PACKET SAMPLING ON COUNTMIN

By - Kanika Sharma, UFID - 71191343

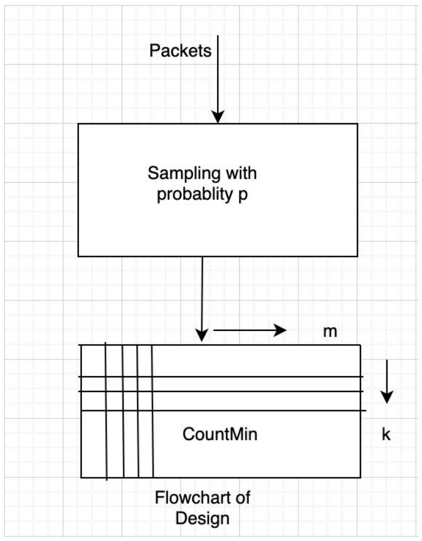
Abstract - Flow distributions are recorded as they are very useful for many reasons. Either sampling or sketches may be used for flow storage. However, storing all the data streams is expensive for network traffic and computationally. Applying the sampling technique gives the benefit of reducing overhead processing in the routers or any other systems. In a traditional countmin sketch, the packets in data streams are recorded without discarding any packet. This compromises network traffic management and accuracy as all the packets are recorded by setting bits of the counter to 1 or more value and results in more noise. Sampling for countmin introduces a selection of packets while still maintaining accuracy and improving traffic management.

The paper has been organized as - Introduction, Background, Design in Detail, Analysis and Evaluation, Related work, Conclusion, and Future Scope.

Introduction - The sampling technique has a major part in network traffic management. There are various sampling techniques and many papers which define their own sampling that can be applied on the network. Sampling reduces the additional packet processing and overhead of routers or switches to reduce latency and delays. Sampling also helps in noise reduction. The sketches were proposed as a solution or replacement for sampling. While the sketches serve the same purpose as sampling, they too suffer from limited space problems as the number of flows being recorded increases. The accuracy of sketches also gets compromised with a constant space, increasing flows and the counters fill quickly thus compromising accuracy and more errors. The study of performance and differences of sampling techniques and sketches have been performed in the paper[1]. Sampling is used for network monitoring and traffic measurement. It can also be used to help sketches get filled less quickly, record fewer packets, and still achieve desirable accuracy by reducing noise. It will also help in avoiding excess bandwidth usage. But the meticulous part here with sampling is setting up a viable sampling rate. The sampling rate decides the trade-off for accuracy and reduction of computational overhead. If the samples are collected more frequently i.e. sampling rate is high, accuracy is higher, and vice versa. But, if the sampling rate is kept higher or maximum it does not solve any purpose and is no good. The paper demonstrates that the desired sampling rate can be found which can reduce overhead without compromising accuracy. The sampling method used here is traditional sampling which has been combined with countmin sketch to record flows. The sampling of packets of each flow is done based on a preset probability. If probability p = ½ i.e 50% then there is a 50% chance that an incoming packet of a flow may be dropped or recorded. Similarly if probability p = ⅕ i.e. 20% the there is a 20% chance that the packet is recorded and an 80% chance that the packet is dropped. The experiment was conducted with countmin and traditional sampling, probability was varied for the same sized countmin sketch. The results were that the error rate decreases very slowly after a certain and though the number of packets dropped is more, it does not impact the accuracy significantly and is largely fair.

Background - The countmin sketch and traditional or linear sampling has been used in the paper. The countmin sketch is bounded by storage and has to store all the packets of all the flows. The linear sampling technique servers as a filter for packets and discards packets. A probability is chosen for the sampling tool. The packets are discarded with probability 1-p and accepted with probability p. The sampling does cause sampling error but it can be overlooked with the desired accuracy for a tradeoff of less network traffic. All the flows and each packet of each flow are treated fairly as probability remains the same throughout. While retrieving the flow size from countmin let Cf be the measure packet count then the actual count can be easily retrieved by Cf * 1/p. Thus actual flow size measurement can be done easily. Thus, with limited space and slowing the filling rate of countmin we can achieve higher accuracy while reducing network traffic at the same time. There are two major benefits reaped with this model. *a)* Increased accuracy of traffic measurement. *b)* Increased measurement period. The usage of countmin for calculating frequency in linear space already improves memory efficiently by storing only frequency and not value. With added linear sampling to it, the efficiency of the model increases furthermore. ***Problem*** - However, till now what value of probability for linear sampling will give the desired results has not been discovered. The analysis of different values of probability with a restricted memory size of the countmin sketch is yet to be evaluated and compared with a traditional countmin without sampling.

