COMP90056 Assignment1 Report

Experimental Evaluation on Count-Min Sketch and Its Variants

Xiuge Chen 961392

1. Introduction

Many real-world applications, like IP traffic analysis and web crawls, generate massive data streams, these data and associated information arrive very rapidly and usually need a vast amount of space to store. However, often in reality, only limited space is available and the required time of update and query should be reasonably fast. Therefore, managing stream data requires compact data structure and fast, accurate update/query algorithm. Detailed motivation for these constraints is described by Estan [1], Henzinger [2], and Zhu [3] in different contexts.

Count-Min Sketch [4] (CMS) is a hashing-based, probabilistic, and lossy data representation that suits turnstile stream. In turnstile stream, data arrives in the form (O_k , $\Delta_{k\,t}$), where O_k is the identifier, Δ_k is the associated update at time *t*, the overall update of each item at any time should be no less than 0. CMS has many applications such as heavy hitters, range queries and point queries.

During the experiments, the theoretical and experimental estimation (space, time and accuracy) of CMS and two of its variants, conservative CMS and CMS with Morris Counter, are compared. Also evaluate these three sketches on different kinds of streams data that vary in size, item distribution and so on.

2. Theoretical Background

2.1 Count-Min Sketch

Count-Min Sketch (CMS) is a 2-dimensional array of counter with width *w* and depth *d*. For each row, a hash function is randomly chosen from 2-universal hash family, so the probability that 2 different objects will be hashed to the same value is no greater than 1/*v*. Thus, CMS is a compact data structure with *d* hash function and *wd* counters. Upon each update of object O_k , CMS will use each row's hash function to allocate and update the corresponding counter. And the query procedure is very similar, since each row maintains an estimation of O_k , where every estimation is no less than the true value (turnstile stream), so the CMS will simply return the minimum one out of all the estimations of O_k .

After the carefully analyze showed in the subject notes [5], The CMS has the following guarantee about its estimation if we choose $w = 2/\varepsilon$ and $d = log(1/\delta) + log(n)$:

$$f_y \leq \text{estimated } ef_y \leq f_y + \varepsilon F_1, \text{ with global confidence of at least } \delta$$

where f_y is the correct final value, F_1 is the sum of all objects' final values, *n* is the universal size. Therefore, the theoretical space and time complexity of CMS could be calculated as follows. If the 2-universal hash function is constructed as shown in the subject note[5], each hash function will maintain 3 values: *a*, *b* and *p*, that are all smaller than *2n*. To store such hash function, O(*log a +log b+log p*)=O(*3log 2n*)=O(*log n*) bits are needed. For each counter, O(*log* F_1) bits are needed since F_1 is the largest possible total value. So the overall space complexity of CMS is O($d \log n + dw \log F_1$) = O(($\log \delta^{-1}$ + *log n*) (*log n* + $\varepsilon^{-1} log\ F1$)). For each update/query, CMS need to loop through each row, calculate hash value and get corresponding counter values, if assume the hash calculation and minimum computation take constant O(1) time, the time complexity for update/query will be O(*d*) = O($\log \delta^{-1}$ + *log n*).

2.2 Conservative Count-Min Sketch

Conservatice Count-Min Sketch (CCMS) is very similar to CMS, except it changes the update procedure as follows. For each update *c* of O_k , first get the current estimation efk of Ok , then update the corresponding counter *C[i][j]* with max(ef_k + c, *C[i][j]*).

Since only the update procedure is changed in CCMS, the space and query time complexity of CCMS will be the same as CMS. As update complexity, during each update CCMS will first query the current

estimation, so the update complexity becomes O(d + d) = O(2($log\ \delta^{-1}$ + $log\ n$))=O($log\ \delta^{-1}$ + $log\ n$) if the comparison cost is constant.

