# Modeling Average False Positive Rates of Recycling Bloom Filters

Kahlil Dozier
*Department of Computer Science*
*Columbia University*
New York, US
kad2219@columbia.edu

Loqman Salamatian
*Department of Computer Science*
*Columbia University*
New York, US
ls3748@columbia.edu

Dan Rubenstein
*Department of Computer Science*
*Columbia University*
New York, US
danr@cs.columbia.edu

*Abstract*—**Bloom Filters are a space-efficient data structure used for the testing of membership in a set that errs only in the False Positive direction. However, the standard analysis that measures this False Positive rate provides a form of worst case bound that is both overly conservative for the majority of network applications that utilize Bloom Filters, and reduces accuracy by not taking into account the actual state (number of bits set) of the Bloom Filter after each arrival. In this paper, we more accurately characterize the False Positive dynamics of Bloom Filters as they are commonly used in networking applications. In particular, network applications often utilize a Bloom Filter that "recycles": it repeatedly fills, and upon reaching a certain level of saturation, empties and fills again. In this context, it makes more sense to evaluate performance using the average False Positive rate instead of the worst case bound. We show how to efficiently compute the average False Positive rate of recycling Bloom Filter variants via renewal and Markov models. We apply our models to both the standard Bloom Filter and a "two-phase" variant, verify the accuracy of our model with simulations, and find that the previous analysis' worst-case formulation leads to up to a 30% reduction in the efficiency of Bloom Filter when applied in network applications, while two-phase overhead diminishes as the needed False Positive rate is tightened.**

*Index Terms*—**Bloom Filter, False Positives**

## I. Introduction

Bloom Filters [3] are a time-tested, space-efficient data structure for identifying duplicate items within an input sequence, and have been applied in an extremely broad set of computing applications, including packet processing and forwarding/routing in P2P networks [20], cache summarization and cache filtering in CDNs [14], network monitoring [10], data synchronization [13], and even Biometric authentication [16]. The Bloom Filter (BF) is attractive because it errs only in the *False Positive* direction (an arriving input, which we call a *message*, can be incorrectly identified as a repeat of a previous message) and never the *False Negative* direction (a repeat message is never incorrectly classified as new).

There are several variants of "back-of-the-envelope" analyses[1] that give the False Positive rate $f$ of an $n+1$st message after $n$ messages have been inserted, e.g., one common variant yields $f = (1-(1-1/M)^{kn})^k$, where $M$ is the memory (number of bits in the BF) and $k$ is the number of hash functions, with each hash function mapping the message to one of the $M$ bits. This analysis, while useful to demonstrate efficacy of the BF, does not measure False Positives in a way that best serves practical network applications for two reasons:

**Observability**: First, consider the addition of an $n$th element. If the application maintains a count, $b$, of the number of bits set, the next new arrival being classified as a repeat (a False Positive) can be directly computed as $(b/M)^k$; the earlier analysis disregards the ability to inspect current BF state.

**Average rate**: Second, the False Positive rate of a newly arriving message grows as the BF's number of set bits increases. Because of this, applications that merely wish to bound the *average* False Positive rate across all newly arriving messages significantly overestimate this average rate when only considering the message with the worst case bound.

A class of applications that would benefit from a measure of False Positive rate that takes into account both observability and average rate are those that employ what we call a *Recycling Bloom Filter (RBF)* [14], [17], [21]. These applications utilize BF's across sufficiently long timescales, where new messages are continuously arriving, but are interspersed with repeats of previous messages. Since newly arriving messages almost always set bits in the BF, over time the number of bits will become excessive, driving up the False Positive rate, such that some bits must be cleared to keep False Positive rates at a reasonable level. While there are BF variants that do support deletion, their implementation introduces substantial overhead or requires knowledge of the specific elements destined for deletion [2], [8], [11], [18]. Such overhead and complexity is not suited for network applications where message arrivals have a "fading popularity" character, with repeat probability decreasing over time. For applications with this type of message arrival process, one can utilize an active metric that maintains some measure of BF fullness, such as the number of messages inserted, or the number of bits set in the BF. When this metric exceeds a threshold, the BF *recycles*: all bits are cleared, and the filling process begins anew (we say a new BF *cycle* is initiated). This process can continue indefinitely, providing a simple and effective means for maintaining a low

This material is based upon work supported by the National Science Foundation under Grant Nos. CNS-1910138, CNS-2106197, and CNS-2148275. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

[1]Later in the paper, we look at different variants.

maximal (and average) False Positive rate of arrivals.

In this paper, we develop recursive, quickly computable (on today's conventional hardware) analytical models that accurately capture the average False Positive rates of Recycling BFs (RBFs). We show that using an active metric to gauge the recycle to maintain an *average* (as opposed to maximum) False Positive rate can offer a 30% or more increase in amount of messages stored.

While the recycling process does introduce the possibility of False Negatives, the aforementioned decline in popularity of each message over time ensures that well-provisioned RBFs can make False Negative rates negligible. While a detailed investigation of False Negatives is beyond the scope of this paper, a commonly used solution ( [14], [21]) to further drive down False Negative rate is to implement what we refer to as a *two-phase* RBF (see IV-F1 for details) that splits the memory between alternately active and "frozen" BFs. We extend our models to analyze these two-phase variants, and measure the loss in efficiency of minimizing False Positive rate as a function of memory and expected number of messages stored in the BF.

In this paper, we make the following contributions:

- In §II, we provide background on Bloom Filters and their previous (worst case) analysis.
- In §III and §IV, we analytically examine the average False Positive rate of RBF variants using via renewal and Markov Models, and extend our models to cover two-phase variants (§IV-F1).
- We verify our models using discrete-event-driven simulation in §V, and, using our models, contrast performance of the RBF variants as functions of memory size and desired False Positive rates.
- The paper closes by presenting related work §VI and concluding §VII.

Our findings show that the variant that recycles based on the number of bits set, while most complex to model to find the right parameters, is the best performing variant in that it can maintain a given False Positive rate while packing in the largest expected number of messages prior to recycling. In contrast, the variants that recycle based on message counts to achieve an average False Positive rate are easier to model and achieve only a slightly smaller expected number of messages prior to recycling. These average-case variants store around 30% more messages in expectation than the commonly used worst case variant. We also find that implementing a two-phase variant will closely match the one-phase as the False Positive rate to be met decreases. These results are important in that they point to users of BFs (and specifically RBFs) likely being overly conservative for their needs when using previous measures of False Positives.

## II. Background

### A. Bloom Filters

A Bloom Filter [3] can be thought of as an array of $M$ bits indexed $0, \cdots, M-1$, all initially set to 0, that uses $k$ hash functions, each of which maps an arriving *message* to a bit in the array: the bit is then set to 1. *Importantly, these $k$ hash functions' mappings are independent across messages, such that each message's assignment of bits is effectively independent from those assigned to other messages.* This independence property is what makes the BF such a powerful abstraction, as well as amenable to mathematical analysis.