Image 1

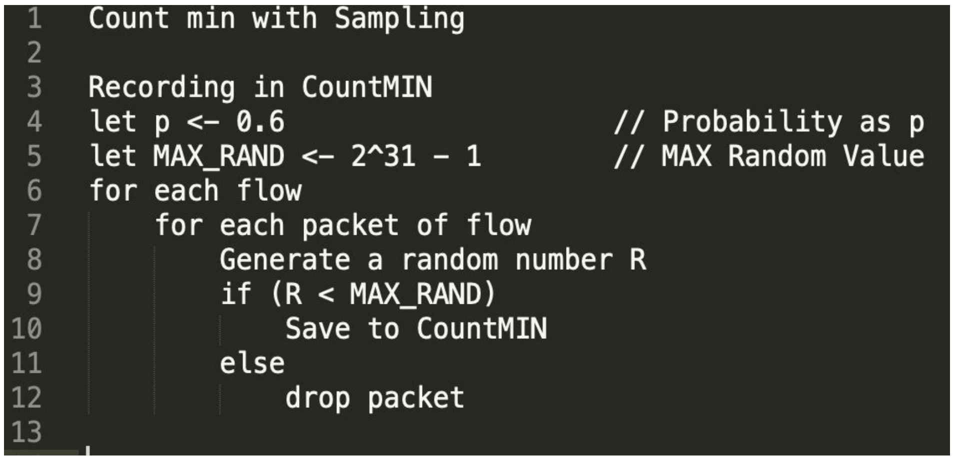


Design in Detail - This section looks into details of linear sampling algorithm and scaling sampling. Then the paper explains the countmin algorithm to be used with scaling sampling. The sampling and sketch are implemented together to evaluate performance and generate results. The code for this can be found at github[2]. The flow diagram can be seen in image 1.

Sampling Algorithm

The sampling Algorithm receives all the packets of all flows. It acts as a filter and discards packets with a probability $1-p$. The selected packets are recorded in a countmin sketch. The algorithm can be seen in image 2.

The algorithm for recording the packets has been explained ahead.



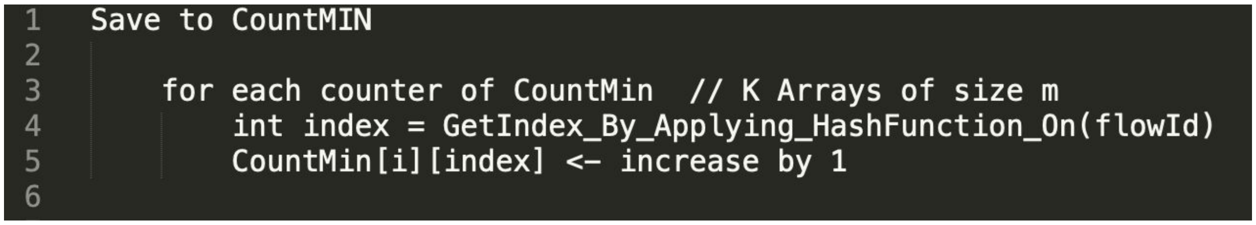
```
Count min with Sampling

Recording in CountMIN
let p <- 0.6                    // Probability as p
let MAX_RAND <- 2^31 - 1        // MAX Random Value
for each flow
    for each packet of flow
        Generate a random number R
        if (R < MAX_RAND)
            Save to CountMIN
        else
            drop packet
```

Image 2

Countmin Design and Algorithm

The countMin has been described in the paper[4]. The countmin is a probabilistic data structure that linearly stores the flows within the m size arrays. There are k arrays.

The total space available is m*k. Each flow is recorded in each of the k arrays by increasing the value at the position by 1. In order to find the position of where a flow is to be recorded, hashing is used. There are k hash functions. Each hash function records a flow in one array of size m respectively. The hash function is applied to the flowId and stored in the corresponding position. For retrieval, again all the hash functions are applied on all the flows and the minimum value recorded for each flow is chosen as the number of packets received to which the scaling algorithm mentioned ahead is applied. The algorithm for recording to countmin can be seen in image 3.

