Impact of Packet Sampling on Count Min

Raghuveer Sharma Saripalli

Dept. of Computer & Information Science & Engineering, University of Florida

rsaripalli@ufl.edu

Abstract

In this paper, we are going to discuss how sampling effect of the CountMin flow sketch algorithm. In traditional CountMin Sketch, all the flows and their packets are recorded. This has a huge impact on the network accuracy and traffic because when more bits are set in the flow table there is more scope for noise. When introduced sampling into the CountMin it reduces the load on the network by decreasing the number of packets flowing through the routers and switch fabric. CountMin sampling introduces a variety of packets while retaining correctness, efficiency and enhancing network management.

Introduction

There is a great deal of sampling techniques used in myriad papers. Each paper defines its own way of sampling that can be utilized to the network. Sampling plays an important role in controlling network traffic. To decrease latency and delays, sampling lowers the packet analysis and burden of routers or switches. Noise reduction is also aided by sampling. The sketches were presented as a sample solution or substitute. It also reduces the number of packets flowing into the network which in turn reduces the latency caused by those extra packets.

An added benefit of sampling is it reduces the error rate. Sampling is better than sketches because the sketches face the same space constraint with increased flows. Along with the space, the additional problem due to increasing flows is the counters will be filled up fast and leads to accuracy being jeopardized and more errors. This paper performed a study of performances and discuss the various techniques and sketches of sampling and their differences.

One of the uses for Sampling is it can be used for monitoring the network and measuring the traffic. It could help reduce the rate at which sketches fill up and reach the target accuracy. It will also reduce bandwidth usage. The difficult part with Sampling is we have to decide upon the most suitable and appropriate rate of sampling. This rate is deciding factor for reaching the target accuracy and lower computation overhead. The relation between the sampling rate and accuracy is directly proportional and this is proven in the rest of the paper. Even though it is directly proportional we can’t allow it to be a very high rate because in that case is almost equal to the original CountMin problem. Now here in this paper, we would demonstrate the ideal sample rate which could help to reduce the overhead of calculations and improve the target accuracy.

We are going old school and using a simple sampling method, and this is used along with the CountMin algorithm for the flow recording. We perform sampling of packets using a pre-determined probability value. That means if the probability p is ¼ then there is a 25% chance for the received flow packet to be recorded and a 75% chance for the flow packet not to be recorded. We have conducted various experiments using CountMin using the original sampling method for different probability values. The results say about the error rate and the trends show that when packets dropped keep on increasing the accuracy change get stagnant.

Background

In this paper we have used the linear probability and CountMin sketch algorithm. The CountMin sketch required memory to record all the flow packets received. This linear probability technique acts a sieve to separate the recorded ones and non-recorded flow packets. Flow packets that are received are accepted with a probability P_f and declined with probability of $1-P_f$. Like every other sampling this one also produces error, but we can ignore for the greater good of reaching target accuracy. We ensure fairness in choosing a packet for every flow.

Let's say M_f be the number of packets in a single flow when it is retrieved from the CountMin sketch. Then the actual size will be $M_f * 1/P_f$. This is a very primitive operation with O(1) time complexity. Thus, it results in reduced traffic occurs in network and targeting higher accuracy rates with limited memory usage.

This model which we are using gives us 2 benefits: -

a. Improved network traffic monitoring accuracy.
b. Extensive measurement duration.

Problem

Until now we discussed that we could obtain a particular preset probability value will provide us with results we wanted. But we didn't find out that value yet. We have to run sampling with various values and find out the optimal value using the linear probability of sampling against the original CountMin Sketch Algorithm.

Design

In this section we are going to discuss about the linear sampling and how to scale the probability values. Follows the explanation of how CountMin is being used with the scaling of sampling. The implementation of CountMin sketch is done along with Sampling of probability values. We can see the flow diagram of the program in Figure 1.

