, V) is uniformly random in the set of tuples (A, B, f) where A and B are disjoint size-$n/2$ subsets of U, and f is a function mapping $A \cup B \to [2^b]$. It follows by Lemma 2.2 that Alice's message has expected length at least

$$\log\left(\binom{u}{n/2}\binom{u-n/2}{n/2}(2^b)^n\right) = \log\binom{u}{n/2} + \log\binom{u-n/2}{n/2} + nb$$
$$\geq \log\binom{u}{n/2} + \log\binom{u}{n/2} + nb - o(n) \qquad \text{(since } u = \omega(n)\text{)}$$
$$\geq \mathrm{info}(S_0) + \mathrm{info}(S_1) + nb - o(n). \quad \square$$

Next, we establish an upper bound on the expected length of Alice's message.

Lemma 4.8. *The expected length of Alice's message is at most*

$$\text{info}(S_0) + \text{info}(S_1) + nb + nr - \tfrac{n}{2}\log\tfrac{1}{1-\kappa} + o(n).$$

Proof. The quantities j and D_t together take at most $nb+nr+1$ bits in expectation and $\mathcal{S}_j = \mathsf{enc}_{n/2}(S_j, U)$ takes at most $\text{info}(S_j)+1$ bits. Thus, it suffices to show that $\mathcal{S}_{1-j} = \mathsf{enc}_{n/2}(S_{1-j}, U')$ has expected size $\text{info}(S_{1-j}) - \frac{n}{2}\log\frac{1}{1-\kappa} + O(1)$ bits.[8]

By Lemma 2.1,

$$\mathbb{E}\left[|\mathsf{enc}_{n/2}(S_{1-j}, U')|\right] \le \text{info}(S_{1-j}) - \tfrac{n}{2}\log\frac{1}{\mathbb{E}[\text{density}(U')]} + 1,$$

where U' is as defined in Algorithm 8. So it actually suffices to show that $\mathbb{E}[\text{density}(U')] \le 1-\kappa$. Recall that

$$U' = \bigcap_{f:S_j\to[2^b]} \Big(U \setminus \text{Churn}(D_t, D_f)\Big) \subseteq U \setminus \text{Churn}(D_t, D_{t+1}).$$

By the definition of κ, and since t is random in $\{1,\dots,n\}$, we have $\mathbb{E}[\text{density}(\text{Churn}(D_t, D_{t+1}))] = \kappa$. Therefore, $\mathbb{E}[\text{density}(U')] \le \mathbb{E}[\text{density}(U \setminus \text{Churn}(D_t, D_{t+1}))] = 1-\kappa$ as desired. □

Finally, we can prove Proposition 4.4:

Proof of Proposition 4.4. Combined, Lemmas 4.7 and 4.8 give us the inequality

$$\begin{aligned}&\text{info}(S_0) + \text{info}(S_1) + nb + nr - \tfrac{n}{2}\log\tfrac{1}{1-\kappa} + o(n)\\ &\ge \text{info}(S_0) + \text{info}(S_1) + nb - o(n).\end{aligned}$$

It follows that $r \ge \frac{1}{2}\log\frac{1}{1-\kappa} - o(1)$. □

5 AN UPPER BOUND FOR INCREMENTAL FILTERS

In this section, we show that it is possible to construct a simple incremental filter, with capacity n and false-positive rate $\epsilon = o(1)$, that uses space $n\log\epsilon^{-1} + o(n)$ bits.[9] We do not concern ourselves with the intermediate space usage, i.e., the temporary space used while transforming the filter from one state to the next. However, this can also be straightforwardly reduced to $o(n)$ bits using Dietzfelbinger and Rink's splitting trick [18].