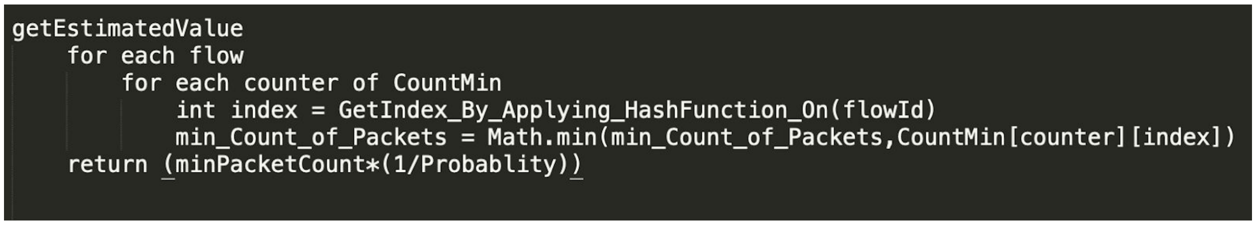


```
Save to CountMIN

    for each counter of CountMin  // K Arrays of size m
        int index = GetIndex_By_Applying_HashFunction_On(flowId)
        CountMin[i][index] <- increase by 1

```

Image 3

Scaling Sampling

The basic algorithm for spatial storage remains the same but the retrieval varies slightly as all the packets have not been stored. This requires scaling up. Only the packets passed from the sampling algorithm have been recorded in the sketch so multiplication of the packet count C_f by $1/p$ is required to get N_f which is the flow size. The algorithm can be seen in image 3 which is an addition to the algorithm of image 4.

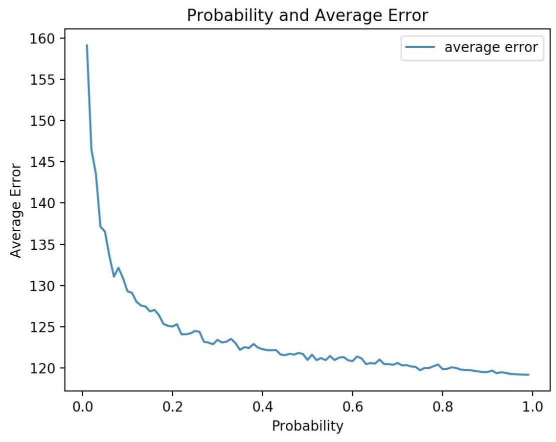


```
getEstimatedValue
    for each flow
        for each counter of CountMin
            int index = GetIndex_By_Applying_HashFunction_On(flowId)
            min_Count_of_Packets = Math.min(min_Count_of_Packets,CountMin[counter][index])
    return (minPacketCount*(1/Probablity))
```

Image 4

Analysis & Evaluation -

Image 5



In this section, some metrics have been used to evaluate the countmin with sampling. *First*, the Average error for the dataset[2] has been found for different values of probability p. *Second*, The perflow difference has been found for both countmin with sampling and countmin without sampling. *Third*, the graph of dropped packets with each probability for countmin with sampling has been plotted.

The experiments have been performed on a dataset of 10000 flows which have total count packets = 2622468.

A) For the graph of Average error and Probability, the graph has been plotted for Probability values on X-Axis and average error recorded on Y-Axis. The probability value starts from 0.01. The step increment in probability is 0.01. The average error is calculated by the sum of all differences of in the actual value of packets in the flow and recorded packets for flows divided by the total number of flows.

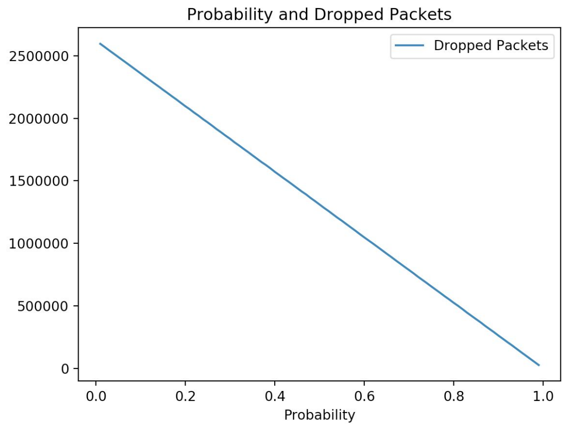
$$Avg\ Error\ =\ (\sum_{flowId=0}^{flowId=10000} |ActualValue - Recorded\ Value| * 1/dataset\ Size)$$

The sharp decrement in average error can be seen from lower probability values to a higher probability. This can be justified and is actually the trade-off that the paper discussed before lowering the network traffic by dropping packets and compromising with accuracy. This is because with lower values of probability the MAX_RAND value when multiplied with probability p is less. The MAX_RAND value used for the experiment is 2^31 - 1. Therefore, the chance that the random number that is generated for each packet is less than the MAX_RAND * probability value is quite less and hence the error is high. The equation ahead describes the situation when a packet is sampled in succinct. $RAND < p * MAXRAND$

b) When the value of probability increases the error decreases, indicating improved accuracy.

c) Another reason for improved accuracy is noise reduction. As a packet is recorded, it is stored as +1 to the current value present. With decreasing packets the possibility of collisions decreases. There will still be partial collisions but lesser, hence accuracy is still better eleven when the complete flow is not being recorded.

