Accuracy Bounds For The Scaled Bitmap Data Structure

Sumeet Singh, Cristian Estan, George Varghese, Stefan Savage
University of California, San Diego

This report describes the *scaled bitmap* data structure, that can accurately estimate the number of unique elements in a set (assuming the set has a continuously increasing number of elements). This data structure is particular motivated by the need to efficiently count the number of IP addresses, infected by a network worm during an epidemic.

The EarlyBird [3, 4] system provides a method to identify worm signatures from network traffic by using two key metrics, content prevalence and address dispersion. Prevalence can be defined as the frequency at which the signature (byte sequence) appears in the traffic stream and address dispersion is defined as the number of unique sources and destinations associated with the particular signature.

It is possible to count unique IP addresses using a simple list or hash table. This implementation provides exact results, and can be appropriate if there are very few potential worm signatures. However, more efficient solutions are needed if there are many pieces of content suspected of being generated by worms. One approach to significantly reduce the memory required is to approximate the result using a *direct bitmap* [5, 1] : hash each source to a bitmap, set the corresponding bit, and conclude that the number of sources exceeded the dispersion threshold T when the number of bits set exceeds a certain number. Thus, if T is 30, one can hash the source address into a bitmap of 32 bits and estimate that the number of sources crosses 30 if the number of bits set crosses 20 (the correction factor accounts for hash collisions). This small direct bitmap can provide an accurate indication of when the threshold is exceeded, but it cannot accurately estimate the number of addresses associated with the worm if their number is large.

Additionally, it can also be important to track the *progress* of an attack during an outbreak. So while an address dispersion of 30 may be sufficient to triger an alert, it is useful to know that in the next second there were 60 distinct sources, and so on. Other techniques such as probabilistic counting [2] and multiresolution bitmaps [1] can count up to high values, but require more memory. For example, a multiresolution bitmap requires 512 bits to accurately count a million distinct items. To reduce this requirement, we exploit the fact that a worm epidemic will produce strictly increasing address dispersion during its growth phase. Using this observation we devise a new, compact data structure for estimating address dispersion during outbreaks that we call a *scaled bitmap*.

1 Scaled Bitmap Data Structure

Similar to multiresolution bitmaps, the scaled bitmap design is based on subsampling a larger hash space. For example, to count up to 64 sources using 32 bits, one can hash sources into a space from 0 to 63 but only only set bits for values that hash between 0 and 31 – thus ignoring half of the sources. To compensate for this subsampling, we can scale the estimate by a factor of 2 at the end of a measurement interval. However, as the number of sources rises (due to rising infection levels) the bitmap will fill and underestimate the true count. To address this increase, the scaling factor can be dynamically increased to compensate as the bitmap fills. For example, in a subsequent configuration the bitmap might map only a quarter of a larger hash space and be scaled by a factor of 4.

However, after the bitmap is scaled to the next configuration, the count of addresses in the previous configuration is lost. To compensate for this, one could preserve the count from the previous configuration and add it to the current estimate. However, if sources counted in the previous configuration remain active, then this "correction" may introduce double counting. Thus, this algorithm is inherently biased: it will consistently estimate higher counts if the traffic mix contains sources that send packets in multiple measurement periods. In the worst case, if the same sources are active in every configuration, then the maximal correction factor (Lemma 1), can overestimate the true count by a factor of three (Lemma 2). We next show how to reduce this bias significantly by applying a more sophisticated algorithm.

The scaled bitmap shown in Figure 1 is a more accurate technique for counting distinct sources with only a modest increase in memory. The idea is to use multiple bitmaps ($numbmps = 3$ in this example) mapped to progressively smaller and smaller portions of the hash space. To compute the count, one estimates the number of sources hashing to each bitmap, sum the estimates and then multiply by the inverse of the fraction of the hash space covered by all the bitmaps in total. When the bitmap covering the largest portion of the hash space becomes too full (i.e., it has has too many bits set to provide an accurate estimate), we advance to the next configuration by recycling it: all its bits are cleared and it is mapped to the next slice of the hash space (Figure 2 and lines 7 to 13 in Figure 1).