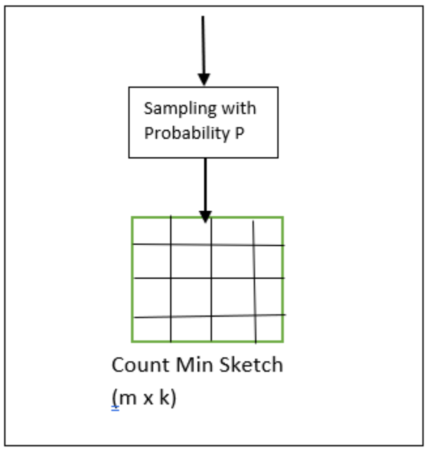
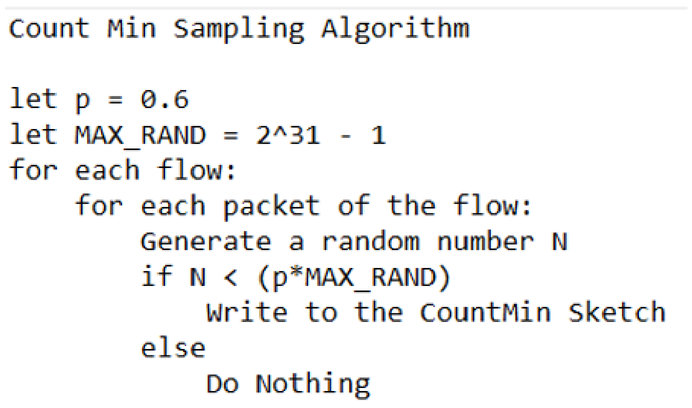


Figure 1



```
Count Min Sampling Algorithm

let p = 0.6
let MAX_RAND = 2^31 - 1
for each flow:
    for each packet of the flow:
        Generate a random number N
        if N < (p*MAX_RAND)
            Write to the CountMin Sketch
        else
            Do Nothing
```

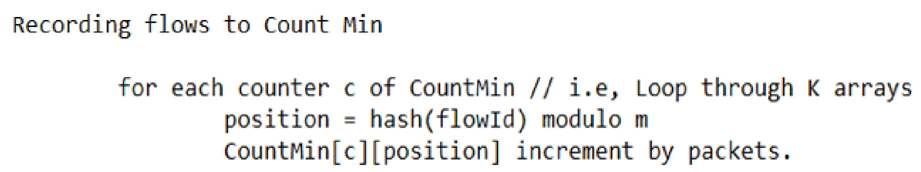
Figure 2

Algorithm for Sampling: -

This algorithm interprets every packet of all the flows. It decides whether a packet is to be considered or not by using the Probability P. When a packet is considered then it is recorded in the CountMin Sketch..

Algorithm for CountMin: -

The CountMin is a 2D data structure that holds flows data in m size lists in a linear fashion. There are k arrays and each of them with size m. Therefore, there is a total of m*k space. The flow when received is hashed using an efficient hash function and at that index, the value is increased per packet by 1. For each of K arrays, there will be K hashing functions. Each of them will be used to hash the flowId. And then incrementing per packet is done at every k array at the positions which we get after hashing the flowIds. Now let’s dive into the retrieval part of the algorithm. When querying data from the data structure we again use K hashing functions for each of the K arrays. When we got the respective indices at which the number of packets is present, we get the minimum of K values and assume it will be the correct number of packets received for this flow. Then we also apply additional steps and make it a sample scaling algorithm to convert this original CountMin Sketch into a probabilistic data structure. Figure 3 gives the algorithm for Count Min recording the flows.

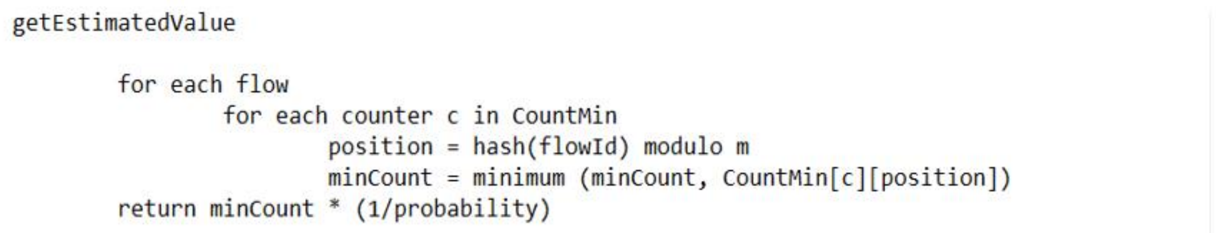


```
Recording flows to Count Min

        for each counter c of CountMin // i.e, Loop through K arrays
                position = hash(flowId) modulo m
                CountMin[c][position] increment by packets.
```

Figure 3

Algorithm for Sample Scaling: -

In this algorithm we introduce the probability and sampling part into our original CountMin algorithm. The way we record the flow will be the same. However, when we retrieve the count, the probability comes into picture because we didn’t record all the packets in a flow. So, we use probability to scale up the values. Let’s say the count we get when we query is C_f. When C_f is multiplied by 1/p then we get the scaled count which is N_f. Figure 4 shows the algorithm.