CCMS still ensures the estimated ef_y is at least f_y because of the same reason. Moreover, usually the error rate ε will be not worse than CMS, since the max comparison prevent the counter value from increasing if the current estimation is smaller. It is like compensation mechanism, that constraints the counter value to be the greatest estimated value of one object. Indeed, it can improve accuracy "up to an order of magnitude" [6]. However, it does not work quite well with lots of negative updates, since the max comparison will discard negative updates and keep the original value. Therefore, in insertion-only stream, CCMS usually outperforms CMS because of the compensation mechanism, especially when collisions happen a lot. However, if the stream is both turnstile and dynamic, CCMS expects to be worse than CMS since it can not deal with negative updates.

2.3 Count-Min Sketch with Morris Counter

Another variant of CMS is CMS with Morris Counter (CMS-MC), the update and query procedure remain the same in CMS-MC, every primitive data counter is replaced by a Morris Counter.

Morris Counter (MC) is a memory-efficient counter commonly used to estimate large counts. Instead of storing n, MC stores $z = log(n)$, so that only $log(log\ n)$ bits is used. Everytime n increments by 1, MC will increment z with probability $1/2^z$, and return 2^z-1 when querying. As shown in subject notes[5], a single MC has expectation that $E[Y_n] = n+1$, where Y_n is the result after n update, so on average MC will give a good estimation. But the variance of MC is proportional to n, which is a problem in CMS-MC.

Assume querying of MC (compute 2^z) could be done in constant time, then the query time of CMS-MC stays the same with CMS. However, the update time cost is slightly different, for each positive update c, CMS-MC will increment every related MC c times, each MC could be incremented at most F_1 times (extreme case). Thus, the update complexity of CMS-MC is O($d\ F_1$) = O(F_1 ($log\ \delta^{-1}$ + $log\ n$)). Moreover, the space complexity is being reduced to O($d\ log\ n$ + $dw\ log\ (log\ F_1)$) = O($d\ log\ n$ + $dw\ log\ (log\ F_1)$) = O(($log\ \delta^{-1}$ + $log\ n$) ($log\ n$ + $\varepsilon^{-1} log\ (log\ F_1)$)) since CMS-MC only use $log(log\ F_1)$ bits to store F_1 .

CMS-MC does not guarantee the estimated ef_y is at least f_y , since it underestimates the actual count. Also, since the query in CMS will return the minimum estimation, CMS-MC will return an underestimation if one of the MCs returns an underestimation. Besides, CMS-MC can only deal with positive updates since MC only allow positive increment, so that CMS-MC will perform badly on lots of negative updates. However, CMS-MC will not give a bad underestimation all the time, if the final values f_y are roughly the same, and collision happened frequently, CMS-MC might also be able to obtain good estimated results by summing up underestimation. In all, CMS-MC is also not good for streams are both turnstile and dynamic, and it often outputs underestimations, so its overall accuracy might not be as good as CMS.

	Default CMS	Conservative CMS	CMS with Morris Counter
Memory/Space	O((log δ^{-1} + log n) (log n + ε^{-1}log F1))	O((log δ^{-1} + log n)(log n + ε^{-1}log F1))	O(($log\ \delta^{-1}$ + $log\ n$) ($log\ n$ + $\varepsilon^{-1} log\ (log\ F_1)$))
Update Time	O(log δ^{-1} + log n)	O(2(log δ^{-1} + log n)) = O(log δ^{-1} + log n)	O(F_1 ($log\ \delta^{-1}$ + $log\ n$))
Query Time	O(log δ^{-1} + log n)	O(log δ^{-1} + log n)	O(log δ^{-1} + log n)

Table 2-1: Theoretical comparison among Count-Min Sketch and its variants

3. Implementation

Java implementation is chosen here for the following reasons. Firstly, a scaffold of java is provided, which is easier to extend. Secondly, many great java libraries, such as *Commons Math* [7], are very helpful to generate and load artificial stream data. Moreover, in order to monitor the actual space usage, software

like *VisualVM* [8] could be used to analyze the memory usage (heap allocation) of Java. Lastly, although implementation over C or C++ might speed up the overall performance, the main focus here is comparing the relative performance of CMS and its variants, a Java implementation is pretty enough.

3.1 Hash Function

The implementation of 2-universal hash function in subject notes [5] is provided, the only change being made is to use the absolute value of hashcode since it sometimes will return negative value.