When a message arrives, the bits to which it hashes are checked before being set. If they are all already set, the message is assumed to be a repeat of a prior message. In the rare instance that a new message's $k$ hashes point to bits already set, it is incorrectly classified as a repeat, i.e., a *False Positive*.

*1) Colliding vs. Non-Colliding Hash Functions:* Conventional implementations of hash functions permit several of the $k$ hash function mappings to *collide* and hash to the same bit (i.e., a pseudo-random sampling with replacement). A slightly more sophisticated hashing scheme can ensure the $k$ hashes for a message are distinct from one another (i.e., a pseudo-random sampling without replacement). We respectively refer to these two Bloom Filter variants as *colliding* and *non-colliding*, and we analyze both types. While we show how to incorporate these variants into our models, for values of $M$ and $k$ used in practice (typically $M > 1,000$), the colliding/non-colliding distinction has an unnoticeable impact on False Positive rate (collisions are sufficiently rare in the colliding variant). Note that for either variant of hash functions, the independence of bits assigned to differing messages still holds. Hence, our results are generally presented using the colliding variant.

### B. Traditional False Positive Analysis

As discussed in §I, after having received $n$ unique messages, the (colliding) False Positive likelihood of the $n+1$st message is

$$f_w = [1 - (1 - \frac{1}{M})^{kn}]^k \tag{1}$$

where, for a given $M$ and $n$, $k \approx \frac{M}{n} \ln(2)$ will minimize the corresponding False Positive rate. Wikipedia provides a

TABLE I
Summary of Variables

| Variable | Description |
|---|---|
| | Model input parameters |
| $M$ | Size of RBF (bits) |
| $k$ | Number of hash functions per message |
| $N$ | RBF recycle point (number of new message arrivals) |
| $\sigma$ | RBF recycle point (number of bits set) |
| | Markov Model internal variables |
| $\tau_k^*(i,j)$ | Probability next arrival takes RBF from $i$ to $j$ bits<br>* is one of $\{cn, cr, nn, nr\}$ |
| $\pi_i$ | Steady-state probability RBF contains $i$ bits set |
| | False Positive Variants |
| $f_w$ | False Positive rate using worst-case threshold (1) |
| $f_o$ | False Positive rate using fixed $N$ oracle lower bound (2) |
| $f_a$ | f.p. rate using fixed $N$ bit-setting msgs lower bound (3) |
| $f_\sigma^1$ | False Positive rate using $\sigma$ bound (# bits set $\leq \sigma$),<br>one-phase RBF (18) |
| $f_\sigma^2$ | False Positive rate using $\sigma$ bound, two-phase RBF (26) |

succinct derivation of this simple formula [22], as well as a formula for non-colliding hash functions (in terms of Stirling numbers). From (1), the False Positive rate approaches 1 as the number of messages $n$ increases– an undesirable property for applications with no limit on the number of potential message insertions. We call using such a bound in practice to decide when to recycle the BF a *worst case* bound (hence $f_w$) because the recycling occurs as soon as a particular message's False Positive rate exceeds $f_w$ (while the average over messages inserted will be lower). We refer to the lifetime of a BF between recycling events as a *cycle* of the BF.

## C. Two-Phased Bloom Filter

An RBF, upon recycling, clears all memory with respect to previous arrivals. To mitigate the effects of this memory loss, a BF variant called a *Two-Phase Bloom Filter* has been used in various network applications, e.g., [14], [21].

The two-phase Bloom Filter can be thought of as *two* arrays with $M'$ bits each, indexed $0, ..., M'-1$, along with $k$ independent hash functions that are shared across both arrays. Typically, $M' = \frac{M}{2}$, where $M$ is what the application's memory constraint would be for the standard (one phase) Bloom Filter.

The two-phase variant operates much like the standard recycling variant, with the addition that at any point in time, one Filter array is designated *active* and the other *frozen*. Incoming messages are hashed and added to the active filter. When the active filter is considered to be "full enough" (e.g., the number of bits set or messages stored exceeds some threshold), instead of clearing the bits of the active filter, we clear the bits of the *frozen* filter. The roles of the active and frozen filter are then switched; incoming messages are added to the previously frozen (now active) filter, and the status of the previously active (now frozen) filter remains unaffected. This process repeats indefinitely.

In the two-phase variant, an arriving message is hashed and its bits are compared separately within both the active and frozen filters. A match with either filter is interpreted as the message having been received previously (and having set the corresponding bits in the respective filter). While a two-phase RBF cannot reduce its false positive rate as effectively as its one-phase counterpart using identical total memory, its corresponding False Negative rates will often be significantly lower.

## D. BF Thresholding Approaches

There are two basic approaches to deciding when to recycle a BF. One can count the number of bits $b$ that are set in the BF and recycle when $b$ exceeds a threshold $\sigma$. We call this approach a *$\sigma$-bounded approach.* Alternatively, one can bound the number of bit-setting messages, $N$, that can be admitted into the BF before recycling.[2] We call this approach an *$N$-bounded approach.* We note that one complication with an $N$-bounded approach is that when a message that arrives without setting any bits, a user cannot tell if it is a repeat message or a False Positive. A drawback of formulas for an $N$-bounded approach, such as (1), is that they count new messages regardless of whether or not new bits are set; since in practice new messages that do not set bits are assumed to be repeats, a user will underestimate the number of new messages received when False Positives occur, such that applications of a formula such as (1) to gauge when to recycle will be slightly off.

## E. Averaging Variants

A BF is only of use when the message arrival process consists both of new and repeat messages. We wish to clarify a subtle point about what we are averaging over. When a message first arrives, if it sets additional bits, it is identified as new, and if not, it is classified as a repeat. Consider the $i$th new message that arrives, and let $\eta(i)$ represent the number of times the message arrives within a cycle, and let $F_i$ be an indicator that equals 1 if the $i$th message, upon first arrival, is a False Positive. We can count these $\eta(i)$ arrivals in three different ways.