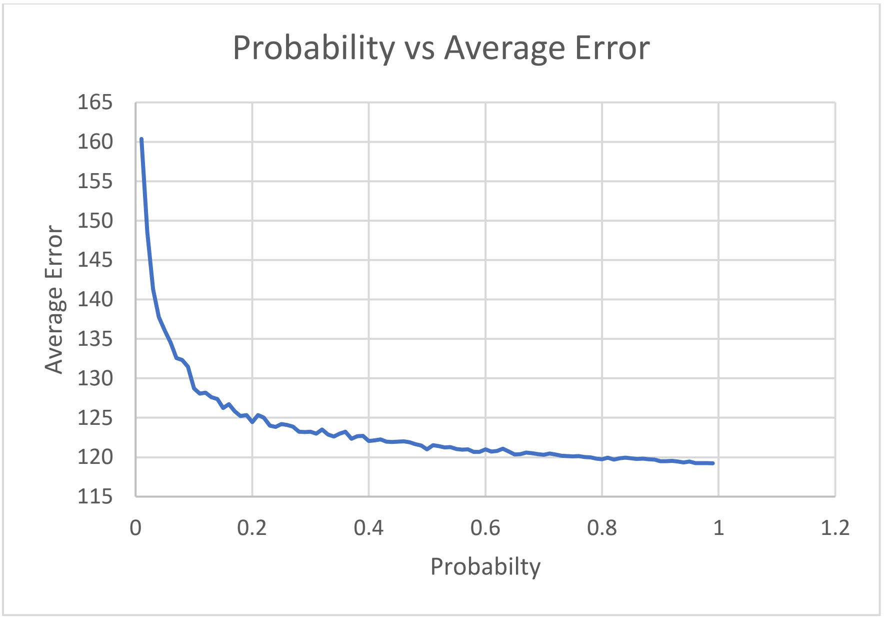
```
getEstimatedValue

        for each flow
                for each counter c in CountMin
                        position = hash(flowId) modulo m
                        minCount = minimum (minCount, CountMin[c][position])
        return minCount * (1/probability)
```

Figure 3

Analysis & Evaluation

Some performance measures were used to analyze the countmin with and without sampling.

We have determined the average error rate for various values of probability p. Later , for both with and without sampling of count min, a perflow difference has been discovered. Third, we plot a graph showing the relationship between the dropped packets with the countimin sampling probability. The tests were carried out on a data of 10K flows with a total packet count of 2622469.



We have plotted a graph plotting probability on the X-axis and on y-axis we have taken average error rate with a step increment of 0.01.
The average error = The total no of flows / (sum of all variations between the actual value of packets in the flow and the recorded packets for flows)

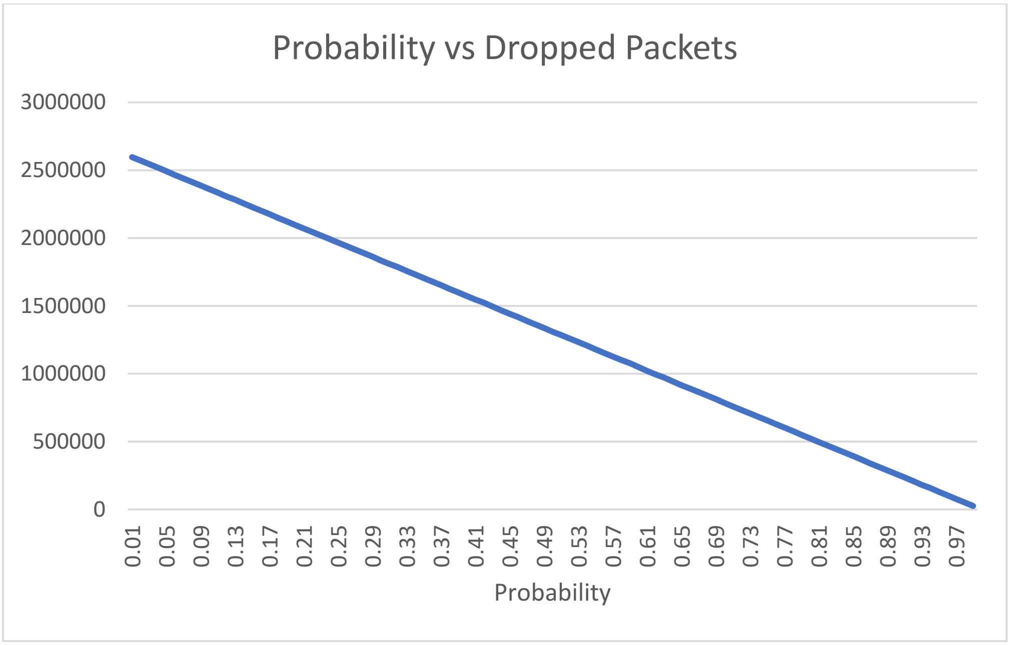
a) While increasing the probability, there is a rapid decrease in average error rate. This can be supported, because it is the trade-off that the study described previously: reducing data traffic by discarding packets while sacrificing accuracy. This behaviour is illustrated as when the MAX RAND value is multiplied by probability p, the MAX RAND value is smaller. MAX RAND value is taken as 2^31-1 in this experiment. As a result, the likelihood that the random number generated for each packet is smaller than MAX RAND multiple by p is quite low, and hence the error is considerable. When a packet is sampled in a concise manner, the equation summarizes as MAXRAND < p * RAND