Theorem 5.1. *Supposing $\log\epsilon^{-1} \le n$, there is an incremental filter with capacity n and false-positive rate ϵ, that uses space*

$$n\log\epsilon^{-1} + n\cdot O\left(\frac{(\log\log\epsilon^{-1})^2}{\log\epsilon^{-1}}\right).$$

Proof. Let $k = \log\frac{\log\epsilon^{-1}}{(\log\log\epsilon^{-1})^2} = \Theta(\log\log\epsilon^{-1})$. We wish to achieve false-positive rate ϵ and space $n\log\epsilon^{-1} + O(n/2^k)$ bits. However, by a change of variables, it suffices to achieve false-positive rate $\epsilon\cdot(1+O(2^{-k}))$ and space $n\log\epsilon^{-1} + O(n/2^k)$ bits, since this implies the original result for $\epsilon' = \epsilon\cdot(1+O(2^{-k}))$. We can further assume without loss of generality that $\epsilon = o(1)$.

Select a random hash function $h : U \to [n\epsilon^{-1}2^k]$. Note that, for any set S of up to n items, and any $y \notin S$, the probability that $h(y) \in h(S)$ is at most $\epsilon/2^k$. We call this the *collision rate* of h.

The first $t = (1-2k/\log\epsilon^{-1})n$ insertions. We implement the first $t = (1-2k/\log\epsilon^{-1})n$ insertions $x_1,\dots,x_t$ by storing a space-optimal representation of the set $A = \{h(x_1), h(x_2),\dots,h(x_t)\}$. During this time period, our false positive rate is bounded by the collision rate of h, which is less than ϵ. Our space consumption, on the other hand, is at most

$$\log\binom{n\epsilon^{-1}2^k}{t}$$

bits. As Stirling's approximation bounds $\log\binom{a}{b}$ by $b(\log a - \log b) + O(b)$, the space consumption is at most

$$\begin{aligned}&t(\log n + \log\epsilon^{-1} + k - \log t) + O(t) \le t(\log\epsilon^{-1} + k) + O(n)\\ &\le n\cdot(1-2k/\log\epsilon^{-1})(\log\epsilon^{-1}+k) + O(n) \le n\log\epsilon^{-1}.\end{aligned}$$

The final $n-t$ insertions. After the t-th insertion, we construct a *static* filter F_1 for the set $A = \{h(x_1), h(x_2),\dots,h(x_t)\}$ with false positive rate ϵ. We then implement the remaining insertions with a *dynamic* filter F_2 that has false positive rate $\epsilon/2^k$.

To bound space usage, it suffices to bound the total space consumed by F_1 and F_2. Our static filter F_1 can be implemented with

$$t\log\epsilon^{-1} + O(\log n) = t\log\epsilon^{-1} + n\cdot O\left(\frac{(\log\log\epsilon^{-1})^2}{\log\epsilon^{-1}}\right)$$

bits using the probabilistic method construction from [10].[10][11] Our dynamic filter, on the other hand, can be implemented to use space

$$\begin{aligned}(n-t)\log\tfrac{2^k}{\epsilon} + O(n-t) &\le (n-t)\log\epsilon^{-1} + O\left((n-t)k\right)\\ &\le (n-t)\log\epsilon^{-1} + O\left(\frac{nk^2}{\log\epsilon^{-1}}\right)\\ &\le (n-t)\log\epsilon^{-1} + n\cdot O\left(\frac{(\log\log\epsilon^{-1})^2}{\log\epsilon^{-1}}\right).\end{aligned}$$

Summing, the total space used by F_1 and F_2 is at most

$$n\log\epsilon^{-1} + n\cdot O\left(\frac{(\log\log\epsilon^{-1})^2}{\log\epsilon^{-1}}\right) = n\log\epsilon^{-1} + O(n/2^k).$$

[8] Note that, because Alice's message is a tuple with variable-length components, Alice must also encode the lengths of the components; this only adds $o(n)$ additional bits in expectation, so it does not affect our bound.