1) **count-first**: We can assume that only the first arrival can count as a False Positive, making the False Positive rate for the cycle equal to $\sum_i F_i / \sum_i \eta(i)$
2) **count-each:** We can assume that each arrival of a message that initially triggered a False Positive is counted as a False Positive, making the False Positive rate equal to $\sum_i F_i \eta(i) / \sum_i \eta(i)$
3) **count-instance**: We only measure the False Positive rate of first-time arrivals of a message, making the False Positive rate equal to $\sum_i F_i / \sum_i 1$

Which variant to use depends on the needs of the underlying application. For instance, a False Positive in a caching application that uses the BF to determine cache hits for requests would make the wrong assessment for an initial arrival of a request, but the mistake would be detected upon attempting to fetch the item from the cache, so the item would be cached and subsequent requests would not be false positives. Hence, count-first is the appropriate measure. Alternatively, a mechanism that wishes to count unique messages would have a single miscount for each message initially classified incorrectly as having been previously received, hence count-instance is the appropriate measure. Finally, in a setting where the BF is used to drop repeat requests, and where a penalty is paid per dropping of an unsatisfied request, count-each would be the appropriate measure.

We can prove that count-instance is the most stringent (largest) of False Positive rates of the three versions[3]. The proof showing that count-instance results in the largest False Positive rate is omitted here but is included in an extended version [7] of this publication. This observation yields two

[2]Messages that don't set bits are generally not counted because the user must assume they are repeat messages, and since repeat messages never set bits, the False Positive rate of the BF does not increase due to their repeated arrival.

[3]count-first is straightforwardly smaller count-each.

benefits. First, if we compute the false positive rate for count-instance, we have upper bounds on the rates of the count-first and count-each variants.[4] Second, count-instance is not affected by underlying distribution of arriving messages: how often a particular message repeats is of no import, since we only decide its contribution to False Positive rate based on its first arrival, and any subsequent arrivals do not alter the bits in the BF, making its analysis identical regardless of this underlying distribution, whereas the other variants do depend on the underlying distribution.

The remainder of this paper utilizes count-instance, i.e., the False Positive rate is defined with respect to how new arrivals are classified upon their arrival, as it is the most stringent of the three, and provides an upper bound on the other interpretations of false-positive rate.

## III. $N$-bounded average False Positive rate

We compute a lower bound on the $N$-bounded average False Positive rate that utilizes the formula which determines worst-case False Positive rate (i.e., (1) for colliding hash functions). Define $F_i$ to be an indicator r.v. that equals 1 when the $i$th new arrival is a False Positive. Note that $E[F_i] = P(F_i = 1)$ is solved directly by worst-case bound formula (1) directly.

First, let us consider an *oracle* user, who, when inserting a message that sets no bits, can distinguish whether this message is a repeat, or a new message that failed to set bits.[5] This oracle would insert exactly $N$ new messages, and the resulting average False Positive rate can be computed as a simple renewal process, yielding a count-instance *oracle $N$-bounded average* False Positive rate of:

$$f_o = \sum_{i=1}^{N} E[F_i]/N. \tag{2}$$

In contrast, a "real" user would assume that a new message that sets no bits is instead a repeat message, and would not include such a message in their count of $N$ messages. Hence, the oracle would recycle no later than an actual user, thereby ensuring a lower false positive rate. For this real user, we let $i$ iterate over the number of new messages that set bits, and define r.v. $R_i$ to equal the number of new messages that do not set bits and arrive between the $i-1$st and $i$th new messages that do set bits. Then for a single cycle of the RBF, the False Positive rate is $\sum_{i=1}^{N} R_i / \sum_{i=1}^{N}(R_i + 1)$.

Note that for the messages that comprise a given $R_i$, the number of bits set in the BF remains constant (none of the $R_i$ messages are changing the number of bits set), so each such message has an equal likelihood, which we call $p_i$, of not setting additional bits. Note it follows that $P(R_i > j) = {p_i}^{j+1}$, such that $E[R_i] = p_i(1-p_i)^{-1}$. By renewal theory, we have the False Positive rate is thus equal to

$$\frac{\sum_{i=1}^{N} p_i(1-p_i)^{-1}}{\sum_{i=1}^{N}(1 + p_i(1-p_i)^{-1})}.$$

[4]In fact, our analysis of count-instance is itself an upper-bound in an RBF for a subtle reason that we address in the extended version.

[5]This oracle is a hypothetical user because the user can make such a distinction has no need for a BF.

Finally, we show that $p_i$ is lower-bounded by $P(F_i = 1)$. This follows from the fact that the $i$th new message that set bits is actually the $j$th arriving message with $j \geq i$, so $p_i = P(F_j = 1)$ for some $j \geq i$. The lower bound then follows from the fact that $P(F_i = 1)$ is increasing in $i$ (False Positive rates increase as more messages arrive), and that the worst-case analyses' value for $N$ is over all messages, including those that set no bits. Hence, $p_i = P(F_j = 1) \geq P(F_i = 1)$ for $j \geq i$. Since we are underestimating each $p_i$, we are underestimating the likelihood of new arrivals not setting bits, so we are underestimating False Positive rates. Note that for low False Positive rates that are used in practice, $E[R_i]$ will tend to be very small, such that the bound is expected to be tight.

It will be useful for our purposes to define, for $f_i = P(F_j = 1)$ and directly applying (1), a lower bound on the $N$-bounded False Positive rate of

$$f_a = \frac{\sum_{i=1}^{N} f_i(1-f_i)^{-1}}{\sum_{i=1}^{N}(1 + f_i(1-f_i)^{-1})}. \tag{3}$$

We refer to this as the *average-case N* lower bound. Note that these lower-bound the False Positive rate as a function of $N$. In §V, we use these formulae to determine $N$ as a function of False Positive rate: since the formula gives a lower bound on the False Positive rate for a given $N$, it indicates an upper bound on the value of $N$ needed to ensure (lower-bound) a given False Positive rate.

## IV. $\sigma$-bounded average False Positive rate

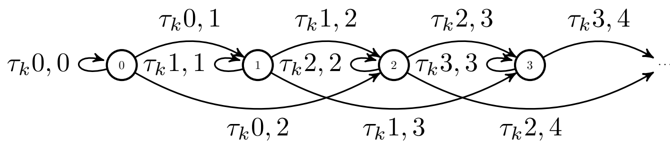

Fig. 1. Partial Markov Chain diagram for Recycling Bloom Filter, showing the "forwards" transition behavior, for $k = 2$ using colliding hash functions. States represent the number of RBF bits set to 1 and $\tau_k i, j$ is the probability of transitioning from state $i$ to state $j$.

As a starting point towards developing a recursive analytical model that accurately captures the average False Positive rates for the $\sigma$-bounded model, we consider a process where messages sequentially arrive at the RBF. Once a message arrives, the $k$ hash functions are applied to it and the corresponding bits are set to 1 (or remain 1 if already set). In a departure from the $N$-bounded method of Bloom Filter recycling, we do not reset the RBF after $n$ unique message insertions. Instead, we have a predetermined bit capacity parameter $\sigma < M$. After the insertion of a message, let $b$ be the number of bits set to 1 in the RBF. We reset the RBF when $b > \sigma$. Our recursive Markov Model will allow us to determine the long-term average False Positive rate across all new arrivals for a given value of $\sigma$ (another departure from the traditional False Positive analysis, which only considers the "worst case" False Positive rate for the last arrival of a RBF reset cycle).