B) we can observe that the error rate decreases as we increase the probability value.

C) Noise removal is another factor that contributes to increased accuracy. When a packet is captured, it is assigned we increment by one to the present value. Collisions become less likely as the number of packets decreases. However they will still be the collisions but they will occur less frequently, GVGFtherefore accuracy will be greater when the entire flow is not captured.

d) A detailed inspection reveals that the improvement in correctness with increasing the probability value after a certain point is not effective. There are always a significant number of packets discarded, which is not impacting accurateness.

The probability = 1 value indicates that sampling is done for all the packets



The graphic shows that the no of packet drops is greatly reduced by increasing the probability. When probability is less than one, error rate is also less.

Related Work

In this work, more extensive sampling was used, as well as improvements to the sketch to make it better. They utilised NetFlow, which includes a local database for tracking each flow's packets. They also employed sFlow, which will send the packet headers of captured packets to a server on a regular basis. They also introduced a new per-flow traditional sampling.

Conclusion

The sampling concept was introduced in the paper using Countmin. However, the same design can be used for different flow size counters. The study shows how we can reduce network traffic using sampling while maintaining low error rate. The study also shows that sampling slows the filling of the counter and reduces noise. Future research could employ a variety of probabilistic sampling techniques. Per-flow sampling is performed using varied probability value, considering the number of packets in a flow, can be performed. This may result in additional overheads, possibly requiring the development of new methods to reduce them. In order to increase performance we need to improve the countmin design or we have to research on optimised structures to maintain and analyse the flow sizes.

Counter trees, according to the article, are a cost-effective method to count the packet count for active flows that uses a 2-D counter sharing mechanism to ensure good memory economy. Flow spread, rather than flow size, can be monitored to provide additional insight into network traffic while maintaining sampling. However, because a standard sampling approach cannot detect flow spread, more research will be required. Finally, this paper states the groundwork for new ideas and additional study in the field of data sampling, which has previously received little attention.

References

1. Tune, Paul & Veitch, Darryl. (2011). Sampling vs sketching: An information theoretic comparison. 2105-2113. 10.1109/INFCOM.2011.5935020.

2. R. Jang, D. Min, S. Moon, D. Mohaisen and D. Nyang, "SketchFlow: Per-Flow Systematic Sampling Using Sketch Saturation Event," IEEE INFOCOM 2020 – IEEE Conference on Computer Communications, Toronto, ON, Canada, 2020, pp. 1339-1348, doi: 10.1109/INFOCOM41043.2020.9155252.

3. An Improved Data Stream Summary: The Count-Min Sketch and its Applications, Graham Cormode and S. Muthukrishnan

4. A. Kumar and J. Xu, "Sketch Guided Sampling - Using On-Line Estimates of Flow Size for Adaptive Data Collection," Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications, Barcelona, 2006, pp. 1-11, doi: 10.1109/INFOCOM.2006.326.

5. Chen, M., Chen, S., & Cai, Z. (2017). Counter Tree: A Scalable Counter Architecture for Per-Flow Traffic Measurement.