[9] Although we do not concern ourselves with the time-efficiency of the filter, one can use known building blocks (e.g., the hash table and dynamic filter in [7] and the static filter in [44]) in order to implement our construction with constant amortized expected cost per operation (and $o(n)$ additional bits of space), so long as ϵ^{-1} is a power of two satisfying $\epsilon \ge \log n/\log^{(O(1))} n$. Here $\log^{(c)} n$ refers to $\log\log\cdots\log n$, with c logarithms.

[10] Here we are obtaining a worst-case bound on the size. Alternatively, one can get an expected bound of $t\log\epsilon^{-1} + O(1)$.

[11] As Carter et al. stop short of fully analyzing their construction, we include a variation of the construction here, where as notation we are storing t elements from a universe U with false-positive rate $\epsilon \ge 1/2^t$: Let $\delta = 2^{-2t}$, set $m = (\epsilon-\delta)^{-t}\log\delta^{-1}$, and let $S_1, S_2,\dots,S_m$ be random subsets of U where each element is included independently with probability $\epsilon-\delta$. If the set of elements stored in our filter is contained in some S_i, then encode the filter as the smallest such i, and return true on exactly the elements in S_i; otherwise, the filter is encoded as the number 0, and the filter returns true for all queries. Either way, the filter is $\log m + O(1) = t\log(\epsilon-\delta)^{-1} + \log\log\delta^{-1} + O(1) = t\log\epsilon^{-1} + O(\log n)$ bits. On the other hand, the false-positive rate is at most $\epsilon - \delta + \Pr[i \text{ doesn't exist}]$, and the probability that i doesn't exist is, in turn, at most $(1-(\epsilon-\delta)^t)^m \le (1-(\epsilon-\delta)^t)^{(\epsilon-\delta)^{-t}\log\delta^{-1}} \le \delta$.

Finally, queries must query both F_1 and F_2 (returning true if either query is positive). The false-positive rate is therefore bounded by the sum of (1) the collision rate of h; (2) the false-positive rate of F_1; and (3) the false-positive rate of F_2. This gives a bound of

$$\epsilon/2^k + \epsilon + \epsilon/2^k = \epsilon \cdot (1 + 1/2^{k-1})$$

on the false positive rate, which completes the proof. □

6 AN UPPER BOUND FOR VALUE-DYNAMIC RETRIEVAL

In this section, we give a simple value-dynamic retrieval solution that achieves $nb + n\log e - n \cdot g(b) + o(n)$ bits of space (with high probability), where $g(b)$ is a positive-valued function. For $b = O(1)$, this improves over the bound achieved by minimal perfect hashing by $\Omega(n)$ bits.

Our construction, which uses a parameter $\delta \in (0, 1)$ that we will decide upon later, is as follows: we will store an array A of $m = (1 + \delta)n$ entries, each of which is b bits, and we will store a hash function h mapping the elements of S to distinct indices in A. We can then encode $f : S \to [2^b]$ by storing $f(x)$ in position $A[h(x)]$ for each $x \in S$.

When $\delta = 0$, this becomes the standard minimal-perfect-hashing solution. What is interesting about this construction is that, in general, there is actually an advantage to setting $\delta > 0$. The key insight is that, even though this causes A to use more space (it now has δn unused entries), it saves space on the number of bits needed to encode h.

We will not concern ourselves with time efficiency in this section. We remark, however, it is straightforward to build a time-efficient version of our construction using standard techniques (see, e.g., [16] for a discussion of how to implement the perfect hash function h, and [7] for how to construct efficient dynamic hash tables and filters that can be used in our construction).

