Design of a Novel Statistics Counter Architecture with Optimal Space and Time Efficiency *

Qi (George) Zhao Jun (Jim) Xu
College of Computing
Georgia Institute of Technology

Zhen Liu
IBM T.J Watson Research Center

ABSTRACT

The problem of how to efficiently maintain a large number (say millions) of statistics counters that need to be incremented at very high speed has received considerable research attention recently. This problem arises in a variety of router management algorithms and data streaming algorithms, where a large array of counters is used to track various network statistics and to implement various counting sketches respectively. While fitting these counters entirely in SRAM meets the access speed requirement, a large amount of SRAM may be needed with a typical counter size of 32 or 64 bits, and hence the high cost. Solutions proposed in recent works have used hybrid architectures where small counters in SRAM are incremented at high speed, and occasionally written back ("flushed") to larger counters in DRAM. Previous solutions have used complex schedulers with tree-like or heap data structures to pick which counters in SRAM are about to overflow, and flush them to the corresponding DRAM counters.

In this work, we present a novel hybrid SRAM/DRAM counter architecture that consumes much less SRAM and has a much simpler design of the scheduler than previous approaches. We show, in fact, that our design is optimal in the sense that for a given speed difference between SRAM and DRAM, our design uses the theoretically minimum number of bits per counter in SRAM. Our design uses a small write-back buffer (in SRAM) that stores indices of the overflowed counters (to be flushed to DRAM) and an extremely simple randomized algorithm to statistically guarantee that SRAM counters do not overflow in bursts large enough to fill up the write-back buffer even in the worst case. The statistical guarantee of the algorithm is proven using a combination of worst case analysis for characterizing the worst case counter increment sequence and a new tail bound theorem for bounding the probability of filling up the write-back buffer. Experiments with real Internet traffic traces show that the buffer size required in practice is significantly smaller than needed in the worst case.

*The work of Qi Zhao and Jun Xu is supported in part by NSF grant NETS-NBD 0519745 and NSF CAREER Award ANI 0238315.

SIGMetrics/Performance'06, June 26–30, 2006, Saint Malo, France.

Categories and Subject Descriptors

C.2.3 [COMPUTER-COMMUNICATION NETWORKS]: Network Operations - Network Monitoring, Network Management

General Terms

Algorithms, Design, Theory, Performance

Keywords

Statistics Counter, Router, Data Streaming

1. INTRODUCTION

How to efficiently store and maintain a large number (say millions) of statistics counters that need to be incremented at very high speed has been recognized as an important research problem [13, 11]. In this problem, tens of millions of increments need to be performed every second, each of which can happen to any of these counters[1]. In addition, the size of each counter needs to be as large as 64 bits [11], since the value of some of these counters can become very high. The above speed requirement precludes the storage and maintenance of these counters in slower yet inexpensive memory such as DRAM. While fitting these counters entirely in fast yet more expensive SRAM meets the speed requirement, a large amount of SRAM may be needed with such large counter sizes, and hence the high cost. The common research issue in both the prior work [13, 11] and this work, is whether we can design a counter architecture that satisfies the above speed and size requirements, yet use much less SRAM than storing the counters entirely in SRAM.

1.1 Motivation

The need to maintain a large array of counters arises in various router algorithms and data streaming algorithms, where a large array of counters is used to track various network statistics and to implement various counting sketches respectively.

As described in the prior work [13, 11], maintaining a large number of statistics counters is essential for a range of statistical accounting operations (e.g., performing SNMP link counts) at Internet packet switches and routers (e.g., IP routers, ATM switches, and Ethernet switches). Such operations are needed in network performance monitoring, management, intrusion detection, tracing, traffic engineering, etc. [13, 11]. As discussed in [11], the number of statistics counters can be as large as millions when routers would like to support traffic accounting based on various traffic filters such as source and/or destination IP prefixes, traffic type, AS

[1]Lack of locality in counter accesses prevents the traditional caching approach from being effective.

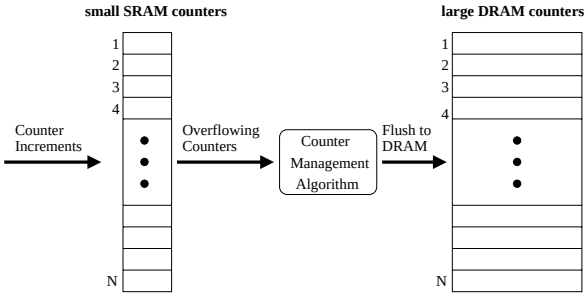


Figure 1: Hybrid SRAM/DRAM counter architecture

(Autonomous System) pairs, and their combinations. In a typical application scenario, each incoming packet triggers an increment (typically by 1) to one or more of the counters depending on the traffic filter(s) that the packet matches. For high speed links such as OC-768 (40 Gbps), such an increment needs to be performed within several nanoseconds to keep up with the packet arrival rates.

The need to maintain a large number of high speed counters is also motivated by the recent advances of data streaming algorithms such as [4, 2, 5, 6, 14, 15, 7]. As defined in [10, 8], data streaming is concerned with processing a long stream of data items in one pass using a small working memory in order to approximately estimate certain statistics of the stream. A data streaming algorithm typically organizes its working memory into a synopsis data structure called sketch, which is specialized to capture as much information pertinent to the statistics it intends to estimate, as possible. While different sketches are proposed for estimating various statistics about the data stream, they often consist of one (e.g., [5, 6, 15, 7]) to several (e.g.,[4, 3, 14]) arrays of counters and have a common online operation called "hash and increment". In these algorithms, an incoming data item (e.g., a packet) is fed to a hash function and the hash result is treated as the index into an array of counters. The corresponding counter is then incremented (often by 1). In network and even some database applications, data items can arrive at a very high speed, and in many sketches, each data item can trigger an increment to multiple counters[2]. In addition, several of these algorithms need to use up to millions of counters in various application scenarios. Therefore, techniques to significantly reduce the amount of SRAM needed to maintain these counters will benefit all these data streaming algorithms in considerably reducing their implementation costs.

1.2 Hybrid SRAM/DRAM counter architectures

A hybrid SRAM/DRAM counter architecture is proposed in the seminal work of [13] as a more SRAM-efficient way of maintaining a large counter architecture, and this architecture is further improved in the followup work of [11]. Figure 1 shows the generic architecture for maintaining a large number of counters using a hybrid SRAM/DRAM design. The baseline idea of this architecture is to store some lower order bits (e.g., 9 bits) of each counter in SRAM, and the full-size counter (e.g., 64 bits) in DRAM. The increments are made only to these SRAM counters, and when the value of a SRAM counter becomes close to overflow, it will be scheduled to be *flushed* to the corresponding DRAM counter. The "flush" operation here is defined as adding the value of the SRAM counter to the corresponding DRAM counter and resetting the SRAM counter to 0. The key research challenge in this architecture is the design of a counter management algorithm (CMA) that flushes the right set of SRAM counters to DRAM at the right time.

Both prior works achieve impressive reductions in SRAM usage through this hybrid SRAM/DRAM approach. Since the result in [11] is strictly better than that of [13], here we only compare our work with [11], deferring a careful comparison with both schemes to later sections. As we will show, with a SRAM/DRAM speed difference of 30 times, the architecture in [11] reduces the SRAM usage from 64 bits per counter to only 11 bits per counter. Among these 11 bits, each SRAM counter takes 9 bits and the other 2 bits per counter are used by the CMA. However, 9 bits are far from the minimum number of SRAM bits per counter that is theoretically possible, which we will show to be 5 in this case (with SRAM/DRAM speed difference of 30). Moreover, the CMA control logic in [11] is fairly complicated to implement, which requires the maintenance of a tree-like data structure in pipelined hardware and consumes 2 bits per counter, although it is simpler and more efficient than that in [13], where a heap has to be maintained in hardware and the control logic consumes about 20 bits per counter.

1.3 Our approach and contributions

In this paper, we present a novel hybrid SRAM/DRAM statistics counter architecture that is provably optimal in terms of SRAM consumption yet has extremely simple control logic. With the same assumption as above, our scheme only requires $5+\epsilon$ bits per counter, where each SRAM counter consumes 5 bits and our CMA consumes ϵ bits per counter. Here ϵ is typically a small number (e.g., 0.01). Note that reducing the SRAM counter size from 9 bits per counter to 5 bits per counter is to a certain extent $2^{9-5} = 16$ times harder, since overflows from an SRAM counter happen 16 times faster with the smaller overflow threshold (2^5 as compared to 2^9), requiring the "flushing" mechanism to operate 16 times more efficiently. What is more remarkable is that this improvement in efficiency is achieved with a CMA that is extremely simple and consumes only a small fraction of bits per counter, as compared to 2 bits per counter in [11].

There is a tiny price to pay for the above significant improvements in terms of both operational efficiency and implementation complexity. Our CMA uses a randomized algorithm, which with an extremely small yet nonzero probability, may lose some increments to the counters, while both previous approaches are deterministic, guaranteeing no such loss. However, in practice there is no need to worry about this probability since it can be made so small that even if a router operates continuously for billions of years (say from Big Bang to now), the probability that a single loss of increment happens is less than 1 over a billion. Note that router software/hardware failures and other unexpected or catastrophic events (e.g., power outage or earthquake) that may disable a router happen with a probability many orders of magnitude higher.