## A. RBF Recursive Markov Model

Our Markov Model for the dynamics of the RBF has $\sigma + 1$ states labeled $0, 1, \cdots, \sigma$. The state labels correspond to the number of bits set to 1 in the RBF during a given cycle. After the arrival of a *unique* message (i.e., not yet seen this cycle), depending on the outcome of the hash operation, the number of set bits in the RBF may or may not change. We represent this event by transition probabilities between the states, labeled $\tau_k i, j$. The number of employed hash functions, $k$, determines which transition probabilities are nonzero– e.g. from state $i$, the "largest" possible state we can transition to is state $\min(i + k, \sigma)$. A partial diagram of our Markov Model, displaying only the "forward" transition behavior, is shown in Fig. 1.

## B. Retaining vs. Non-Retaining RBF

We model the resetting of the RBF with "backwards" transition probabilities from certain states $j$ to earlier states $i < j$. In practice, RBFs can implement recycling behavior in two ways: if message $m$ causes the RBF to reset ($b > \sigma$), after clearing all the bits, we can:

i Retain message $m$ and re-insert it as the first message for the new RBF cycle
ii Forget message $m$ and have the *next* arrival as the first message for the new RBF cycle

We call (i) the *retaining* version of the RBF and (ii) the *nonretaining* version. For both the retaining and non-retaining RBF, nonzero backwards transition probabilities exist at states $\{\sigma - k + 1, \sigma - k + 2, ..., \sigma\}$. For the retaining RBF, the backwards transitions are to states $\{1, 2, ..., k\}$. For the non-retaining RBF, all backwards transitions are to state 0. While we analyze both versions, the non-retaining version is slightly simpler for analysis purposes and is the version we work with unless otherwise mentioned. Fig. 2 shows a partial diagram of our Markov Model displaying the backward transitions.

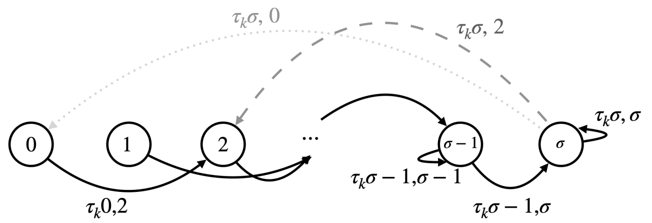

Fig. 2. Partial Markov Chain diagram for Recycling Bloom Filter showing the "backwards" transition behavior, for $k = 2$ (some transition arrows omitted). The light grey arrow depicts the transition for the *retaining version*, while the dark grey arrow is for the *non-retaining version*, non-colliding.

## C. RBF Long-Term Average False Positive Rate

With our Markov Model specified, we can derive an expression for the long-term average False Positive Rate of the RBF, for both colliding and non-colliding BFs, and also derive the backward transitions for colliding/noncolliding and retaining/non-retaining versions (four in total).

*1) Transition Probabilities:* We start by deriving expressions for the forward transition probabilities $\tau_k(i,j)$ of an arriving (new message), where $k$ is the number of hash functions, $i$ is the number of bits set to 1 in the BF prior to insertion of the arriving message, and $j$ is the ending number of bits set to 1 (after the next message is hashed into the filter). We accomplish this for forward transitions from $i$ to $j \geq i$ by recursively defining $\tau_k(i,j)$ in terms of $\tau_{k-1}(i,j)$ and $\tau_{k-1}(i,j-1)$. The explicit recursive equation depends upon whether the BF is being implemented using colliding (vs. non-colliding) hash functions, as well as whether it is retaining (vs. non-retaining). The intuition behind the recursion is that the $k$th hash function will add at most one bit in the BF after the other $k-1$ hashes have been applied. The BF transitions from $i$ to $j$ bits being set if either a) the first $k-1$ hashes transition to $j$ bits and the last hash is to a bit previously set (by earlier messages and/or the preceding $k-1$ hashes of the current message), or b) the first $k-1$ hashes transition to $j-1$ bits set, and the last hash maps to an unset bit. Note that these recursive equations make use of the independence properties of the hash functions applied within Bloom Filters; each message's hashes are independent of those of previous hashes, colliding hashes for a given message are also independent from one another, and non-colliding hashes are chosen via sampling without replacement. The former property ensures that a message arriving to a BF with $b$ bits set can assume these $b$ bits were simply sampled uniformly at random.

We define $\tau_k^{cn}(i,j)$ to be the transition probability when the BF uses colliding hash functions and is non-retaining. In this case when $j \geq i$:

$$\tau_k^{cn}(i,j) = \tau_{k-1}^{cn}(i,j)\frac{j}{M} + \tau_{k-1}^{cn}(i,j-1)\frac{M-j+1}{M} \quad (4)$$

$$\tau_k^{cn}(i,0) = \tau_{k-1}^{cn}(i,0) + \tau_{k-1}^{cn}(i,\sigma)\frac{M-\sigma}{M} \quad (5)$$

And the base case relations:

$$\begin{aligned} \tau_0^{cn}(i,j) &= I_{i=j} & (6)\\ \tau_k^{cn}(0,0) &= 0; k > 0 & (7)\\ \tau_k^{cn}(i,j) &= 0; \quad j > i + k, k > 0 & (8) \end{aligned}$$

where $I_X$ is an indicator that equals 1 when $X$ is true, and is otherwise 0. For colliding + retaining, denote $\tau_k(i,j)$ as $\tau_k^{cr}(i,j)$. $\tau_k^{cr}(i,j)$ has identical form to equation (4) for $j \geq i$. We also have:

$$\tau_k^{cr}(i,0) = 0 \quad (9)$$

$$\tau_k^{cr}(i,j) = \tau_k^{cn}(i,0) \cdot \tau_k^{cn}(0,j); \quad j \leq k < i \quad (10)$$

The base cases for $\tau_0^{cr}(i,j)$ have identical form as (6)-(8). For non-colliding + non-retaining, denote $\tau_k(i,j)$ as $\tau_k^{nn}(i,j)$. We have the recursive relations:

$$\tau_k^{nn}(i,j) = \tau_{k-1}^{nn}(i,j)\frac{j-(k-1)}{M-(k-1)} + \tau_{k-1}^{nn}(i,j-1)\frac{M-j+1}{M-(k-1)}; \quad j \geq i \quad (11)$$

$$\tau_k^{nn}(i,0) = \tau_{k-1}^{nn}(i,0) + \tau_{k-1}^{nn}(i,\sigma)\frac{M-\sigma}{M-(k-1)} \quad (12)$$

$$\tau_k^{nn}(0,k) = 1 \quad (13)$$

The base cases for $\tau_0^{nn}(i,j)$ have identical form as (6)-(8).