We will make use of the following standard lemma [23, 26, 32]:

Lemma 6.1. *Given a set $S \subseteq U$ of size n, the optimal space needed to encode an injection from S to $[m]$, with high probability in n, is*

$$\log \frac{m^n}{m \cdot (m-1) \cdots (m-n+1)} + O(\log n).$$

Proof. Let $h_1, h_2, \ldots$ be random functions from S to $[m]$. Each h_i has probability

$$\frac{m \cdot (m-1) \cdots (m-n+1)}{m^n}$$

of being injective. It follows that, with high probability in n, there will be some

$$i \le 2^{\log \frac{m^n}{m \cdot (m-1) \cdots (m-n+1)} + O(\log n)}$$

such that h_i is injective. The number i can therefore be used to encode an injection from S to $[m]$. □

By Stirling's approximation, we can simplify the number of bits in Lemma 6.1 to

$$\begin{aligned} &n\log m - \log m! + \log(m-n)! + o(n) \\ &= n\log m - m\log\tfrac{m}{e} + (m-n)\log\tfrac{m-n}{e} + o(n) \\ &= n\log e + (m-n)(\log(m-n) - \log m) + o(n) \\ &= n\log e + (m-n)\log\tfrac{m-n}{m} + o(n). \end{aligned}$$

Setting $m = (1+\delta)n$, the space used by our value-dynamic retrieval construction is

$$\begin{aligned} &n\log e - \delta n \log(\delta^{-1} + 1) + mb + o(n) \\ &= n\left(\log e + b - \delta\log(\delta^{-1}+1) + \delta b + o(1)\right). \end{aligned}$$

For $b = 1$, this is minimized by setting $\delta \approx 0.302$, which results in a space usage of less than

$$nb + n\log e - 0.334n + o(n).$$

For $b > 1$, we can set $\delta = 1/(2^{b+1} - 1)$ to get a space bound of

$$\begin{aligned} &n\left(\log e + b - \delta\log(\delta^{-1}+1) + \delta b + o(1)\right) \\ &= n\left(\log e + b - \delta(b+1) + \delta b + o(1)\right) \\ &= n\left(\log e + b - \delta + o(1)\right) \\ &= n\left(\log e + b - 1/(2^{b+1}-1) + o(1)\right). \end{aligned}$$

Thus, we have the following theorem:

Theorem 6.2. *There is a solution to the value-dynamic retrieval problem using $n\log e + nb - n\cdot g(b) + o(n)$ bits (with high probability in n) where $g(b) = \Omega(1/2^b)$. Moreover, for $b = 1$, we have $g(b) \ge 0.334$.*

REFERENCES

[1] Anes Abdennebi and Kamer Kaya. 2021. A Bloom Filter Survey: Variants for Different Domain Applications. *CoRR* abs/2106.12189 (2021). arXiv:2106.12189

[2] Yuriy Arbitman, Moni Naor, and Gil Segev. 2010. Backyard Cuckoo Hashing: Constant Worst-Case Operations with a Succinct Representation. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS*. IEEE Computer Society, 787–796. https://doi.org/10.1109/FOCS.2010.80

[3] Martin Aumüller, Martin Dietzfelbinger, and Michael Rink. 2009. Experimental Variations of a Theoretically Good Retrieval Data Structure. In *Algorithms - ESA 2009, 17th Annual European Symposium (Lecture Notes in Computer Science, Vol. 5757)*. Springer, 742–751. https://doi.org/10.1007/978-3-642-04128-0_66

[4] Michael A. Bender, Rathish Das, Martin Farach-Colton, Tianchi Mo, David Tench, and Yung Ping Wang. 2021. Mitigating False Positives in Filters: to Adapt or to Cache?. In *2nd Symposium on Algorithmic Principles of Computer Systems, APOCS*. SIAM, 16–24. https://doi.org/10.1137/1.9781611976489.2

[5] Michael A. Bender, Martin Farach-Colton, Mayank Goswami, Rob Johnson, Samuel McCauley, and Shikha Singh. 2018. Bloom Filters, Adaptivity, and the Dictionary Problem. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS*. IEEE Computer Society, 182–193. https://doi.org/10.1109/FOCS.2018.00026