In short, our solution works as follows. Each logical counter is represented by a 5 bit counter in SRAM, and a larger 64 bit counter in DRAM. Increments to a logical counter happen to its 5 bit SRAM counter until it reaches the overflow value 32, at which point the value of this SRAM counter (i.e., 32) needs to be flushed to the corresponding DRAM counter. Since updates to DRAM counters take much longer than to SRAM counters, several SRAM counters may overflow during the time it takes to update just one DRAM counter. Our solution is to install a small SRAM FIFO buffer between the SRAM counters and the DRAM counters to temporarily hold the "flush requests" that need to be made to the DRAM in the future. However, this solution as stated so far will

[2]They often reside in different logical arrays, but these arrays are often stored in a single SRAM chip due to various design constraints.

not work well in the worst case, where a large number of counter overflows may happen during a short period of time so that the SRAM FIFO buffer has to be very large (thereby negating the advantage of our scheme). Our solution to this problem lies in a simple randomization technique with which we can statistically guarantee that SRAM counters do not overflow in bursts large enough to fill up that small SRAM FIFO buffer, even in the worst case. This randomization technique is the key innovation of this work.

Through a rigorous analytical modeling of the proposed solution, we show that a small SRAM buffer (several hundred slots) is large enough to ensure that the probability for it to be filled up by flush requests is vanishingly small, when there are millions of counters. Here each slot holds a counter index to be flushed, which is usually shorter than 4 bytes. This translates into the aforementioned ϵ bits per counter when the cost of hundreds of buffer slots are amortized over millions of counters.

Our analytical modeling of the proposed solution is a combination of worst case analysis and a novel tail bound technique. The purpose of the worst case analysis is to characterize the worst-case workload (counter increment sequence) to the counter array, which is modeled as the best workload generation strategy of an adversary (explained later) that has complete knowledge of our algorithm but not the random values generated by our algorithm during execution. This analysis allows us to establish a tight bound on the variance of the number of counter overflows during a measurement interval. The next step is to bound the probability that the FIFO buffer is overwhelmed by the overflowed counters using some well-known tail bound theorems. However, traditional tail bound techniques such as Chernoff bound will not allow us to take advantage of the variance bound obtained through the above worst case analysis. We develop a novel tail bound theorem, which takes full advantage of the variance bound, to establish probability bounds which is able to improve the Chernoff bound significantly.

Simulations of this architecture using real-world Internet traces demonstrate that the actual buffer size needed is much smaller than the bounds derived through our analysis. This is not surprising given that the analytical bounds work for the worst case while typical traffic patterns observed in a real-world network tend to be fairly "benign" (not in the security sense). One could design a more specific solution targeting observed patterns in real world traffic. We advise against this extension, however, because the savings on SRAM will be very small (in terms of percentage) and it will limit the applicability of our scheme to certain traffic arrival patterns that may not hold true in general.

Note that our hybrid SRAM/DRAM counter architecture is designed for "increment by 1" and will not work well for other increment sizes. While this is sufficient for counting the number of packets, in which each incoming packet triggers the increment of one or more counters by 1, it will not work for counting the number of bytes, in which each incoming packet needs to increment one or more counters by hundreds or thousands (e.g., performing SNMP link counts). A simple extension of the architecture to accommodate other increment sizes is proposed in [11]. Its idea is to statistically quantize an increment of size S to a Bernoulli random variable with mean S/M, where M is the maximum packet size. After this quantization, a counter only needs to be incremented by 1, but the tradeoff is that some estimation error will be introduced. This extension can be readily used in combination with our architecture to handle increments of other sizes. We refer readers to [11] for details of this extension. In the rest of the paper we focus on the problem of "increment by 1".

The rest of the paper is organized as follows. Section 2 summarizes the previous work on this topic and compares them with our architecture briefly. In Section 3 we describe our architecture in detail. In Section 4 we provide a rigorous analysis on the performance of our architecture in the worst case. In Section 5 we present the numerical results of this analysis and simulate the performance of our architecture using real world Internet traces. Section 6 concludes the paper.

2. BACKGROUND AND RELATED WORK

Our scheme and the two existing solutions all adopt the SRAM/DRAM hybrid architecture as described before. These schemes differ on the design of counter management algorithm (CMA). Recall that the CMA tracks which counters are close to overflow and schedules these counters for flushing. The CMA should guarantee that no flush requests are overlooked due to unexpected and unprocessed overflows, with certainty or very high probability.

The CMA used in [13], called LCF (Largest Counter First), adopts a heap-based priority queue to select the counter that is closest to overflow to be flushed to the DRAM. This greedy algorithm clearly makes the best possible selection decision (without future knowledge). However, maintaining a heap in hardware incurs high implementation complexity and a large amount of SRAM, which is about twice the size needed to store the SRAM counters.

A more efficient CMA, called $LR(b)$ (Largest Recent with threshold b), is introduced in [11]. $LR(b)$ avoids the expensive operation of maintaining a priority queue, by only keeping track of counters that are larger than a threshold b using an bitmap. A tree structure is imposed on the bitmap (resulting in a hierarchical bitmap) to allow for a fast retrieval of the "next counter to be flushed". This retrieval operation has complexity $O(\log N)$, where N is the number of counters in the array, but using a large base such as 8 makes the complexity essentially a small constant. The hardware control logic in $LR(b)$ is simpler than LCF, and uses much less SRAM.

The control logic in our scheme, which is to send a flush request to the DRAM counter when the SRAM counter overflows, is clearly much simpler than both previous approaches. Our SRAM requirement is also much smaller than in LCF and $LR(b)$. For example, when the speed difference between SRAM and DRAM is 30 and there are $N = 10^6$ counters, LCF, $LR(b)$ and our scheme need 29 Mb, 11 Mb, and $5 + \epsilon$ (ϵ around 0.01) Mb respectively. Moreover, our scheme uses the smallest counter size that is theoretically possible, and is therefore optimal. Existing solutions do have one advantage over our algorithm, that is, they are deterministic in nature and guarantee no data loss while our algorithm may lose one or more updates in a long measurement interval with small (e.g., 10^{-14}) yet nonzero probability. However, this probability is not a big issue as explained before.

In determining the speed ratio between DRAM and SRAM, we use the following parameters from the commodity DRAM and SRAM chips. DRAM access time varies between 50ns and 100ns, with 60ns being the most typical. SRAM access time varies between 2ns and 5ns. If we fix DRAM access time at 60ns, the speed ratio between DRAM and SRAM varies between 1/30 to 1/12. Note that the ratio of 1/20 used in LR(b) scheme [11] is inside this range. In examples throughout this paper, we will use either 1/30 or 1/12.

3. OUR SCHEME

The basic intuition behind our scheme is as follows. Like in prior approaches [13, 11], we maintain l-bit counters in SRAM and full-size counters in DRAM. The SRAM counters will handle increments at very high speed, and once an SRAM counter reaches value 2^l (overflow), its value needs to be flushed to its corresponding DRAM counter. The CMA in our scheme is extremely simple.

We maintain a small FIFO queue in SRAM that holds the indices of the SRAM counters that have overflowed but have not yet been flushed to DRAM. Note that the access speed of DRAM (departure rate from the queue) should be faster than the average rate of counter overflow (arrival rate to the queue), since otherwise, the queue will fill up no matter how large it is. This implies that if the ratio of DRAM access speed to that of SRAM is μ (< 1), the SRAM counter size l should be larger than $\log_2 \frac{1}{\mu}$ bits, which we refer to earlier as the theoretically minimum SRAM counter size.

A queue is still needed even when $l > \log_2 \frac{1}{\mu}$, however, since the instantaneous counter overflow rate could be much faster than the DRAM access speed *in the worst case*. From queuing theory, we know that the queue size is small when the arrival process is "smooth". A key innovation of our scheme is to guarantee that the arrival process (the arrival of the counter overflows) is fairly smooth even in the worst case through a simple randomization scheme. This ensures that a small queue can guarantee no loss of the indices (to be flushed to DRAM) with overwhelming probability.

The pseudo-code of the algorithm is shown in Algorithm 1. The counter arrays A and B correspond to l-bit SRAM counters and full-size DRAM counters respectively. Let N be the number of counters in A as well as in B. An increment to an arbitrary counter i is handled in lines 1 through 5. The SRAM counter $A[i]$ will first be incremented, and if this increment causes it to overflow (line 3), its content needs to be flushed to DRAM. In this case, its index i is placed into the FIFO queue Q (line 4) and $A[i]$ is reset to 0 (line 5). The actual flush operation is shown in lines 6 through 9. When DRAM finishes the previous write (flushing), a controller will fetch a counter index i from the head of the queue if it is not empty, and increment the DRAM counter $B[i]$ by 2^l (line 9). In terms of implementation, we only require that the array A and the queue Q reside in different SRAM modules, as we will explain later in this section.

Algorithm 1: The pseudo-code of the algorithm

```
1  UpdateSRAM(i)
2     A[i] := A[i] + 1;
3     if (A[i] == 2^l) /* overflow happens */
4        Q.enqueue(i);
5        A[i] := 0;

6  FlushToDRAM()
7     while (Q is not empty)
8        i := Q.dequeue();
9        B[i] := B[i] + 2^l;

10 Initialize()
11    for i:=1 to N
12       A[i] := uniform(0, 2^l - 1);
13       B[i] := -A[i];
```

The specification of our scheme so far is not complete, as we have not assigned initial values to $A[i]$ and $B[i]$, $i = 1, 2, ..., N$. These initial assignments turn out to be the most critical part of our scheme. While simply setting $A[i]$ and $B[i]$, $i = 1, 2, ..., N$, to 0 at the beginning of a measurement (counting) interval is a standard practice, it will not work well in our scheme for the following reason. In the worst case, an adversary (explained below) could choose the sequence of the indices of the counters to be incremented to be $1, 2, ..., N, 1, 2, ..., N,$ ... (i.e., the repetition of the subsequence "$1, 2, 3, ..., N$" over and over). At the end of the $(2^l - 1)_{th}$ repetition, the values of $A[i]$, $i = 1, 2, ..., N$, will all become $2^l - 1$.