For non-colliding + retaining, denote $\tau_k(i,j)$ as $\tau_k^{nr}(i,j)$. $\tau_k^{nr}(i,j)$ has identical form to $\tau_k^{nn}(i,j)$ for $j \geq i$. We also have:

$$\tau_k^{nr}(i,k) = \tau_k^{nn}(i,0); \quad i > k \quad (14)$$

The base cases for $\tau_0^{nr}(i,j)$ have identical form as (6)-(8).

*2) Steady-state Probabilities:* For our Markov Model, let us denote by $\pi_i$ the steady-state probability the RBF has $i$ bits set, where $0 \leq i \leq \sigma$ (this is equivalent to the probability of the Markov chain being in state $i$).

For both the colliding + non-retaining / non-colliding + non-retaining versions of the RBF, the Markov Chain is clearly ergodic. Therefore the stationary distribution $\pi_i$ can be computed by solving for $\pi_i$:

$$\pi_i = \frac{\sum_{j=\max(0,i-k)}^{i-1} \pi_j \tau_k(j,i)}{1-\tau_k(i,i)} \quad (15)$$

We can solve the equations for all $i$ by starting with an arbitrary positive assignment (a "guess") for $\pi_0$; call this $g_0$. Then, we apply (12) to recursively compute $g_i$ for $\pi_i, i > 0$ as functions of $g_0$. Since the sum over all steady state probabilities must be 1, we can renormalize each $\pi_i = \frac{g_i}{\sum_{j=0}^{\sigma} g_j}$.

For the steady-state probabilities of the non-colliding, non-retaining version of the RBF, we can apply an identical method as above, except we have $\pi_i = 0$ for $i < k$. Thus, we make our initial "guess" for the steady-state probability $\pi_k$, but the sets of equations are otherwise unaltered from the previous case.

For the steady-state probabilities of the colliding, retaining version of the RBF, we have an additional complexity. Equation (12) now holds only for $i > k$. For $0 \leq i \leq k$, due to the backwards transitions to these states, the steady-state probabilities are given by:

$$\pi_i = \frac{\sum_{j=1}^{i-1} \pi_j \tau_k(j,i) + \sum_{j=\sigma-k+1}^{\sigma} \pi_j \tau_k(j,i)}{1-\tau_k(i,i)} \quad (16)$$

To solve for the set of $\pi_i$, we can take the set of equations of (12) and (13), along with the constraint $\sum_{j=0}^{\sigma} \pi_i = 1$, and solve by standard linear methods.

*3) Expression for Long-Term Average False Positive Rate:* Let us denote by $\rho(i)$ the long-term average False Positive rate when in state $i$.

For both the colliding and non-colliding versions of the RBF, we have:

$$\rho(i) = \left\{ \begin{array}{ll} \frac{\binom{i}{k}}{\binom{M}{k}} & \text{non-colliding} \\ (\frac{i}{M})^k & \text{colliding} \end{array} \right\} \quad (17)$$

Then the overall long-term average False Positive rate for the RBF is given by (using the desired definition of $\rho(i)$ from (17):

$$f_\sigma^1 = \sum_{i=0}^{\sigma} \pi_i \cdot \rho(i) \quad (18)$$

## *D. Closed-Form expression for $k = 1$*

For the case of colliding, non-retaining, and $k = 1$, we can explicitly solve the above expressions in closed form for the $\pi_i$, yielding a closed form expression for the false long-term average False Positive rate:

$$f_\sigma^{1,k=1} = \sum_{i=0}^{\sigma} \frac{i}{M(M-i)\sum_{j=0}^{\sigma} \frac{1}{M-j}} \quad (19)$$

## *E. Computational Complexity of Calculations*

Our computation proceeds with an outer-most loop iterating over $k$, with $\tau_0(i,j) = 1$ only when $i = j$ and is otherwise 0. For the next value of $k$, we compute $\tau_k(i,j)$ for values of $i$ and $j$ ranging between 0 and $\sigma + k$. Noting that $\tau_k(i,j) = 0$ for $j > i + k$ (hashing to $k$ bins cannot set more than $k$ bits), this involves $(\sigma + k)k$ total computations, making the overall time complexity of $O(Mk^2)$. If only one value of $k$ is being evaluated, it is possible to produce the table of all necessary $\tau_k(i,j)$ holding only $\sigma(k+1)$ values, so it can also be done efficiently in memory.

Once the requisite $\tau_k(i,j)$ have all been computed, computing the steady-state probabilities $\pi_i$ for the *non-retaining* case can be done in $O(kM)$ time, since there are a total of $\sigma + 1$ steady-state probabilities ($\sigma \leq M$) and each of them involve a sum across $k$ terms.

Computing the steady-state probabilities $\pi_i$ for the *retaining* case can be done in $O(k^2M)$ time. There are a total of $\sigma + 1$ steady-state probabilities ($\sigma \leq M$), and $\sigma + 1$ corresponding equations for $\pi_i$. Each of these equations contain $k + 1$ terms; linearly combining a pair of rows is therefore an $O(k)$ operation. Thus, with $O(M)$ row reductions, each with complexity $O(k)$, we can compute $\pi_i$ for all $i \leq k$ (total complexity $O(k^2M)$. We can subsequently compute $\pi_i$ for $i > k$ directly from (16); this takes $O(kM)$ time, so the row reduction step is the complexity bottleneck.

Computing the long-term average RBF False Positive rate as in (18) can be done in $O(kM)$ time for colliding and $O(M)$ time for non-colliding (using the fact that $\binom{i+1}{k} = \binom{i}{k}\frac{i+1}{i+1-k}$ to compute the $\rho(i)$ terms in constant time).

Since the long-term average RBF False Positive rate is computed by successive and separate (but dependent) computations each upper-bounded by $O(k^2M)$, the overall computational complexity is $O(k^2M)$.

## *F. Expected messages within a $\sigma$-bounded RBF*

In each cycle of a $\sigma$-bounded RBF, the number of messages that are hashed into the BF prior to recycling will vary, depending on the number of hash collisions between (and for colliding, also within) messages. We conclude our analysis of the one-phase BF by showing how the expected number of messages can be computed.

Consider a particular sample path (i.e., cycle) of the RBF, and let $b(i)$ indicate the number of bits set after the arrival of the $i$th new message. Let $N(b(i))$ equal the number of additional messages sent after the $i$th message to trigger a recycle (i.e., cross the sigma threshold). Note that when $b(i) \geq$

$\sigma$, we have already crossed the threshold such that $N(b(i)) = 0$. Otherwise, when $b(i) < \sigma$, more messages must be received, such that

$$N(b(i)) = 1 + N(b(i+1)) = 1 + \sum_{j=0}^{k} X_j(i) N(b(i) + j)$$

where $X_j(i)$ is an indicator that equals 1 only when $j$ additional bits get set from the arrival of the $i$th message.