[6] Michael A. Bender, Martin Farach-Colton, Rob Johnson, Russell Kraner, Bradley C. Kuszmaul, Dzejla Medjedovic, Pablo Montes, Pradeep Shetty, Richard P. Spillane, and Erez Zadok. 2012. Don't Thrash: How to Cache Your Hash on Flash. *Proc. VLDB Endow.* 5, 11, 1627–1637. https://doi.org/10.14778/2350229.2350275

[7] Michael A. Bender, Martin Farach-Colton, John Kuszmaul, William Kuszmaul, and Mingmou Liu. 2022. On the optimal time/space tradeoff for hash tables. In *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing*. ACM, 1284–1297. https://doi.org/10.1145/3519935.3519969

[8] James Blustein and Amal El-Maazawi. 2002. Bloom filters. a tutorial, analysis, and survey. *Halifax, NS: Dalhousie University* (2002), 1–31. https://cdn.dal.ca/content/dam/dalhousie/pdf/faculty/computerscience/technical-reports/CS-2002-10.pdf

[9] Andrei Z. Broder and Michael Mitzenmacher. 2003. Survey: Network Applications of Bloom Filters: A Survey. *Internet Math.* 1, 4 (2003), 485–509. https://doi.org/10.1080/15427951.2004.10129096

[10] Larry Carter, Robert W. Floyd, John Gill, George Markowsky, and Mark N. Wegman. 1978. Exact and Approximate Membership Testers. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing*. ACM, 59–65. https://doi.org/10.1145/800133.804332

[11] Bernard Chazelle, Joe Kilian, Ronitt Rubinfeld, and Ayellet Tal. 2004. The Bloomier filter: an efficient data structure for static support lookup tables. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*. SIAM, 30–39. http://dl.acm.org/citation.cfm?id=982792.982797

[12] Hanhua Chen, Liangyi Liao, Hai Jin, and Jie Wu. 2017. The dynamic cuckoo filter. In *25th IEEE International Conference on Network Protocols, ICNP*. IEEE Computer Society, 1–10. https://doi.org/10.1109/ICNP.2017.8117563

[13] John G. Cleary. 1984. Compact Hash Tables Using Bidirectional Linear Probing. *IEEE Trans. Computers* 33, 9 (1984), 828–834. https://doi.org/10.1109/TC.1984.1676499

[14] Erik D. Demaine, Friedhelm Meyer auf der Heide, Rasmus Pagh, and Mihai Puatracscu. 2006. De Dictionariis Dynamicis Pauco Spatio Utentibus (*lat.* On Dynamic Dictionaries Using Little Space). 3887 (2006), 349–361. https://doi.org/10.1007/11682462_34

[15] Erik D Demaine, Friedhelm Meyer auf der Heide, Rasmus Pagh, and Mihai Pătraşcu. 2006. De Dictionariis Dynamicis Pauco Spatio Utentibus: (lat. On Dynamic Dictionaries Using Little Space). In *Latin American Symposium on Theoretical Informatics*. Springer, 349–361.

[16] Martin Dietzfelbinger. 2007. Design Strategies for Minimal Perfect Hash Functions. In *Stochastic Algorithms: Foundations and Applications, 4th International Symposium, SAGA (Lecture Notes in Computer Science, Vol. 4665)*. Springer, 2–17. https://doi.org/10.1007/978-3-540-74871-7_2

[17] Martin Dietzfelbinger and Rasmus Pagh. 2008. Succinct Data Structures for Retrieval and Approximate Membership (Extended Abstract). In *Automata, Languages and Programming, 35th International Colloquium, ICALP (Lecture Notes in Computer Science, Vol. 5125)*. Springer, 385–396. https://doi.org/10.1007/978-3-540-70575-8_32

[18] Martin Dietzfelbinger and Michael Rink. 2009. Applications of a Splitting Trick. In *Automata, Languages and Programming, 36th International Colloquium, ICALP (Lecture Notes in Computer Science, Vol. 5555)*. Springer, 354–365. https://doi.org/10.1007/978-3-642-02927-1_30