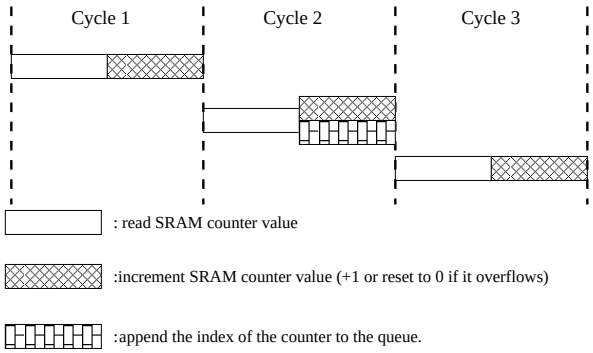


Figure 2: Timing diagram for our algorithm

Then during the next repetition, $A[i]$, $i = 1, 2, ..., N$ will overflow one by one after each increment, resulting in a "burst arrival" of size $O(N)$ to the queue. The queue has to be made very large (in fact much larger than the SRAM counter array A) to be able to accommodate this burst, which negates the purpose of our scheme to save SRAM.

Our solution to this problem, specified in lines 10 through 13 in Algorithm 1, is again very simple. For each index i, we generate a random integer number uniformly distributed over $\{0, 1, 2, ..., 2^l - 1\}$, and assign this number to $A[i]$ (line 12). We need to somehow remember this value since it should be subtracted from the observed counter value the end of a measurement (counting) interval. This is achieved in line 13, by setting the initial value of $B[i]$ to $-A[i]$, for $i = 1, 2, ..., N$. [3] We assume that these initial values of $A[i]$ and $B[i]$ are not known to the adversary (explained next). Intuitively, this randomization ensures that given any workload (counter increment sequence), the counter overflow process will be quite smooth. We show that the queue size only needs to be around several hundreds to guarantee that the queue Q will not fill up with very high probability.

Note that here this adversary is defined entirely in the well-established context of randomized online algorithm design [9], and has nothing to do with its connotation in security and cryptography; It is defined as a deterministic or randomized algorithm that aims at maximizing the size of our queue, and has complete knowledge of our algorithm except (i.e., oblivious to) the initial values of $A[i]'s$ and $B[i]'s$. The sole purpose of introducing this adversary is to model the worst-case workload (i.e., counter increment sequence) to our architecture. It has nothing to do with security or cryptography.

We show the timing diagram of our scheme in Figure 2. Each cycle is the time it takes for a counter to be read from SRAM, incremented, and written back to SRAM. It is only slightly longer than two memory accesses since the increment operation takes much less time with hardware implementation. Therefore Figure 2 only shows a read step and a write step, omitting the tiny increment step in the middle. When a counter overflows, it is set to zero during the write step, and simultaneously its index is written into the queue. We only require the SRAM counter array and the queue to be implemented on two different SRAM modules so that these two writes can happen simultaneously; No pipelining logic is needed. Note that this requirement is much weaker than used in prior ap-

[3]This needs 1 extra DRAM bit per counter for the sign. An alternative method is to use an additional DRAM counter array (each counter has l bits) to remember the initial values. Both methods increases the DRAM cost only slightly (i.e., N and $l \times N$ bits respectively).

proaches [13, 11], in which different parts of a data structure need to be placed on different memory modules and nontrivial pipelining logic needs to be placed between them. Note that even in the worst case when there is one counter overflow every cycle, the read/write bandwidth of the queue is only 50% utilized because the insertion of an index (of an overflowed counter) only happens in the second part of a cycle. The first part of a cycle can always be used to move an index at the top of the queue to a DRAM write buffer, as soon as DRAM finishes the previous write operation. To summarize, we show that our architecture is able to handle one increment per cycle. Therefore, this concept of *cycle* will be used as the basic unit of time for our analysis, the topic of the next section.

Note that while our improved CMA scheme allows for the use of fewer SRAM bits per counter, it significantly increases the amount of traffic to DRAM (through the system bus). In fact, for every bit we save on SRAM counter size, the amount of traffic to DRAM is doubled. Although this problem also exists in other CMA schemes [13, 11], it is not as severe there since they use longer SRAM counter and therefore generate less traffic to DRAM. This increase in DRAM traffic may become a serious concern in today's network processors where system bus and DRAM bandwidth is heavily utilized already for packet processing. We acknowledge that this problem has not been addressed in this work. We plan to study and hopefully solve the problem in our future research.

4. ANALYSIS

In this section, we prove that even with a small buffer size, the probability that the FIFO queue Q overflows during a fairly long time interval is extremely small in the worst case. Some numerical examples will be shown in Section 5.1.

4.1 Notations and Summary of Results

In the previous section, we have defined the concept of a cycle (see Figure 2) and explained that our architecture can handle one increment per cycle. Therefore, we will use "cycle" as the basic unit of time. Throughout the following analysis, we assume there is an increment in each and every cycle (i.e., being continuously busy). It is intuitive that this assumption indeed represents the worst-case in the sense that the probability bounds derived for this case will be no better than allowing certain cycles to be idle (i.e., no increment during these cycles). We omit the proof of this fact here since it would be a tedious application of the elementary stochastic ordering theory [12]. We also assume that the sequence of array indices to be incremented is arbitrary. In other words, the probability bounds we derive in this section will apply to any increment sequence.

Let K be the number of slots, each of which stores an index to be flushed to DRAM, in the FIFO queue Q. Let N be the number of counters in the array and l be the size of each SRAM counter as defined before. Let μ be the ratio of DRAM access time to SRAM access time. For example, if SRAM is 30 times faster than DRAM, then μ is equal to $\frac{1}{30}$. Recall from the previous section that a cycle is approximately one SRAM read and one SRAM write (with the time needed for "increment by 1" omitted). Note that to flush an index i from Q to DRAM, we need to read the corresponding DRAM entry $B[i]$, add 2^l to it, and writes it back to DRAM. Again omitting the amount of time to perform "increment by 2^l", this transaction takes approximately two DRAM accesses. Therefore, it takes $1/\mu$ cycles to flush an index to the DRAM. Equivalently we can say that μ flushes are finished within one cycle, and μ can be viewed as the departure rate (per cycle) from the queue Q. The average arrival rate to the queue Q is 2^{-l}, since it takes 2^l increments on the average to guarantee a counter overflow[4]. Clearly, the average arrival rate to the queue has to be smaller than the departure rate (for the queue to be stable), and hence $2^{-l} < \mu$ or $2^l > \frac{1}{\mu}$ as we have stated in the previous section.

Let D_n be the event that one or more "flushes to DRAM" requests are dropped because Q is full when they came during the time interval of a total of n cycles (e.g., n increments). Again, in the following, all time parameters and values such as s and t are in the units of cycles. We shall establish a tight bound on the probability of this event D_n, as a function of aforementioned system parameters K, N, l, μ, and n. In this section, we shall fix n and will therefore shorten D_n to D.

We first show that $\Pr[D]$ is bounded by the summation of probabilities $\Pr[D_{s,t}]$, $0 \le s \le t \le n$, i.e.,

$$\Pr[D] \le \sum_{0 \le s \le t \le n} \Pr[D_{s,t}]$$

Here $D_{s,t}$ represents the event that the number of arrivals during the time interval $[s,t]$ is larger than the maximum possible number of departures in Q (if serving continuously), by more than the queue size K. Formally letting $b(s,t)$ denote the number of "flush to DRAM" requests generated during time interval $[s,t)$, then we have

$$\Pr[D_{s,t}] \equiv \Pr[b(s,t) - \mu(t-s) > K].$$

The inequality above is a direct consequence of the following lemma, which states that if the event D happens, at least one of the events $\{D_{s,t}\}_{0 \le s < t \le n}$ must happen.

LEMMA 1. $D \subseteq \bigcup_{0 \le s \le t \le n} D_{s,t}$

PROOF. Given an outcome $\omega \in D$, suppose an overflow happens at time z. The queue is clearly in the middle of a busy period at time z. Now suppose this busy period starts at y. Then the number of departures from y to z is equal to $\lfloor \mu(z-y) \rfloor$. Since a "flush to DRAM" request happens at time z to find the queue of size K full, $b(y,z)$, the total number of arrivals during time $[y,z]$ is at least $K+1+\lfloor \mu(z-y) \rfloor \ge K+\mu(z-y)$. In other words, $D_{y,z}$ happens and $\omega \in D_{y,z}$. This means for any outcome ω in the probability space, if $\omega \in D$, then $\omega \in D_{s,t}$ for some $0 \le s < t \le n$. □

Remark: As a consequence,

$$\Pr[D] \le \Pr[\bigcup_{0 \le s \le t \le n} D_{s,t}] \le \sum_{0 \le s \le t \le n} \Pr[D_{s,t}]$$

The rest of the section is devoted to deriving tight tail bounds for individual $\Pr[D_{s,t}]$ terms. We develop two main techniques in these derivations, both of which are based on the properties of the sum of independent random variables. These two bounds are described in detail in Section 4.2 and 4.3 respectively. We first provide a brief summary of these results in the following.

1. Our first bound, stated as Theorem 2, is derived using a simplified form of the well-known Chernoff bound. It has the following form:

$$\Pr[D_{s,t}] \equiv \Pr[b(s,t) - \mu(t-s) > K]$$
$$< e^{-2(K+\mu(t-s)-2^{-l}(t-s))^2 / \min\{t-s,N\}}$$

[4]One may feel and even we felt that since the counter values are initialized to be uniformly distributed between 0 and $2^l - 1$, it takes an adversary on the average 2^{l-1} to overflow a counter. This is not true because the adversary does not know the initial values of the counters and therefore may waste some effort (increments) on counters that have already overflowed once.

This is proven by showing that the centered random variable $b(s,t) - \mathrm{E}[b(s,t)]$ is the summation of $\min\{t-s, N\}$ independent random variables, each of which is a Poisson trial (explained later), which allows a Chernoff-type theorem optimized for the summation of Poisson trials to be applied. This bound shows that a small number of slots in the queue will allow us to achieve very small (queue) overflow probability.