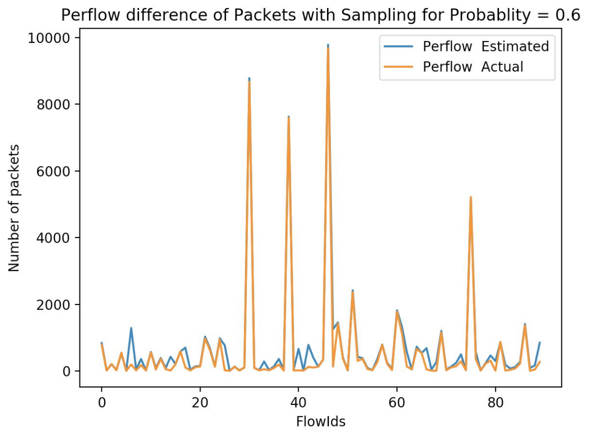
d) If a careful observation is made we see that with increasing probability after a certain probability value the gain in accuracy is not very high. There are still a significant amount of packets being dropped and improving network traffic management but not affecting the accuracy largely.



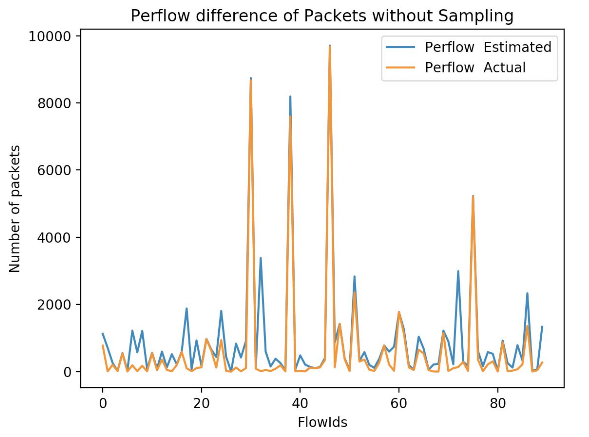
The value with probability = 1 means all the packets have been sampled and recorded on the counter.

The number of dropped packets can be seen in the image which decreases as the probability increases. Thus we can see the accuracy at probabilities a little less than 1 are still reasonably fairer.

B) To get more depth of how the sampling technique fairs with countmin, graphs have been plotted for a probability p = 0.6 and p = 1. A small section of 90 flowIds has been taken from a large set of 10000 flowIds dataset. The yellow line shows the actual packets in the flow. The blue line shows the estimated packets or the packets which have been stored in the sketch after sampling. On Y-Axis the number of packets has been plotted and on X-Axis the flow Ids have been plotted. For simplicity, the flowIds have been numbered from 0 to 10000 of which the top 90 have been chosen. The two graphs are for countmin with sampling with probability 0.6 and without sampling. The same dataset has been used for both the experiments with constant hash functions so that indexing remains the same for the scenarios. From the observation, it can be deduced that there is not much difference between the two flows. Even when the packets being collected is only 60% and 40% packets being dropped, the accuracy is very close for higher packet values of flows and small differences can be observed for lower packet counts of flows. Working on a better hash function to improve indexing can give better results and improve accuracy even more.



Related Work - The paper[3] has used more sophisticated sampling and has also made additions to the sketch for further improvement. They have used NetFlow which contains a local table for recording the packets of each flow. They also used a sFlow which periodically sends the packets headers of collected packets to a server on the network. They also introduce a new concept of per-flow systematic sampling.



Conclusion & Future Work - The paper introduces a notion of sampling with a sketch counter. The paper exclusively uses countmin. However, the same design is extensible for other counters that measure flow size. The paper also shows how sampling can reduce traffic and achieve accuracy. The paper also shows that sampling helps in reducing the speed with which the counter is filled and reduces noise. Future work can be using different types of sampling to achieve better trade-offs than what is achieved with probabilistic sampling. Per-flow sampling can be done with a different probability value, considering the number of packets that belong to a flow or a feedback mechanism can be devised. This might introduce additional overheads for which new mechanisms have to be explored to reduce overhead. Also, another route to improve performance can be introducing design changes to countmin or exploring other probabilistic data structures to record flow size. The paper[6] explores the use of counter trees: a scalable architecture for counting the number of packets for active flows, that achieve high memory efficiency with a 2-D counter sharing scheme. For getting more insight into network traffic flow spread can be measured instead of flow size while still using sampling. Though additional work has to be done for that as a traditional sampling technique can’t measure flow spread. In the conclusion, this paper lays the foundation for new techniques and more work in the direction of data sampling, where largely not many existing works have been done.

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. https://dsf.berkeley.edu/cs286/papers/countmin-latin2004.pdf
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.

Git Link to Code

6. https://github.com/skanika-git/CountFlowWithSampling