[19] Martin Dietzfelbinger and Stefan Walzer. 2019. Constant-Time Retrieval with $O(\log m)$ Extra Bits. In *Proc. 36th STACS*. 24:1–24:16. https://doi.org/10.4230/LIPIcs.STACS.2019.24

[20] Peter C. Dillinger, Lorenz Hübschle-Schneider, Peter Sanders, and Stefan Walzer. 2022. Fast Succinct Retrieval and Approximate Membership Using Ribbon. In *20th International Symposium on Experimental Algorithms, SEA (LIPIcs, Vol. 233)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 4:1–4:20. https://doi.org/10.4230/LIPICS.SEA.2022.4

[21] Peter C. Dillinger and Stefan Walzer. 2021. Ribbon filter: practically smaller than Bloom and Xor. *CoRR* abs/2103.02515 (2021). arXiv:2103.02515

[22] Bin Fan, David G. Andersen, Michael Kaminsky, and Michael Mitzenmacher. 2014. Cuckoo Filter: Practically Better Than Bloom. In *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies, CoNEXT*. ACM, 75–88. https://doi.org/10.1145/2674005.2674994

[23] Michael L Fredman and János Komlós. 1984. On the size of separating systems and families of perfect hash functions. *SIAM Journal on Algebraic Discrete Methods* 5, 1 (1984), 61–68.

[24] Afton Geil, Martin Farach-Colton, and John D. Owens. 2018. Quotient Filters: Approximate Membership Queries on the GPU. In *2018 IEEE International Parallel and Distributed Processing Symposium, IPDPS*. IEEE Computer Society, 451–462. https://doi.org/10.1109/IPDPS.2018.00055

[25] Thomas Mueller Graf and Daniel Lemire. 2020. Xor Filters: Faster and Smaller Than Bloom and Cuckoo Filters. *ACM J. Exp. Algorithmics* 25, Article 1.5 (mar 2020), 16 pages. https://doi.org/10.1145/3376122

[26] Torben Hagerup and Torsten Tholey. 2001. Efficient minimal perfect hashing in nearly minimal space. In *STACS 2001: 18th Annual Symposium on Theoretical Aspects of Computer Science*. Springer, 317–326. https://doi.org/10.1007/3-540-44693-1_28

[27] David J. Lee, Samuel McCauley, Shikha Singh, and Max Stein. 2021. Telescoping Filter: A Practical Adaptive Filter. 204 (2021), 60:1–60:18. https://doi.org/10.4230/LIPICS.ESA.2021.60

[28] Mingmou Liu, Yitong Yin, and Huacheng Yu. 2020. Succinct Filters for Sets of Unknown Sizes. 168 (2020), 79:1–79:19. https://doi.org/10.4230/LIPICS.ICALP.2020.79

[29] Shachar Lovett and Ely Porat. 2013. A Space Lower Bound for Dynamic Approximate Membership Data Structures. *SIAM J. Comput.* 42, 6, 2182–2196. https://doi.org/10.1137/120867044

[30] Lailong Luo, Deke Guo, Richard T. B. Ma, Ori Rottenstreich, and Xueshan Luo. 2019. Optimizing Bloom Filter: Challenges, Solutions, and Comparisons. *IEEE Commun. Surv. Tutorials* 21, 2 (2019), 1912–1949. https://doi.org/10.1109/COMST.2018.2889329

[31] Lailong Luo, Deke Guo, Ori Rottenstreich, Richard T. B. Ma, Xueshan Luo, and Bangbang Ren. 2019. The Consistent Cuckoo Filter. In *2019 IEEE Conference on Computer Communications, INFOCOM*. IEEE, 712–720. https://doi.org/10.1109/INFOCOM.2019.8737454

[32] Kurt Mehlhorn. 1984. Data Structures and Algorithms 3: Multi-dimensional Searching and Computational Geometry. 3 (1984). https://doi.org/10.1007/978-3-642-69900-9