2. The first bound still leaves considerable room for improvement because the Chernoff bound does not take advantage of the second moment information of $b(s,t)$ which can be derived analytically in our context. We will show in Section 4.3 that the variance of $b(s,t)$ is bounded by

$$\mathrm{Var}[b(s,t)] \leq \begin{cases} \frac{N}{4} & t-s \geq 2^{l-1}N, \\ \frac{(2^l - \frac{t-s}{N})(t-s)}{2^{2l}} & N \leq t-s < 2^{l-1}N, \\ \frac{(2^l-1)(t-s)}{2^{2l}} & 0 < t-s < N. \end{cases}$$

We derive a novel tail bound theorem that takes advantage of such additional information (the bound on the variance), which significantly improves the first bound for most s and t values, especially when the queue size K is relatively small. Let $\tau = t-s$ and let σ be the standard deviation of $b(s,t)$. Using this new theorem, we obtain the following tail bound for $\Pr[D_{s,t}]$.

$$\Pr[D_{s,t}] \equiv \Pr[b(s,t) - \mu\tau > K] < e^{-\frac{a^2}{2}(1-\frac{\epsilon}{3})},$$

where

$$a = \min\{a_0,\ \frac{K + (\mu - 2^{-l})\tau}{\sigma}\}, \epsilon = e^{\frac{a}{\sigma}} - 1,$$

and a_0 is the unique root of $g(a) = 4 - e^{\frac{a}{\sigma}} - \frac{ae^{\frac{a}{\sigma}}}{2\sigma}$ for $a \in (0, \sigma \ln 4)$.

3. we propose a hybrid overall bound which is obtained when we use the minimum of the first bound and the second on each $D_{s,t}$ term based on the comparison of the above two bounds. The detailed analytical comparison of the bounds can be found in Appendix 8.2.

We will show some numerical results of these three types of bounds in Section 5.1. We find that when the queue size K is relatively small, the second bound is orders of magnitude better than the first one. Using the hybrid bound only provides negligible improvement. However, when the queue size becomes large, the first bound can become better than the second one gradually. In this case, the second bound still proves its usefulness by making the hybrid case several times smaller than the first bound.

4.2 Bounding the probability of $D_{s,t}$ using simplified Chernoff bound

In this section, we use a variant of Chernoff bound optimized for a family of random variables called Poisson trials to bound $\Pr[b(s,t) - \mu(t-s) > K]$. We first state a technical lemma that will be used in the later derivation. Its proof can be found in Appendix 8.1.

LEMMA 2. *Let $V_j(t)$ be the value of counter j at time t. Given an arbitrary counter increment sequence (allowing idling), we have (i) $V_1(t)$, $V_2(t)$, ..., $V_N(t)$ are mutually independent random variables and (ii) each of them has the following distribution:*

$$V_j(t) = \begin{cases} 0 & \textit{with probability } 2^{-l}, \\ 1 & \textit{with probability } 2^{-l}, \\ \ldots & \\ 2^l - 1 & \textit{with probability } 2^{-l}. \end{cases}$$

Let b_j be the number of "flush to DRAM" requests generated by the counter j during the time interval $[s,t]$. Clearly we have $\sum_{j=1}^{N} b_j = b(s,t)$. Let c_j be the number of increments to counter j during time period $[s,t]$, $j = 1, 2, ..., N$. In all our derivations $c_j's$ are allowed to take arbitrary values, and are considered constants once the values are chosen. We first prove some properties of b_j and c_j in the following theorem, which will be used in the later derivations. In that theorem we use a new notation: for any real number x, we define $\{x\}$ as $x - \lfloor x \rfloor$. In other words, $\{x\}$ is the fraction part of x.

THEOREM 1. *(i) $b_1, b_2, ..., b_N$ are mutually independent random variables during any time interval $[s,t]$; (ii) $\mathrm{E}[b_j] = c_j 2^{-l}$; (iii) $\mathrm{Var}[b_j] = \{c_j 2^{-l}\}(1 - \{c_j 2^{-l}\})$; (iv) $E[b(s,t)] = \frac{t-s}{2^l}$*

PROOF. It is easy to obtain that

$$b_j = \left\lfloor \frac{c_j + V_j(s)}{2^l} \right\rfloor$$

that is, b_j is a function of the counter value V_j at cycle s and the constant c_j. Since $V_1(s)$, $V_2(s)$,...,$V_N(s)$ are mutually independent due to Lemma 2, b_j are mutually independent as well for any possible s. So (i) holds.

Since by Lemma 2, we know that $V_j(s)$ is uniformly distributed over $\{0, 1, ..., 2^l - 1\}$, the distribution of b_j can be simplified as

$$b_j = \begin{cases} \lfloor \frac{c_j}{2^l} \rfloor & \text{with probability } 1 - \{2^{-l}c_j\}, \\ \lfloor \frac{c_j}{2^l} \rfloor + 1 & \text{with probability } \{2^{-l}c_j\}. \end{cases}$$

For simplicity let α and β denote $\frac{c_j}{2^l}$ and $\lfloor \frac{c_j}{2^l} \rfloor$ respectively. $\mathrm{E}[b_j] = \beta(\beta + 1 - \alpha) + (\beta+1)(\alpha - \beta) = \alpha$. So (ii) holds.

We can also obtain that $\mathrm{E}[b_j^2] = \beta^2(\beta + 1 - \alpha) + (\beta+1)^2(\alpha - \beta) = 2\alpha\beta - \beta^2 + \alpha - \beta$. Therefore

$$\begin{aligned} \mathrm{Var}[b_j] &= \mathrm{E}[b_j^2] - (\mathrm{E}[b_j])^2 \\ &= \alpha - \beta - (\alpha - \beta)^2 = (\alpha - \beta)(1 - \alpha + \beta) \end{aligned}$$

Thus we finish the proof of (iii).

Finally, we have $\mathrm{E}[b(s,t)] = \sum_{j=1}^{N} \mathrm{E}[b_j] = \frac{\sum_{j=1}^{N} c_j}{2^l} = \frac{(t-s)}{2^l}$. Thus (iv) holds.

□

A direct consequence of the above theorem is that $b_j - E[b_j]$, $j = 1, 2, ..., N$, are mutually independent random variables and

$$b_j - \mathrm{E}[b_j] = \begin{cases} -\{2^{-l}c_j\} & \text{with probability } 1 - \{2^{-l}c_j\}, \\ 1 - \{2^{-l}c_j\} & \text{with probability } \{2^{-l}c_j\}. \end{cases}$$

Such random variables are called (centered) *Poisson trials*. Note that $b(s,t) - E[b(s,t)]$ is the summation of these independent Poisson trails $b_j - E[b_j]$, $j = 1, 2, ..., N$. The following variant of Chernoff bound theorem can be used to derive a tail bound on $\Pr[b(s,t) - \mu(t-s) > K]$.

LEMMA 3 (CITED FROM [1]). *Let $X_1, X_2, ..., X_m$ be mutually independent random variable such that, for $1 \leq j \leq m$, $\Pr[X_j = 1 - p_j] = p_j$ and $\Pr[X_j = -p_j] = 1 - p_j$, where $0 < p_j < 1$. Then, for $X = \sum_{j=1}^{m} X_j$ and $a > 0$,*

$$\Pr[X > a] < e^{-2a^2/m}$$

To apply this theorem we map all the parameters to our context. m corresponds to $\min\{t-s, N\}$ because during the first $t-s$ cycles at most $t-s$ counters are updated and X_j corresponds to

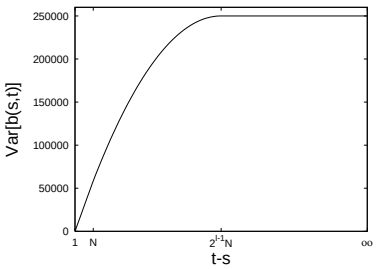


Figure 3: variance of $b(s,t)$.

$b_j - 2^{-l}c_j$ so that $p_j = \{2^{-l}c_j\}$ according to Theorem 1. Thus X corresponds to $b(s,t) - 2^{-l}(t-s)$. To simplify the formula, we use τ to replace $t-s$ in the following derivation when it is more convenient.

Therefore

$$\begin{aligned}\Pr[b(s,t) - \mu(t-s) > K] &= \Pr[X + 2^{-l}\tau - \mu\tau > K] \\ &= \Pr[X > K + \mu\tau - 2^{-l}\tau]\end{aligned}$$

We set a in Lemma 3 to $K + \mu\tau - 2^{-l}\tau$ and finally obtain

THEOREM 2. *For any $s < t$, let $\tau = t - s$.*

$$\Pr[D_{s,t}] \equiv \Pr[b(s,t) - \mu\tau > K] < e^{-2(K+\mu\tau-2^{-l}\tau)^2/\min\{\tau,N\}}$$

The computational complexity to obtain the overall bound of $\Pr[D]$ is $O(n)$ because the bound on $\Pr[D_{s,t}]$ is a shift-invariant in the sense that it it same as the bound on $\Pr[D_{s+\Delta,t+\Delta}]$, i.e., $\Pr[D_{s,t}]$ is only a function of $\tau = t - s$. So we only need to compute such a bound once and multiply it by $n - \tau + 1$ to account for the overall bound on $\Pr[D]$. This complexity may be further reduced since $\Pr[D_{s,t}]$ is monotonically decreasing when τ increases. Then if the value of $\Pr[D_{s,t}]$ already decreases to a negligible level (e.g., 2.2251×10^{-308} in MATLAB 7.0), we need not compute the following terms.