Thus, each bitmap covers half the hash space covered by its predecessor. The first bitmap, covering the largest portion of the hash space, is the most important in computing the estimate, but the other bitmaps provide "memory" for counts that are still small and serve to minimize the previously mentioned biases. Consequently, not much correction is needed when these bitmaps become the most important.

Lemma 1 *If each source is active for only one configuration, using an additive correction of* $\frac{2(2^{base}-1)}{2^{numbmps}-1} b \ln\left(\frac{b}{b-max}\right)$ *provides an unbiased estimate of the number of unique sources.*

Proof First we show that we do not need to change the correction during the lifetime of a configuration and then bound by how much we need to increase the correction when we switch from one configuration to the next.

```
UpdateBitmap(IP)
1   code = Hash(IP)
2   level = CountLeadingZeroes(code)
3   bitcode = FirstBits(code << (level+1))
4   **if** (level $\geq$ $base$ and level $<$ $base$+$numbmps$)
5       SetBit(bitcode,bitmaps[level-$base$])
6       **if** (level $==$ $base$ and CountBitsSet(bitmaps[0]) $==$ $max$)
7           NextConfiguration()
8       **endif**
9   **endif**

ComputeEstimate($bitmaps$,$base$)
1   numIPs=0
2   **for** i= 0 **to** $numbmps$-1
3       numIPs=numIPs+$b\ln(b/$CountBitsNotSet(bitmaps[i])$)$
4   **endfor**
5   correction= $2(2^{base}-1)/(2^{numbmps}-1)\cdot\ b\ln(b/(b-max))$
6   **return** numIPs $\cdot 2^{base}/(1-2^{-numbmps})$+correction
```

Figure 1: A scaled bitmap uses $numbmps$ bitmaps of size b bits each. The bitmaps cover progressively smaller portions of the hash space. When the bitmap covering the largest portion of the hash space becomes too full to be accurate (the number of bits set reaches *max*), it is advanced to the next configuration by "recycling" the bitmap (see Figure 2). To compute an estimate of the number of distinct IP addresses, we multiply the estimated number of addresses mapped by the bitmaps by the inverse of the fraction of the hash space they cover. A correction is added to the result to account for the IP addresses that were active in earlier configurations, while the current bitmaps were not in use at their present levels.

Assume the correction is such that the estimate is unbiased when entering a configuration. Consequently, all addresses active during this configuration were not active earlier and they will hash to each bitmap with probability proportional to the size of the hash space they cover. Since the calculation for computing the number of addresses hashing to the bitmaps is itself unbiased[1], the resulting estimate is unbiased as well.

For the first configuration ($base = 0$), there is no need for correction because the estimate provided by the bitmaps is unbiased. Indeed, the correction formula evaluates to 0 for $base = 0$.

When advancing to the next configuration, the number of addresses hashing to the bitmap covering the largest portion of the hash space is expected to be $m = b\ln(b/(b-max))$. Let s be the sum of the estimates for the other bitmaps. Since the other bitmaps started operating no earlier than the first one, the number of addresses hashing to bitmap i is at most $m/2^i$ and thus $s \leq m(1-2^{-numbmps+1})$. Let c_{base} be the correction that gives an unbiased estimate. This means that $c_{base} + (m+s)2^{base}/(1-2^{-numbmps})$ is an unbiased estimate. Right after the switch of configurations, let the new correction for unbiased estimates be c_{base+1}. Since the bitmap is empty, it will not contribute to the result and therefore the estimate will be $c_{base+1} + s2^{base+1}/(1-2^{-numbmps})$. Thus $c_{base+1} - c_{base} = (m-s)2^{base}/(1-2^{-numbmps}) \geq m2^{base}2^{-numbmps+1}/(1-2^{-numbmps}) = m2^{base+1}/(2^{numbmps}-1)$. Consequently, the bias towards over counting is eliminated by incrementing the correction factor by $m2^{base+1}/(2^{numbmps}-1)$