Noting the above equation holds irrespective of the value of $i$, we can simply replace $b(i)$ with $b$ and just write $N(b)$, which we replace with $N_b$ to get the result $N_b = 1 + \sum_{j=0}^{k} X_j(i) N_{b+j}$. We can similarly substitute in the r.v. $X_{b,b+j}$ for $X_j(i)$ which indicates that this $i+1$st message takes the BF from having $b$ bits set to $b+j$. This can be solved to permit a reverse recursion:

$$N_b = \qquad 0, \qquad b \geq \sigma \tag{20}$$

$$N_b = \frac{1 + \sum_{j=1}^{k} X(b, b+j) N_{b+j}}{1 - X(b,b)}, b < \sigma \tag{21}$$

Noting independence of the $X_{b,b+j}$ from $N_b$, and that $E[X_{b,b+j}] = \tau_k(b, b+j)$, we can rephrase the above as an expectation:

$$E[N_b] = \qquad 0, \qquad b \geq \sigma \tag{22}$$

$$E[N_b] = \frac{1 + \sum_{j=1}^{k} \tau_k(b, b+j) E[N_{b+j}]}{1 - \tau_k(b,b)}, b < \sigma \tag{23}$$

and our solution is simply $E[N_0]$.

*1) Two-Phase RBF Long-term average False Positive Rate:* We can use the results from the standard RBF to also derive an expression for the False Positive rate of the two-phase RBF variant. Let $F_i$ denote the steady-state probability of the *frozen* filter having $i$ bits. Note $F_i$ is only nonzero for $i$ between $\sigma - k + 1$ and $\sigma$. To compute $F_i$, we first define an un-conditional (non-normalized) value of $\hat{F}_i$ to then compute a normalized (conditional) version of $F_i$ in terms of $\hat{F}_i$:

$$\hat{F}_i = \pi_i \sum_{j=\sigma+1}^{i+k} \tau_k(i,j) \tag{24}$$

$$F_i = \frac{\hat{F}_i}{\sum_{j=\sigma-k+1}^{\sigma} \hat{F}_i} \tag{25}$$

The long-term average False Positive rate for the two-phase RBF is then given by the expression (using the desired definition of $\rho(i)$ from (17):

$$f_\sigma^2 = 1 - (1 - \sum_{j=0}^{\sigma} \pi_j \rho(j))(1 - \sum_{i=\sigma-k+1}^{\sigma} F_i \rho(i)) \tag{26}$$

## V. Results

To evaluate our models and draw conclusions about the performance of the RBF under varying parameters, we proceed in three steps:

1) We verify the accuracy of our models through discrete event-driven simulations implemented in Python. The RBF data structures were implemented using standard Python libraries. The uniform hash function generation and random message arrival process leveraged the Python `random` library. Code can be found at: [1]
2) With the accuracy of the models verified, we turn to the question of the maximum message capacity achievable by a RBF while staying below a given False Positive rate. We consider four models of a one-phase RBF: the $\sigma$-bounded model (18), the "oracle" $N$-bounded False Positive rate (2), the lower-bound on the $N$-bounded average False Positive rate, $f_a$ (3), and the traditional worst-case bound $f_w$ (1). The $\sigma$-bounded model maintains the highest number of messages, followed by $f_o$, $f_a$, and $f_w$ the lowest. This sequence implies that $N$-bounding variants are overly conservative estimates. Therefore, if one aims to size a RBF to achieve the best performance, the $\sigma$-bounded variant should be utilized.
3) Finally, for the $\sigma$-bounded model, we investigate the trade-off between using a one-phase and two-phase RBF variant, in terms of the maximum memory capacity achievable by each variant while staying below a given False Positive rate.

We note that the results for the combinations of colliding/non-colliding and retaining/non-retaining variants are nearly identical, and thus for brevity all graphs represent the colliding/non-retaining combination.

### A. Verification of Model Accuracy

*1) One and Two-Phase Sigma Bound:* We verify the accuracy of $f_\sigma^1$ and $f_\sigma^2$ through discrete event-driven simulations. For all simulations, we have a fixed RBF size of $M = 1000$ bits. The two-phase RBF splits this memory evenly between each filter. We run multiple simulation epochs of $100,000$ message arrivals, recording the average False Positive rate at the end of each epoch. Message arrivals are drawn uniformly from a pre-generated distribution of 1000 messages. Given a collection of average False Positive rates, we can plot sample means and confidence intervals using standard statistical methods [19]. We compare these results to the values predicted by $f_\sigma^1$ and $f_\sigma^2$. Fig. 3 depicts the results. We simulated for a total of 7 epochs for $f_\sigma^1$ and 10 epochs for $f_\sigma^2$. The 99% confidence interval is indicated on the plot by the shaded area around the simulation curves. For all data points, the value of $f_\sigma^1$ and $f_\sigma^2$ lies within the confidence interval. This is strong evidence in favor of the accuracy of $f_\sigma^1$ and $f_\sigma^2$ in modelling False Positive rates for RBFs.

*2) Lower bounds on False Positive rate:* We turn next to verifying the accuracy of $f_a$ and $f_o$. For this, we ran 14 simulation epochs of $1,000,000$ messages each. Fig. 4 shows the simulation results plotted against $f_a$ and $f_o$, with the 99% confidence interval shown. For all data points, both $f_o$ and $f_a$ are below the sample mean, and fall either within or below the confidence interval. This demonstrates strong evidence in

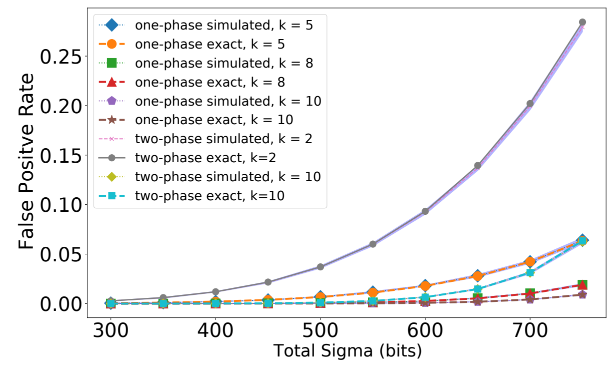

Fig. 3. Verifying our model accuracy through simulation.