4.3 Using second moment information to obtain new bound of $\Pr[D_{s,t}]$

So far we obtain an upper bound of $\Pr[D_{s,t}]$ using Theorem 2. However it does not fully take advantage of the available information such as a bound on the variance of $b(s,t)$ which we can derive analytically. Thus it is possible to obtain a better bound in some cases by combining this variance information. Next we first show (in Theorem 3) how to derive a bound on the variance of $b(s,t)$. Then we propose a new tail bound which takes advantage of both the fact that $b(s,t)$ is the sum of independent random variables $b_j, j = 1, 2, ..., N$ and the fact that we have a bound on its variance. We will show how to adapt this theorem to our context to obtain another often stronger upper bound of $\Pr[D_{s,t}]$.

4.3.1 *Bounding the variance of $b(s,t)$*

THEOREM 3.

$$\mathrm{Var}[b(s,t)] \le \begin{cases} \frac{N}{4} & t-s \ge 2^{l-1}N, \\ \frac{(2^l - \frac{t-s}{N})(t-s)}{2^{2l}} & N \le t-s < 2^{l-1}N, \\ \frac{(2^l-1)(t-s)}{2^{2l}} & 0 < t-s < N. \end{cases}$$

PROOF. Because $b_1, b_2, ..., b_N$ are mutually independent by Theorem 1, $\mathrm{Var}[b(s,t)] = \sum_{j=1}^{N} \mathrm{Var}[b_j]$. In the following we prove the theorem in three different cases:

i) By Theorem 1, we have

$$\begin{aligned}\mathrm{Var}[b_j] =& \{\frac{c_j}{2^l}\}(1 - \{\frac{c_j}{2^l}\}) \\ \le& \frac{(\{\frac{c_j}{2^l}\} + 1 - \{\frac{c_j}{2^l}\})^2}{4} = \frac{1}{4}\end{aligned}$$

The equality is satisfied if and only if $\frac{c_j}{2^l} - \lfloor \frac{c_j}{2^l} \rfloor = 0.5$ which means c_j is equal to $2^l \times d + 2^{l-1}$ where d is a constant. So if $t - s \ge 2^{l-1}N$, i.e., it is large enough to spread to N different counters with 2^{l-1} increments per counter, $\mathrm{Var}[b(s,t)] \le \sum_{j=1}^{N} \frac{1}{4} = \frac{N}{4}$. The equality is satisfied only when $c_j = 2^l d_j + 2^{l-1}, j = 1, 2, ..., N$, where $d_j \ge 0$.

ii) When $t - s < 2^{l-1}N$ the total number of increments (i.e., $t - s$) is not large enough to spread out to all N counters with 2^{l-1} increments each. So the question is how to allocate the total $t - s$ increments to different counters to maximize the variance. We show that the best strategy is spreading the increments to as many different counters as possible. Because each c_j can be represented by $2^l d_j + y_j$ where $0 \le y_j \le 2^l - 1$ and $\mathrm{Var}[b_j]$ is maximized ($= \frac{1}{4}$) when $y_j = 2^{l-1}$ as we showed in the above, we conclude that the variance is maximized when all $d_j = 0, j = 1, 2, ..., N$ and $y_j \le 2^{l-1}$. Then $c_j = y_j \le 2^{l-1}$ such that $\lfloor 2^{-l}c_j \rfloor = 0$. Therefore

$$\begin{aligned}\mathrm{Var}[b(s,t)] =& \sum_{j=1}^{N} \mathrm{Var}[b_j] = \frac{1}{2^{2l}} \sum_{j=1}^{N} y_j(2^l - y_j) \\ =& 2^{-l}(t-s) - 2^{-2l} \sum_{j=1}^{N} y_j^2\end{aligned}$$

Then when $N \le t - s < 2^{l-1}N$, $(t-s)^2 = (\sum_{j=1}^{N} y_j)^2 \le N \sum_{j=1}^{N} y_j^2$. So $\sum_{j=1}^{N} y_j^2 \ge \frac{(t-s)^2}{N}$. The equality is satisfied, i.e., $\sum_{j=1}^{N} y_j^2$ is minimized, if an only if $y_1 = y_2 = ... = y_N = \frac{t-s}{N}$. Thus $\mathrm{Var}[b(s,t)] \le \frac{(2^l - \frac{t-s}{N})(t-s)}{2^{2l}}$.

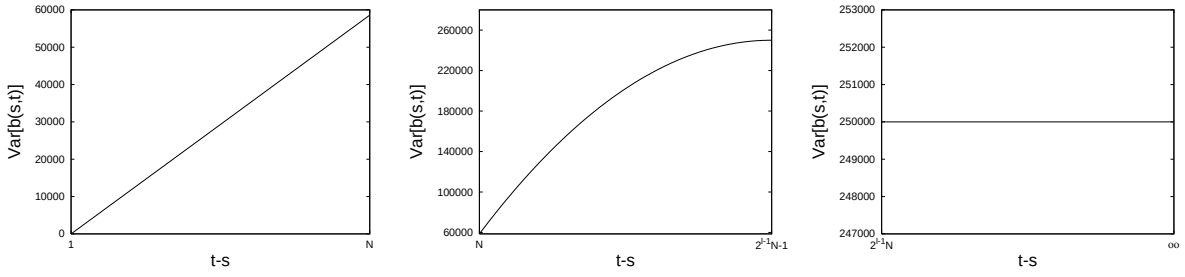
iii) When $t - s < N$, at most $t - s$ counters may be incremented in $t - s$ cycles. Assume these counters are $\psi_1, \psi_2, ..., \psi_{t-s}$. Similarly to the above derivation we have

$$(t-s)^2 = (\sum_{j=1}^{t-s} y_{\psi_j})^2 \le (t-s) \sum_{j=1}^{t-s} y_{\psi_j}^2$$

The last inequality is due to Cauchy Schwartz's inequality. Therefore $\sum_{j=1}^{t-s} y_{\psi_j}^2 \ge t - s$ where the equality is satisfied if an only if $y_{\psi_1} = y_{\psi_2} = ... = y_{\psi_{t-s}} = 1$. Thus $\mathrm{Var}[b(s,t)] \le \frac{(2^l-1)(t-s)}{2^{2l}}$.

□

For example, setting $N = 1,000,000$ SRAM counters and $l = 4$ bits we plot the curve of the bound on $\mathrm{Var}[b(s,t)]$ by varying the value of $t - s$ in Figure 3. To make it clearer we divide it into three parts (shown in Figure 4) according to three different cases of the variance. It is clear that if $t - s < N$ the corresponding variance bound is linearly increasing with $t - s$ (Figure 4(a)); if $N \le t - s < 2^{l-1}N$, the variance bound is monotonically increasing with $t - s$ (Figure 4(b)); otherwise, the corresponding variance bound stays the same everywhere (Figure 4(c)), i.e., $\frac{N}{4}$. In all three cases, the variance bound is always a function of $t - s$. Therefore, for simplicity we again use τ to replace $t - s$ in the following derivations where it is more convenient.



(a) $0 \leq t-s < N$ (b) $N \leq t-s < 2^{l-1}N$ (c) $t-s \geq 2^{l-1}N$

Figure 4: Expanded depiction of variance of $b(s,t)$ for three cases.

4.3.2 A new tail bound theorem

Now we establish a new tail bound theorem on the summation of independent random variables that takes advantage of both their independence and the fact that their total variance is small. This new theorem will be used to bound $\Pr[b(s,t) - \mu(t-s) > K]$. This theorem improves Theorem A.1.19 in [1, pp.270] by sharpening ϵ to $\frac{\epsilon}{3}$ on the RHS of the inequality. The proof can be found in our technical report [16]. We are able to obtain tail bounds from this Theorem that are much tighter than obtainable from Theorem 2 in some cases (e.g., when the queue size K is small.) and the numerical results will be shown in Section 5.1.

THEOREM 4. *Given any $\theta > 0$ and $\epsilon > 0$, the following holds: Let $W_j, 1 \leq j \leq m$, m arbitrary, be independent random variables with $\mathrm{E}[W_j] = 0$, $|W_j| \leq \theta$ and $\mathrm{Var}[W_j] = \sigma_j^2$. Let $W = \sum_{j=1}^{m} W_j$ and $\sigma^2 = \sum_{i=1}^{m} \sigma_j^2$ so that $\mathrm{Var}[W] = \sigma^2$. Let $\delta = \ln(1+\epsilon)/\theta$. Then for $0 < a \leq \delta\sigma$,*

$$\Pr[W > a\sigma] < e^{-\frac{a^2}{2}(1-\frac{\epsilon}{3})}$$

4.3.3 Mapping the new tail bound to our context

We now map this tail bound theorem to our context as follows. W corresponds to the deviation of $b(s,t)$ from its expectation ($= b(s,t) - \mathrm{E}[b(s,t)]$). The deviation of each b_j from its expectation corresponds to W_j, i.e., $W_j = b_j - \mathrm{E}[b_j]$. And σ_j^2 and σ^2 correspond to $\mathrm{Var}[b_j]$ and $\mathrm{Var}[b(s,t)]$ respectively. Theorem 4 essentially states that the probability that W is larger than a times standard deviation is less than $e^{-\frac{a^2}{2}(1-\frac{\epsilon}{3})}$. However, this is true only for $a \leq \delta\sigma$, and δ, ϵ and θ are related by $e^{\delta\theta} \leq 1+\epsilon$. Therefore we need to bound θ, which is the upper bound for $|W_j|$, $j = 1, 2, ..., N$, since otherwise either ϵ has to be large or a has to be very small, and both cases lead to trivial or loose tail bounds. Clearly $|W_j| \leq \max\{\frac{c_j}{2^l} - \lfloor\frac{c_j}{2^l}\rfloor, \lfloor\frac{c_j+2^l-1}{2^l}\rfloor - \frac{c_j}{2^l}\} < 1$. So we set θ to 1. So far we know that

$$\begin{aligned}\Pr[b(s,t) - \mu(t-s) > K] &= \Pr[W + 2^{-l}\tau - \mu\tau > K] \\ &= \Pr[W > K + \mu\tau - 2^{-l}\tau]\end{aligned}$$