[33] Michael Mitzenmacher. 2018. A Model for Learned Bloom Filters and Optimizing by Sandwiching. (2018), 462–471. https://proceedings.neurips.cc/paper/2018/hash/0f49c89d1e7298bb9930789c8ed59d48-Abstract.html

[34] Michael Mitzenmacher, Salvatore Pontarelli, and Pedro Reviriego. 2020. Adaptive Cuckoo Filters. , 20 pages. https://doi.org/10.1145/3339504

[35] Christian Worm Mortensen, Rasmus Pagh, and Mihai Puatracscu. 2005. On dynamic range reporting in one dimension. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing*. ACM, 104–111. https://doi.org/10.1145/1060590.1060606

[36] Sabuzima Nayak and Ripon Patgiri. 2019. A Review on Role of Bloom Filter on DNA Assembly. *IEEE Access* 7 (2019), 66939–66954. https://doi.org/10.1109/ACCESS.2019.2910180

[37] Anna Pagh, Rasmus Pagh, and S. Srinivasa Rao. 2005. An optimal Bloom filter replacement. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*. SIAM, 823–829. http://dl.acm.org/citation.cfm?id=1070432.1070548

[38] Rasmus Pagh and Flemming Friche Rodler. 2004. Cuckoo hashing. *J. Algorithms* 51, 2 (2004), 122–144. https://doi.org/10.1016/J.JALGOR.2003.12.002

[39] Rasmus Pagh, Gil Segev, and Udi Wieder. 2013. How to Approximate a Set without Knowing Its Size in Advance. In *54th Annual IEEE Symposium on Foundations of Computer Science*. IEEE Computer Society, 80–89. https://doi.org/10.1109/FOCS.2013.17

[40] Prashant Pandey, Michael A. Bender, Rob Johnson, and Rob Patro. 2017. A General-Purpose Counting Filter: Making Every Bit Count. In *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference*. ACM, 775–787. https://doi.org/10.1145/3035918.3035963

[41] Prashant Pandey, Alex Conway, Joe Durie, Michael A. Bender, Martin Farach-Colton, and Rob Johnson. 2021. Vector Quotient Filters: Overcoming the Time/Space Trade-Off in Filter Design. In *SIGMOD '21: International Conference on Management of Data*. ACM, 1386–1399. https://doi.org/10.1145/3448016.3452841

[42] Ripon Patgiri, Sabuzima Nayak, and Samir Kumar Borgohain. 2018. Preventing DDoS using Bloom Filter: A Survey. *EAI Endorsed Trans. Scalable Inf. Syst.* 5, 19 (2018), e3. https://doi.org/10.4108/EAI.19-6-2018.155865

[43] Ripon Patgiri, Sabuzima Nayak, and Samir Kumar Borgohain. 2019. Role of Bloom Filter in Big Data Research: A Survey. *CoRR* abs/1903.06565 (2019). arXiv:1903.06565

[44] Ely Porat. 2009. An Optimal Bloom Filter Replacement Based on Matrix Solving. In *Computer Science - Theory and Applications, Fourth International Computer Science Symposium in Russia, CSR (Lecture Notes in Computer Science, Vol. 5675)*. Springer, 263–273. https://doi.org/10.1007/978-3-642-03351-3_25

[45] Amritpal Singh, Sahil Garg, Ravneet Kaur, Shalini Batra, Neeraj Kumar, and Albert Y. Zomaya. 2020. Probabilistic data structures for big data analytics: A comprehensive review. *Knowl. Based Syst.* 188 (2020). https://doi.org/10.1016/J.KNOSYS.2019.104987

[46] Sasu Tarkoma, Christian Esteve Rothenberg, and Eemil Lagerspetz. 2012. Theory and Practice of Bloom Filters for Distributed Systems. *IEEE Commun. Surv. Tutorials* 14, 1 (2012), 131–155. https://doi.org/10.1109/SURV.2011.031611.00024