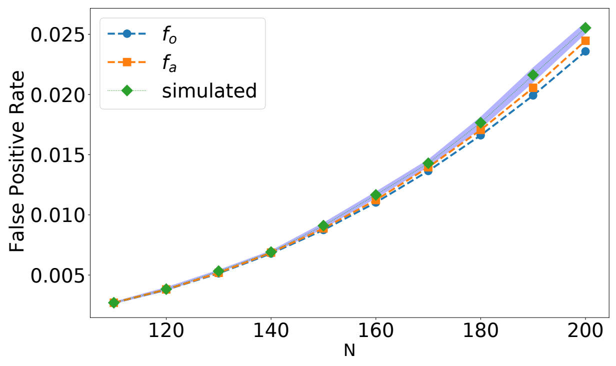

Fig. 4. Verifying $f_o, f_a$ are lower bounds for average False Positive rate through simulation.

favor of the accuracy of the lower bounds $f_o, f_a$. Observe that $f_a$ is a "tighter" lower lower bound than $f_o$.

## B. Expected Message Capacity Comparison

A natural question to ask when optimizing RBF parameters is the expected maximum number of messages one can store while staying under an acceptable average False Positive rate–we refer to this as the *expected message capacity*. For the one-phase $\sigma$-bounded model (False Positive rate given by $f^1_\sigma$), recall this is $E[N_0]$ derived in §IV-F. We can also compute an expected message capacity using $f_a$ or $f_\omega$. Fig. 5 compares the expected message capacity between the three models. For each model, we fix an average False Positive rate of .01. For each value of $M$, we find the maximum number of messages subject to the False Positive constraint. We plot this number normalized to $f^1_\sigma$, as this is the model that yields the highest message capacity. $f_a$ is seen to be a good approximation, a direct consequence of the tightness of the lower bounds discussed in §III. Observe that $f_\omega$ can lead to an overly conservative estimate of message capacity in this context; for the given parameters its maximum message capacity is consistently reduced by more than 30%.

Fig. 6. shows similar results, this time varying False Positive rates on the x-axis. Once again, the worst-case $N$ model sees

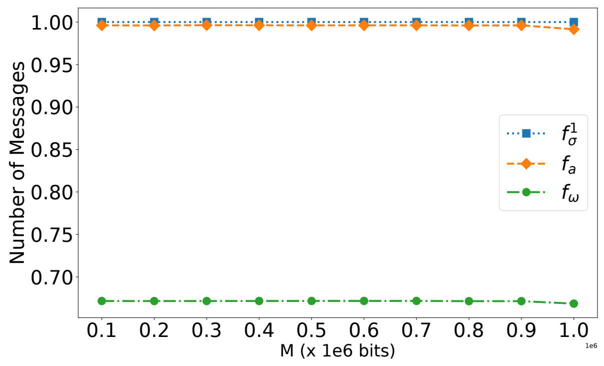

Fig. 5. Expected message capacity of $f_w, f_a$ normalized by $f^1_\sigma$ message capacity, for different values of $M$.

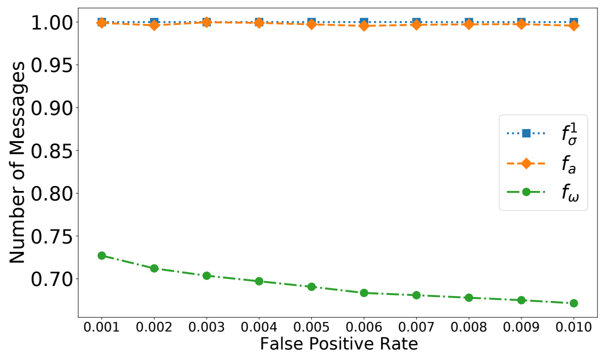

Fig. 6. Expected message capacity of $f_w, f_a$ normalized by $f^1_\sigma$ message capacity, for different False Positive rates.

a consistent underestimation of maximum message capacity of around 30%.

We can easily conclude that, for applications that are more concerned with average False Positive rates than worst-case ones, $f_\omega$ not an ideal metric, as compared to $f^1_\sigma$ or $f_a$.

## C. Tradeoff between One and Two-phased RBF

In Fig. 7, we compare the False Positive rates of one-phase RBFs ($f^1_\sigma$) versus their two-phase counterparts ($f^2_\sigma$). We have a given filter memory $M$ (which in the case of the two-phase RBF is split evenly between both filters) and False Positive rate. We then find the value of $k$ which allows us the maximum number of expected messages for these constraints. In both cases, we observe the result that the one-phase RBF appears to outperform the two-phased filter when it comes to squeezing out extra filter bit capacity. One might be moved to ponder what is the point of the added complexity of a two-phased filter, if it apparently is "less efficient" in terms of average False Positive rate? The answer becomes apparent when we consider the idea of *False Negatives* as we previously mentioned. While a detailed investigation of False Negatives is beyond the scope of this paper, the intuition is that the extra bit capacity seemingly afforded by the one-phase filter comes with an increased probability of False Negative rates, which are mitigated by the two-phase filter.

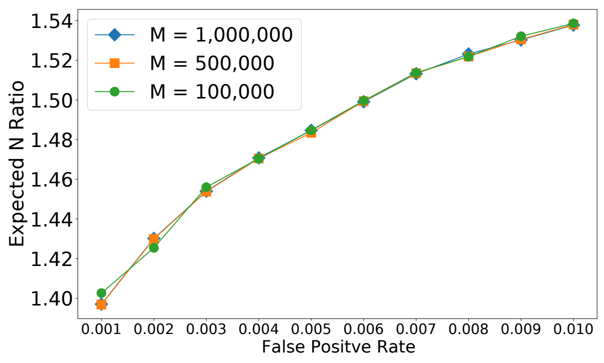

Fig. 7. Expected message capacity ratio of a one-phase RBF to a two-phase, for different values of $M$ and False Positive rates.

## VI. Related Work

Luo *et al.* provides a general summary of Bloom Filters and their variants, along with their applications to problems in computing [12]. To the best of our knowledge, all previous analyses of Bloom Filters approach the question of False Positive rates from the perspective of a "worst case" bound. Equation (1) is widely used in most applications to compute the False Positive rate according to this metric. Two prior works have shown this equation is slightly inaccurate, and in fact a lower bound for the true "worst case" False Positive rate [4], [5].

There have been other BF variants proposed that consider the need to periodically remove items from the filter. The Counting Bloom Filter (CBF) [9] and similar variants support deletion of individual elements. However, the CBF and its variants all come with increased space and algorithmic overhead to support deletion. Among similar lines, variants such as the Deletable Bloom Filter [18], the Ternary Bloom Filter [11] and the Quotient Filter [2] support element deletion under restricted circumstances, also at the cost of increased space and algorithmic complexity.