To apply Theorem 4 we need to find the corresponding a given τ to make $\Pr[W > K + \mu\tau - 2^{-l}\tau] \leq \Pr[W > a\sigma]$. Therefore, we obtain another constraint, which is, $a\sigma \leq K + \mu\tau - 2^{-l}\tau$. Now we take out this constraint for a moment and look at how we optimize the rest. We certainly want a to be as large as possible. However, we have the constraint $a \leq \delta\sigma$ and δ is in turn constrained by $e^{\theta\delta} \leq 1+\epsilon$. Note that a large a will result in large ϵ and $1-\epsilon/3$ can be small or even negative. Intuitively there is an optimal tradeoff point in between. So we can formulate the problem as follows:

$$\begin{aligned}&\textit{maximize} && \frac{a^2}{2}(1-\frac{\epsilon}{3}) \\ &\textit{subject to} && 0 < a \leq \delta\sigma \\ & && e^{\delta} - 1 \leq \epsilon < 3 \\ & && a\sigma \leq K + \mu\tau - 2^{-l}\tau\end{aligned}$$

In the following we show how to resolve this optimization problem. We know that

$$\frac{a^2}{2}(1-\frac{\epsilon}{3}) \leq \frac{a^2}{2}(1-\frac{e^{\frac{a}{\sigma}}-1}{3}) = \frac{a^2}{6}(4-e^{\frac{a}{\sigma}})$$

since $a \leq \delta\sigma$. The RHS of the above equation can be viewed as a function of a, denoted as $f(a)$. Its first derivative $f'(a)$ is equal to

$$\frac{4a}{3} - \frac{ae^{\frac{a}{\sigma}}}{3} - \frac{a^2e^{\frac{a}{\sigma}}}{6\sigma}$$

The maxima of $f(a)$ should be achieved at some of the roots of $f'(a) = 0$, or at the boundary. We first compute and check out these roots (to see if it is a maxima or minima). Since $a > 0$, $f'(a) = 0$ can be simplified as $g(a) = 0$ where

$$g(a) = 4 - e^{\frac{a}{\sigma}} - \frac{ae^{\frac{a}{\sigma}}}{2\sigma}$$

It is clear that $g'(a) = -\frac{3e^{\frac{a}{\sigma}}}{2} - \frac{ae^{\frac{a}{\sigma}}}{2\sigma} < 0$ since $\frac{a}{\sigma} > 0$. Therefore $g(a)$ is a strictly monotonically decreasing function of a. Note that when we tune the parameters to optimize our tail bound, we always keep ϵ under 3 (otherwise the tail bound will be trivial). Since $e^{\frac{a}{\sigma}} \leq e^{\delta} \leq \epsilon + 1 < 4$ we also have $a < \sigma\ln 4$. Because $g(0) = 3 > 0$ and $g(\sigma\ln 4) = -2\ln 4 < 0$ the equation $g(a) = 0$ has a unique root denoted as a_0 between $(0, \sigma\ln 4)$ given σ. We can compute a_0 numerically since there is no closed form solution for it. Recall that $f'(a) = \frac{a}{3}g(a)$. So within $(0, a_0)$, $f'(a) > 0$ and within $(a_0, \sigma\ln 4)$, $f'(a) < 0$. Thus we conclude that $f(a)$ is first monotonically increasing and begins to decrease after a reaches a_0. Therefore $f(a)$ exhibits a maximal value achieved at a_0 which can be calculated numerically.

Combining the third constraint in the formulated problem, we obtain the following solution for the optimization problem. If $K + (\mu - 2^{-l})\tau \geq a_0\sigma$, then the maximization is achieved at $a = a_0$ and $\epsilon = e^{\frac{a}{\sigma}} - 1$. If, however, $K + (\mu - 2^{-l})\tau < a_0\sigma$, then, a must be smaller than a_0 due the third constraint. We know $f(a)$ is

monotonically increasing when $a \in (0, a_0]$. Therefore we set a to be closest possible to a_0 under this constraint so that the objective is maximized: $a = \frac{K+(\mu-2^{-l})\tau}{\sigma}$ and $\epsilon = e^{\frac{a}{\sigma}} - 1$.

Therefore, we obtain

THEOREM 5. *For any $s < t$, let $\tau = t - s$ and let σ be the standard deviation bound of $b(s,t)$ as derived in Theorem 3 (RHS of the inequality),*

$$\Pr[D_{s,t}] \equiv \Pr[b(s,t) - \mu\tau > K] < e^{-\frac{a^2}{2}(1-\frac{\epsilon}{3})},$$

where $a = \min(a_0, \frac{K+(\mu-2^{-l})\tau}{\sigma})$, $\epsilon = e^{\frac{a}{\sigma}} - 1$, and a_0 is the unique root of $g(a) = 4 - e^{\frac{a}{\sigma}} - \frac{ae^{\frac{a}{\sigma}}}{2\sigma}$ for $a \in (0, \sigma \ln 4)$.

The computational complexity to obtain the overall bound of $\Pr[D]$ is $O(N)$, which is much smaller than that using the variant of the Chernoff bound, i.e., $O(n)$, when $N << n$, due to the following two facts. Like the bound from Theorem 2, this bound of $\Pr[D_{s,t}]$ is again shift-invariant, i.e., $\Pr[D_{s,t}] = \Pr[D_{s+\Delta,t+\Delta}]$. So it only needs to be computed once and multiplied by $n - \tau + 1$ to account for the overall bound on $\Pr[D]$. The other fact we should thank to is that when $\tau \geq 2^{l-1}N$ the standard deviation of σ keeps the same ($= \frac{\sqrt{N}}{2}$) no matter how large τ is. Hence the root a_0 of $f(a)$ also keeps the same for all $2^{l-1}N \leq \tau \leq n$ because $g(a)$ is always the same. And so do the bounds of $\Pr[D_{s,t}]$ when $\tau > 2^{l-1}N$ (denoted as ρ). This means $\Pr[D] = \sum_{0 \leq t-s < 2^{l-1}N} \Pr[D_{s,t}] + \rho \sum_{2^{l-1}N \leq t-s \leq n} 1$.

So far we have two different tail bound on $\Pr[D]$ by Theorem 2 and 5 respectively. Carefully analyzing and comparing these two bounds, we found that the bound on $\Pr[D_{s,t}]$ from Theorem 2 is generally better but the one from Theorem 5 outperforms for some $t - s$ values. The detailed analysis is shown in Appendix 8.2. This suggests a hybrid bound which is obtained when we apply the smaller bound between the bounds from Theorem 2 and Theorem 5 for each $\Pr[D_{s,t}]$ to achieve the tightest bound. In other words, when we denote the bounds from Theorem 2 and Theorem 5 on term $D_{s,t}$ as $\Omega_1(s,t)$ and $\Omega_2(s,t)$ respectively, the hybrid bound is

$$\Pr[D_{s,t}] \leq \min\{\Omega_1(s,t), \Omega_2(s,t)\}$$

The numerical results shown in Section 5.1 will provide some examples for comparison of these three bounds. We observe that although $\Omega_1(s,t)$ is better on the most of range $\Omega_2(s,t)$ is able to improve the overall bound of $\Pr[D]$ significantly.

5. EVALUATION

In this section, we evaluate the operational performance of our design. We first present in Section 5.1 numerical examples of tail bounds derived in the previous section under a set of typical parameter configurations. Note that these tail bounds are derived for the worst case scenarios. In practice, as demonstrated by experiments on real-world traffic traces described in Section 5.2, typical distributions in Internet traffic are nowhere close to this worst case. This translates to much smaller values of the maximum queue length being observed in practice. While the analysis predicts that a queue with a few hundred slots would be required to achieve a negligibly small probability of overflow, in practice the queue length never exceeded 18 for experiments with hundreds of millions of counter increments.

K	Theorem 2	Theorem 5	Hybrid
200	trivial (≥ 1)	trivial (≥ 1)	5.0×10^{-5}
300	1.4×10^{-6}	trivial (≥ 1)	2.4×10^{-14}
4,000	7.8×10^{-274}	1.4×10^{-10}	$\leq 2.2251 \times 10^{-308}$

Table 1: Probability of overflow when $N = 10^6$, $n = 10^{12}$, $\mu = 1/12$ and $l = 4$ bits

K	Theorem 2	Theorem 5	Hybrid
500	trivial (≥ 1)	trivial (≥ 1)	1.1×10^{-11}
3033	1.4×10^{-6}	trivial (≥ 1)	8.7×10^{-142}

Table 2: Probability of overflow when $N = 10^6$, $n = 10^{12}$, $\mu = 1/30$ and $l = 5$ bits

5.1 Numerical examples of the tail bounds

In this section we present a set of numerical results computed from the tail bound theorems derived in the previous section using MATLAB 7.0. We only use a set of representative parameters; other parameter settings result in similar observations. Like before, we assume that there are $N = 1,000,000$ counters in our architecture. We also assume that the measurement/counting interval has $n = 10^{12}$ cycles (one counter increment per cycle as discussed in Section 2). This interval is about several hours long when we assume that each packet arrival triggers the increment to one counter and the link speed is 40 Gbps (OC-768), and it is longer (in terms of actual time) with lower link speeds. We construct two sets of numerical examples as stated in Section 2 by setting the ratio of DRAM access speed to SRAM access speed μ to $\frac{1}{12}$ and $\frac{1}{30}$ respectively. Recall that μ has to be larger than 2^{-l} to make the queue stable. Therefore we set l to 4 and 5 bits respectively in above two examples, the aforementioned minimum SRAM counter size that is theoretically possible.