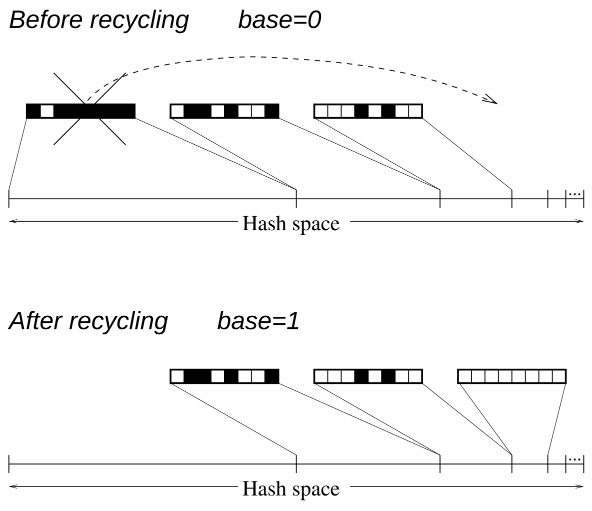

Figure 2: When the bitmap covering the largest portion of the hash range fills up, it is recycled. The bitmap is cleared and it is mapped to the largest uncovered portion of the hash space which is half the size of the portion covered by the bitmap rightmost before recycling. Recycling increments the variable $base$ (see Figure 1) by one.

at each switch of configurations. Adding these increments we obtain the correction from the lemma ■

Note that since $s = 0$ for $numbmps = 1$, the algorithm is never biased towards under counting for this case. Even though a traffic mix for which each source is active only in one configuration cannot bias the scaled bitmap towards over counting, others can. We next bound this bias.

Lemma 2 *The maximum ratio between the bias of the algorithm and the number of active addresses is* $2/(2^{numbmps} - 1)$.

Proof Once a configuration is in effect, new addresses are counted without bias. Since these new addresses increase the number of addresses without increasing the bias, they decrease the ratio between the bias and the number of addresses, so an adversary can maximize this ratio for any given configuration right after the configuration comes in effect, by not activating any new sources, and making all the old ones send one more packet so that the number of bits set in the bitmaps is maximized. Since we expect to advance to the next configuration when the bitmap covering a portion of $2^{-base-1}$ of the hash space has $m = b\ln(b/(b - max))$ addresses hashing to it, the adversary can force the scaled bitmap to expect to reach configuration number $base$ by activating $2^{base}m$ IP addresses, and no sooner. By reactivating all the addresses once we reach this configuration, the expectation of the estimate before the correction is $2^{base}m$ and thus the size of the bias is exactly the correction factor. The ratio between the two is $\frac{2(2^{base}-1)m/(2^{numbmps}-1)}{2^{base}m} = 2(1 - 2^{-base})/(2^{numbmps} - 1) < 2/(2^{numbmps} - 1)$ ■

2 Conclusion

In summary, in this paper we have presented the *scaled bitmap* data structure, that allows us to accurately estimate the number of unique elements in a set with continuously increasing number of elements. This data structure uses roughly 5 times less memory than previously known techniques such as probabilistic counting [2] and multiresolution bitmaps [1]. The *scaled bitmap* data structure has been effectively used in the EarlyBird system [4] to count the number of unique source and destination IP address associated with worm signatures.

References

[1] C. Estan, G. Varghese, and M. Fisk. Bitmap algorithms for counting active flows on high speed links. In *Proceedings of the ACM Internet Measurement Conference*, Oct. 2003.

[2] P. Flajolet and G. N. Martin. Probabilistic Counting Algorithms for Data Base Applications. *Journal of Computer and System Sciences*, 31(2):182–209, Oct. 1985.

[3] S. Singh, C. Estan, G. Varghese, and S. Savage. The EarlyBird System for Real-time Detection of Unknown Worms. Technical Report CS2003-0761, CSE Department, UCSD, Aug. 2003.

[4] S. Singh, C. Estan, G. Varghese, and S. Savage. Automated Worm Fingerprinting. In *Proceedings of the USENIX / ACM OSDI Conference*, Dec. 2004.

[5] K.-Y. Whang, B. T. Vander-Zanden, and H. M. Taylor. A Linear-Time Probabilistic Counting Algorithm for Database Applications. *ACM Transactions on Database Systems*, 15(2):208–229, 1990.