The idea of periodically resetting a targeted subset of BF bits is first proposed in Donnet *et al.* [6]; while this method is shown to reduce the overall False Positive rate, it also comes at the cost of increased algorithmic complexity. More recently, Cuckoo Filters have been used as an extension of Bloom filters. Cuckoo filters use Cuckoo hashing to optimize space utilization. These filters notably offer the ability to delete elements post-insertion without practical overhead [8]. However, this requires knowledge of the elements that necessitate deletion, a condition not commonly met in many networking applications. In addition, unlike RBFs, Cuckoo filters possess a finite unique message capacity. This inherently limits the scope of its usage under conditions where an unknown and continuous influx of elements is anticipated, which are common in many networking applications.

Many applications employ the RBF strategy as a low-cost alternative to deal with the need to remove elements periodically. Akamai deploys a two-phase RBF approach to guarantee that any content that ends up being cached in their edge servers has been requested at least twice within a designated time frame [14]. Bloom Filter Routing (BFR) has also been introduced in Information-Centric Networks (ICN) to simplify the process of content discovery across the networks [15] and in wireless networking where Trindade *et al.* have designed *Time Aware Bloom Filter* to only remove specific bits that have not been "hit" during a predefined time window [21].

## VII. Conclusion

Bloom Filters and their variants are a space-efficient data structure employed widely in all manner of computing applications. Their space efficiency comes with the tradeoff of potential False Positives, and as such much work has been dedicated towards detailed False Positive analysis. Yet all this work approaches the question of False Positives from the perspective of a "worst-case" bound. This bound is overly conservative for the majority of applications that use Bloom Filters, as it does not take into account the actual state of the Bloom Filter after each arrival. In fact, applications that use Bloom Filters often have to periodically "recycle" the filter once an allowable number of messages threshold has been exceeded. In cases such as these, different metrics such as the long-term average False Positive rates across new arrivals may be of more interest than a worst-case bound.

We derive a method to efficiently compute the long-term average False Positive rate of a Bloom Filter that periodically "recycles" itself (termed a Recycling Bloom Filter). We use renewal and Markov models to respectively derive lower bound exact expressions for the long-term average False Positive rates, and apply our model to the standard Recycling Bloom Filter, and a "two-phase" variant that is popular in network applications. We demonstrate that the previous worst-case analysis of False Positives can lead to a reduction in the efficiency of RBFs in certain scenarios.

## References

[1] https://github.com/kadzier/Recycling-Bloom-Filters-False-Positives.
[2] M. A. Bender, M. Farach-Colton, R. Johnson, R. Kraner, B. C. Kuszmaul, D. Medjedovic, P. Montes, P. Shetty, R. P. Spillane, and E. Zadok. Don't thrash: How to cache your hash on flash. *Proc. VLDB Endow.*, 5(11):1627–1637, jul 2012.
[3] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.
[4] P. Bose, H. Guo, E. Kranakis, A. Maheshwari, P. Morin, J. Morrison, M. Smid, and Y. Tang. On the false-positive rate of bloom filters. *Information Processing Letters*, 108(4):210–213, 2008.
[5] K. Christensen, A. Roginsky, and M. Jimeno. A new analysis of the false positive rate of a bloom filter. *Information Processing Letters*, 110(21):944–949, 2010.
[6] B. Donnet, B. Baynat, and T. Friedman. Retouched bloom filters: allowing networked applications to trade off selected false positives against false negatives. In *Proceedings of the 2006 ACM CoNEXT conference*, pages 1–12, 2006.
[7] K. Dozier, L. Salamatian, and D. Rubenstein. Technical report: Modeling average false positive rates of recycling bloom filters, 2024.

[8] B. Fan, D. G. Andersen, M. Kaminsky, and M. D. Mitzenmacher. Cuckoo filter: Practically better than bloom. In *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, pages 75–88, 2014.

[9] L. Fan, P. Cao, J. Almeida, and A. Broder. Summary cache: a scalable wide-area web cache sharing protocol. *IEEE/ACM Transactions on Networking*, 8(3):281–293, 2000.

[10] Y. Li, R. Miao, C. Kim, and M. Yu. {FlowRadar}: A better {NetFlow} for data centers. In *13th USENIX symposium on networked systems design and implementation (NSDI 16)*, pages 311–324, 2016.

[11] H. Lim, J. Lee, H. Byun, and C. Yim. Ternary bloom filter replacing counting bloom filter. *IEEE Communications Letters*, 21(2):278–281, 2016.

[12] L. Luo, D. Guo, R. T. Ma, O. Rottenstreich, and X. Luo. Optimizing bloom filter: Challenges, solutions, and comparisons. *IEEE Communications Surveys & Tutorials*, 21(2):1912–1949, 2018.

[13] L. Luo, D. Guo, J. Wu, O. Rottenstreich, Q. He, Y. Qin, and X. Luo. Efficient multiset synchronization. *IEEE/ACM Transactions on Networking*, 25(2):1190–1205, 2017.

[14] B. M. Maggs and R. K. Sitaraman. Algorithmic nuggets in content delivery. *ACM SIGCOMM Computer Communication Review*, 45(3):52–66, 2015.

[15] A. Marandi, T. Braun, K. Salamatian, and N. Thomos. Bfr: A bloom filter-based routing approach for information-centric networks. In *2017 IFIP Networking Conference (IFIP Networking) and Workshops*, pages 1–9, 2017.

[16] E. Martiri, M. Gomez-Barrero, B. Yang, and C. Busch. Biometric template protection based on bloom filters and honey templates. *Iet Biometrics*, 6(1):19–26, 2017.

[17] L. McHale, J. Casey, P. V. Gratz, and A. Sprintson. Stochastic pre-classification for sdn data plane matching. In *2014 IEEE 22nd International Conference on Network Protocols*, pages 596–602. IEEE, 2014.

[18] C. E. Rothenberg, C. A. Macapuna, F. L. Verdi, and M. F. Magalhaes. The deletable bloom filter: a new member of the bloom family. *IEEE Communications Letters*, 14(6):557–559, 2010.

[19] K. Sidik and J. N. Jonkman. A simple confidence interval for meta-analysis. *Statistics in medicine*, 21(21):3153–3159, 2002.

[20] S. Tarkoma, C. E. Rothenberg, and E. Lagerspetz. Theory and practice of bloom filters for distributed systems. *IEEE Communications Surveys & Tutorials*, 14(1):131–155, 2012.

[21] J. Trindade and T. Vazão. Hran-a scalable routing protocol for multihop wireless networks using bloom filters. In *International Conference on Wired/Wireless Internet Communications*, pages 434–445. Springer, 2011.

[22] Various. Wikipedia's bloom filter description. https://en.wikipedia.org/wiki/Bloom_filter#Bloom1970, 2023.