Table 1 shows the probabilities of queue overflow computed by the three aforementioned tail bounds (Theorem 2, Theorem 5, and the hybrid bound) given three different queue buffer sizes (200 slots, 300 slots and 4,000 slots). The size of each slot in this case is 20 bits ($= \log_2 1,000,000$). In the table "trivial" means that the resulting tail bound is larger than 1 so that it makes no sense. We observe that the bound from Theorem 2 is generally better than that from Theorem 5 (see the case of $4,000$ slots to make it clear). But the bound from Theorem 5 is still valuable because it can improve the first bound significantly. For example, although the first bound is trivial in the case of 200 slots the resulting hybrid bound reaches a good level with the help of the second bound. It is clear that the hybrid bound is orders of magnitude better than any of separate bounds and achieves very desirable level when the buffer size is quite small (around 300 slots). We observe the similar result in Table 2. We also find that the bound from Theorem 5 can improve the bound from Theorem 2 more significantly when the service rate ($1/2^l$) is closer to the average arrival rate (μ) by comparing the second rows of the tables.

Table 3 compares the proposed scheme with the other three existing schemes, i.e, LCF in [13], $LR(b)$ in [11] and the naïve approach of implementing all counters in SRAM, in terms of memory (SRAM and DRAM) consumption and implementation complexity. We only show the data for the case of $\mu = 1/30$ due to interest of space. The data for the other case can be easily computed. Our scheme can be implemented using $5 + \epsilon$ (ϵ around 0.01) bits

	Naive	LCF	$LR(b)$	Ours
Counter memory	64Mb SRAM	9Mb SRAM, 64Mb DRAM	9Mb SRAM,64Mb DRAM	5Mb SRAM, 65Mb DRAM
Control memory	None	20Mb SRAM	2Mb SRAM	10Kb SRAM
Control logic	None	Hardware heap	Aggregated bitmap	FIFO queue
Implementation Complexity	Very low	High	Low	Very Low

Table 3: Cost-benefit comparison for different schemes for the reference system with a million 64-bit counters, $\mu = 1/30$, and $K = 500$ slots.

of SRAM per counter.[5] When compared to the best one of the previous approaches, i.e., $LR(b)$, our scheme achieves a 2.2-fold reduction in SRAM usage. In addition, our scheme, which uses a simple FIFO queue as its control logic, is much easier to implement than $LR(b)$, which needs a pipelined implementation of a tree-like structure (called aggregated bitmap).

5.2 Simulation using real-world Internet traffic

In this section, we evaluate our statistics counter architecture using real-world Internet traffic traces, with each packet arrival triggering one or more counter increments. The experimental results show that the maximum queue length over time is more than two orders of magnitude smaller than the values we set previously in Section 5.1 to obtain the desirable bound of the overflow probability. This reflects the fact that the analytical results derived for the worst case are not typically observed in practice.

For our evaluations, we would like to use long packet header traces to capture low probability events. Most publicly available traces are divided into small time scales and anonymized separately so that the small pieces cannot be merged into a big one. Fortunately, we were able to obtain two large traces that are publicly available for research purposes. They were collected at different locations in the Internet, namely University of Southern California (USC) and University of North Carolina (UNC) respectively. The trace from USC was collected at their Los Nettos tracing facility on Feb. 2, 2004 and the trace from UNC was collected on a 1 Gbps access link connecting to the campus to the rest of the Internet, on April 24, 2003. Both traces are quite large: the trace from USC has 120,773,099 packet headers and around 8.6 million flows; the trace from UNC has 198,944,706 packet headers and around 13.5 million flows.

We use the same parameter settings as in Section 5.1 where N = 1,000,000 and run the experiments for both traces with l=4 bits, $\mu = 1/12$ and l = 5 bits, $\mu = 1/30$, respectively. For each packet header in the trace we hash its flow label (i.e., $\langle$ source IP address, destination IP address, source port, destination port, protocol $\rangle$) and the result is viewed as the index to the counter array. This operation is performed in a number of networking applications [15, 6, 7]. Table 4 shows the maximal and average queue sizes during the experiments. For comparison we also compute the theoretical hybrid bounds of overflow for every experiment we run. The results are shown in Table 5. In the computation we tune the buffer size (the fourth column) to make the overflow probability (the fifth column) a little larger than 10^{-6}. Thus we should see more than one hundred overflows (i.e., larger than the buffer sizes we set) on the average in all the experiments. But we did not observe any overflows in all the experiments. The maximal queue sizes in Table 4 are actually multiple times smaller than the configured buffer sizes in Table 5. For example, in the second experiment (the third row in Table 4) for the trace USC where SRAM is 30 times faster than DRAM (i.e., $\mu = 1/30$), the observed maximal queue size is only 61 which is around 5-6 times smaller than the result from the theoretical analysis, i.e., 336 with 1.08×10^{-6} overflow probability (the third row in Table 5). This is mainly because the bounds in Section 4 are derived for the worst case and the real-world traffic does not exhibit worst-case behavior. Relaxations in our derivations of some inequalities also contribute to the difference.

[5]In that table, our scheme uses 1 extra bit (in DRAM) per counter to remember the sign of the counter value as we described before.

Trace	SRAM counter size (in bits)	μ	Queue Size	
			Max	Average
USC	4	1/12	21	1.6
	5	1/30	61	6.0
UNC	4	1/12	23	1.7
	5	1/30	72	7.0

Table 4: Statistics of queue size with single counter increment

Trace	SRAM counter size (in bits)	μ	Buffer size	Overflow probability
USC	4	1/12	176	1.05×10^{-6}
	5	1/30	336	1.08×10^{-6}
UNC	4	1/12	176	1.73×10^{-6}
	5	1/30	336	1.78×10^{-6}

Table 5: Overflow probabilities with single counter increment

In the next set of experiments, we let each packet arrival result in increments to multiple counters, which is typical in counting sketches or other applications such as [4, 3, 14], as we described before. Since this may make the counter increment process more "bursty", we would like to know its impact on the queue length. Note that during each cycle we still only perform one increment. Using the same traces (USC and UNC) above, each packet arrival results in increments to 4 different counters, indexed by four independent hash functions computed over the flow label fields from the packet header. The statistics of the observed queue size are shown in Table 6. Again we compute the theoretical hybrid bounds of overflow for every experiment we run and tune the overflow probability a little larger than 10^{-6} which should introduce more than four hundred overflow events on the average. The computation results are shown in Table 7. We observe the very similar experimental behavior as in the earlier experiments and the same relationship between the experiment results and the theoretical values. Therefore, we conclude that the "burstiness" caused by multiple counter increments per packet arrival does not seem to impact the practical maximal queue size significantly.

Trace	SRAM counter size (in bits)	μ	Queue Size	
			Max	Average
USC	4	1/12	22	1.7
	5	1/30	75	6.6
UNC	4	1/12	30	1.8
	5	1/30	82	7.3

Table 6: Statistics of queue size with 4 counter increments

Trace	SRAM counter size (in bits)	μ	Buffer size	Overflow probability
USC	4	1/12	182	1.16×10^{-6}
	5	1/30	347	1.07×10^{-6}
UNC	4	1/12	182	1.92×10^{-6}
	5	1/30	347	1.76×10^{-6}

Table 7: Overflow probabilities with 4 counter increments

6. CONCLUSION

Supporting high-speed increments to a large number of counters using limited amounts of fast memory is the problem addressed in this paper. Solutions proposed in recent works have used hybrid architectures where small counters in SRAM are incremented at high speed, and occasionally written back ("flushed") to larger counters in DRAM. In this work, we present a novel hybrid SRAM/DRAM counter architecture that uses the optimal amount of fast memory, while minimizing the complexity of the counter management algorithm. Our design uses a small write-back buffer (in SRAM) that stores indices of the overflowed counters (to be flushed to DRAM) and an extremely simple randomized algorithm to statistically guarantee that SRAM counters do not overflow in bursts large enough to fill up the write-back buffer even in the worst case. The statistical guarantee of the algorithm is proven through a combination of worst-case analysis for characterizing the worst case counter increment sequence and a new tail bound theorem for bounding the probability of filling up the write-back buffer. Experiments with real Internet traffic traces show that the actual queue lengths observed in practice are orders of magnitude smaller than the analytical bounds derived for the worst case.

7. REFERENCES

[1] N. Alon and J. H. Spencer. *The Probabilistic Method*. A Wiley-Interscience Publication, 2000.

[2] S. Cohen and Y. Matias. Spectral bloom filters. In *Proc. of ACM SIGMOD Conference on Management of Data*, 2003.

[3] G. Cormode and S. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. *Journal of Algorithms*, 2004.

[4] C. Estan and G. Varghese. New directions in traffic measurement and a ccounting. In *Proc. of ACM SIGCOMM*, August 2002.

[5] B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen. Sketch-based change detection: Methods, evaluation, and applications. In *Proc. of ACM SIGCOMM IMC*, October 2003.

[6] A. Kumar, M. Sung, J. Xu, and J. Wang. Data streaming algorithms for efficient and accurate estimation of flow size distribution. In *Proc. of ACM SIGMETRICS*, 2004.

[7] A. Kumar, M. Sung, J. Xu, and E. Zegura. A data streaming algorithm for estimating subpopulation flow size distribution. In *Proc. ACM SIGMETRICS*, June 2005.

[8] A. Kumar, J. Xu, J. Wang, O. Spatschek, and L. Li. Space-code bloom filter for efficient per-flow traffic measurement. In *Proc. of IEEE INFOCOM*, March 2004.

[9] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[10] S. Muthukrishnan. Data streams: algorithms and applications. available at http://athos.rutgers.edu/∼muthu/.

[11] S. Ramabhadran and G. Varghese. Efficient implementation of a statistics counter architecture. In *Proc. ACM SIGMETRICS*, June 2003.

[12] S.M. Ross. *Stochastic Processes*. Wiley, 1995.

[13] D. Shah, S. Iyer, B. Prabhakar, and N. McKeown. Maintaining statistics counters in router line cards. In *IEEE Micro*, 2002.

[14] Y. Zhang, S. Singh, S. Sen, N. Duffield, and C. Lund. Online identification of hierarchical heavy hitters: Algorithms, evaluation, and application. In *Proc. of ACM SIGCOMM IMC*, October 2004.

[15] Q. Zhao, A. Kumar, J. Wang, and J. Xu. Data streaming algorithms for accurate and efficient measurement of traffic and flow matrices. In *Proc. of ACM SIGMETRICS*, June 2005.

[16] Q. Zhao, J. Xu, and Z. Liu. Design of a novel statistics counter architecture with optimal space and time efficiency. In *Technical Report*, January 2006.

8. APPENDIX

8.1 Proof of Lemma 2

Recall that the values of $A[i]$, $i = 1, 2, ..., N$ are initialized to values uniformly distributed over $\{0, 1, ..., 2^l - 1\}$ independently. In other words, $V_1(0)$, $V_1(0)$, ..., $V_N(0)$ are mutually independent and each of them has the following distribution:

$$V_j(0) = \begin{cases} 0 & \text{with probability } 2^{-l}, \\ 1 & \text{with probability } 2^{-l}, \\ ... & \\ 2^l - 1 & \text{with probability } 2^{-l}. \end{cases}$$

In other words, the lemma holds at cycle 0. Now assuming that at cycle k the Lemma holds, we prove that the Lemma also holds at cycle $k + 1$ in the following. If cycle $k + 1$ is idle, i.e., there is no counter update at this cycle, the values of all counters remain the same, so do the independence and the distributions. If there is an counter increment to counter j at that cycle, clearly the values of all other counters remain the same. And $V_j(t+1)$ has the same distribution as $V_j(t)$ due to the "modular arithmetic" in lines 3 through 5 in Algorithm 1. Therefore at cycle $k + 1$, values of all counters are still mutually independent and the distribution of each counter value is uniform over $\{0, 1, ..., 2^l - 1\}$.

8.2 Analytical Comparison of two bounds

In this section we study and compare the two different bounds on $\Pr[D_{s,t}]$ analytically. We will show that neither of them would dominate the other in the whole τ (defined as $t - s$) range. Given a fixed K value the bound from Theorem 2 is generally better better while for some τ values the bound from Theorem 5 is better.

The problem we solve in the rest of this section is to find out which τ values make the bound from Theorem 5 better than that from Theorem 2, i.e., $e^{\frac{-2(K+\mu\tau-2^{-l}\tau)^2}{\min\{N,\tau\}}} \geq e^{-\frac{a^2}{2}(1-\frac{c}{3})}$ given a

fixed K value. Notice that in the parameter mappings, X_j in Theorem 2 is exactly W_j in Theorem 5. This fact makes our comparison straightforward. Recall that when applying Theorem 4 to our context the strategy to set the value of a is that when $K+\mu\tau-2^{-l}\tau < a_0\sigma$, we set a to $\frac{K+\mu\tau-2^{-l}\tau}{\sigma}$; otherwise $a = a_0$. It is also easy to see that aforementioned $g(a)$ can also be viewed as a function h of $\frac{a}{\sigma}$, i.e., $h(y) = 4 - e^y - \frac{ye^y}{2}$. Clearly $\frac{a_0}{\sigma}$ is a root of $h(y) = 0$, denoted as C. Next we will study the above problem in these two different settings of a respectively:

Case (i) when $K + \mu\tau - 2^{-l}\tau < a_0\sigma$, i.e., $\sigma^2 > \frac{K+\mu\tau-2^{-l}\tau}{C}$. Therefore the problem becomes for which τ values

$$\frac{2(K+\mu\tau-2^{-l}\tau)^2}{\min\{N,\tau\}} \leq \frac{a^2}{2}(1-\frac{\epsilon}{3}) = \frac{(K+\mu\tau-2^{-l}\tau)^2}{2\sigma^2}(1-\frac{e^{\frac{K+\mu\tau-2^{-l}\tau}{\sigma^2}}-1}{3})$$

Using linear approximation of exponential function, the above inequality becomes

$$\frac{2}{\min\{N,\tau\}} \geq \frac{1}{2\sigma^2}(1-\frac{K+\mu\tau-2^{-l}\tau}{2\sigma^2})$$

Using χ to denote $\frac{1}{\sigma^2}$ we have the following function $F(\chi)$,

$$F(\chi) = \frac{K+\mu\tau-2^{-l}\tau}{6}\chi^2 - \frac{\chi}{2} + \frac{2}{\min\{N,\tau\}} \leq 0 \quad (1)$$

Solving the equation $F(\chi) = 0$, we can obtain its roots, i.e.,

$$\frac{\frac{3}{2} \pm 3\sqrt{\frac{1}{4} - \frac{4}{3\min\{N,\tau\}}(K+\mu\tau-2^{-l}\tau)}}{K+\mu\tau-2^{-l}\tau}$$

when $\frac{K}{\frac{3}{16}-\mu+2^{-l}} \leq \tau \leq \frac{\frac{3N}{16}-K}{\mu-2^{-l}}$; otherwise there is no (real number) root at all. Therefore in this case if $\frac{K}{\frac{3}{16}-\mu+2^{-l}} \leq \tau \leq \frac{\frac{3N}{16}-K}{\mu-2^{-l}}$, the bound from Theorem 5 is better when σ^2 is between the reciprocals of the two roots. Otherwise because $F(\chi)$ does not have any roots and $\frac{K+\mu\tau-2^{-l}\tau}{6}$ is larger than 0 in our context, $F(\chi) \geq 0$ is always false.

Considering the variance σ^2 shown in Theorem 3, we can easily find out the corresponding values of τ given the configuration of the related parameters. We use an example to clarify this further. In Figure 5 we set $N = 1,000,000$ SRAM counters and $l = 4$ bits for each. The queue size K is set to 600 and μ is set to $1/12$. We only plot the part where $\tau \geq \frac{K}{1-\mu}$ because τ at least needs to satisfy $K < b(s,t) - \mu\tau \leq \tau - \mu\tau$, otherwise there is no way that $D_{s,t}$ would happen. The solid curve represents variance (i.e., σ^2), the dashed and dashed-dotted curves represent the reciprocals of two roots of $F(\chi) = 0$ and the dotted curve represents $\frac{K+\mu\tau-2^{-l}\tau}{C}$. The merging points of the dashed curve and the dashed-dotted curve corresponds to $\tau = \frac{K}{\frac{3}{16}-\mu+2^{-l}}$ and $\frac{\frac{3\min\{N,\tau\}}{16}-K}{\mu-2^{-l}}$ respectively where $F(\chi)$ has only one root. We only need to focus on the segment of the solid curve beyond the dotted curve (i.e., $\sigma^2 > \frac{K+\mu\tau-2^{-l}\tau}{C}$). It is easy to locate the part of this segment that is between the dashed curve and the dashed-dotted curve (i.e., between the reciprocals of these two roots), which is the range of τ in which the bound from Theorem 5 is better than that from Theorem 2.

Case (ii) when $K + \mu\tau - 2^{-l}\tau \geq a_0\sigma$, i.e., $\sigma^2 \leq \frac{K+\mu\tau-2^{-l}\tau}{C}$.

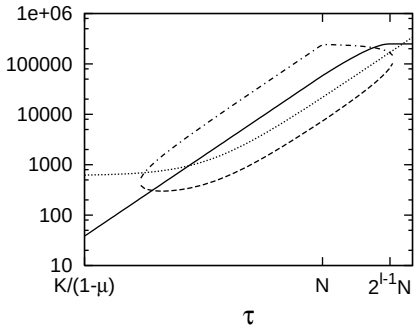


Figure 5: Case i by setting $N = 1,000,000$, $K = 600$, $l = 4$, $u = 1/12$.Notice that both x and y axis are logscale.

We know that $a = C\sigma$. So the problem becomes for which τ

$$\frac{2(K+\mu\tau-2^{-l}\tau)^2}{\min\{N,\tau\}} \leq \frac{a^2}{2}(1-\frac{\epsilon}{3}) = \frac{C^2\sigma^2}{2}(1-\frac{e^C-1}{3})$$

Thus we know that σ^2 has to fall in

$$\left[\frac{12(K+\mu\tau-2^{-l}\tau)^2}{\min\{N,\tau\}C^2(4-e^C)}, \frac{K+\mu\tau-2^{-l}\tau}{C}\right]$$

Again we plot Figure 6 for this case using the same parameter setting in Figure 5 to show an example. Again, the solid curve represents variance (i.e.,σ^2); the dotted and dashed-dotted curves represent the function $\frac{K+\mu\tau-2^{-l}\tau}{C}$ and $\frac{12(K+\mu\tau-2^{-l}\tau)^2}{\min\{N,\tau\}C^2(4-e^C)}$ respectively. We only need to focus on the segments (2 of them in Figure 6) of the solid curve that are below the dotted curve (i.e., $\sigma^2 \leq \frac{K+\mu\tau-2^{-l}\tau}{C}$). It is easy to locate one of these two segments that is above the dashed-dotted curve. This segment captures the range of τ in which the bound from Theorem 5 is better than that from Theorem 2.

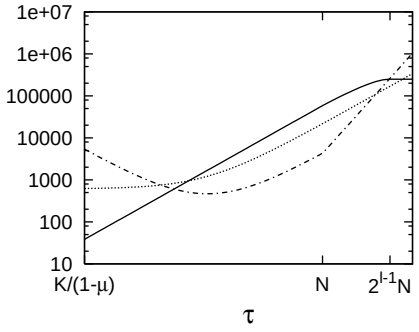


Figure 6: Case ii by setting $N = 1,000,000$, $K = 600$, $l = 4$, $u = 1/12$. Notice that both x and y axis are logscale.