3.2 Count-Min Sketch

Besides the provided implementation, the data type of counter is changed from int to long primitive, since the F_1 value of large universe and long stream could possibly exceed the maximum value of int (2^{31}-1).

3.3 Conservative Count-Min Sketch

Implementation of CCMS is similar to CMS, except the update procedure is changed as described in 2.2 .

3.4 Count-Min Sketch with Morris Counter

Implementation of MC is also provided, to utilize the advantages of MC, the counter *z* is changed from int to byte primitive. Also, estimation 2^z will be calculated dynamically rather than stored in class. As the CMS side, all operations associated with counter is replaced by MC increment and query function.

4. Experimental Preparation and Settings

Since space, time and accuracy are all key attributes of streaming algorithm [5], a good algorithm should be compact, fast and often returns good estimation. Therefore, in this experiment, sketches are also evaluated by these three aspects. All items are drawn from a universe U=[1, 2^{31}-1], let the error rate ε be 0.01 and the bad probability δ be 0.001. Although there are many applications of CMS, point queries is chosen here since it is easier to perform and could still evaluate the differences among sketches.

4.1 Space Usage Comparison

To monitor the actual memory usages, *VisualVM* [8] is used. Also, a theoretical space usage of using array to record every object is estimated for comparison.

4.2 Time Usage Comparison

In this experiment, the start and end time (in nanoseconds) of different methods is recorded, and their average time difference is used as a rough estimation of time complexity.

4.3 Error Rate Comparison

To accurately calculate the error rate ε, the correct results are kept during data generation, so the error rate ε_i of each object O_i could be calculated as:

$$\varepsilon_i = \frac{|ef_i - f_i|}{F_1}$$

where ef_i is the estimation, f_i is the actual result, F_1 is the total f_i . Since these sketches return good estimations with some probability, the average error will be influenced a lot by outliers. Cormode [9] used weighted average and (1 - δ) percentile error (only query each item once) to evaluate point queries, which are adopted in this experiment as well.

4.4 Stream Setting

As mentioned above, this experiment uses artificial generated data to represent real-world streams. So, the experimental data is generated with following parameters consideration.

4.4.1 Stream Length

Real-world Stream varies in size, its length determines the number of distinct items and updates, thus could influence the estimation of sketches. In this experiment, streams with length 10^3, 10^4, 10^5, 10^6, 10^7 are generated to evaluate the impact of stream length on sketches.

4.4.2 Stream Object Distribution

Although all items are drawn from the same universe *U*, different distributions will have a huge impact on sketches. Since most real-world streams are skewed and could be well captured by Zipf distribution with appropriate parameter $s \geq 0$, for example, web page accesses number has *s* between 0.65 and 0.8 [10], transmission times of internet file has *s* approximately 1.2 [9]. Therefore, items from Zipf distribution with parameter 0.4, 0.8, 1.2 and 1.6 is generated. Meanwhile, in order to study different distribution and compare them, normal distribution with different standard deviation (10^4, 10^6, 10^8) and uniform distribution with different universal size (10^4, 10^6, 10^8) are also created.

4.4.3 Stream Update Data Type

Real-world streams also vary in update types. Some updates will always be positive (insertion-only), while others might contain negative update (turnstile). Also, among those streams contain negative updates, some results might not grow with stream length (kind of like dynamic). Consequently, three different update types are considered, positive-only update with range [0,100] (insertion only), negative-allowed update with range [-100,100] and constant final result (dynamic plus turnstile), and negative-allowed update with range [-100,100] and results proportional to stream length (general turnstile).

5. Result and Discussion

5.1 Space Usage Comparison

Since the space complexity for each sketch is consistent as the universe size, error rate and bad probability are fixed, all results obtained from different experiments is the same as displayed in Table 5-1. (here memory usage of data-irrelevant filed such as *ClassLoader* is ignored)

Memory / bits	Default CMS	Conservative CMS	CMS with Morris Counter	Array (estimated)
Hash Table	3936	3936	3936	-
Counter	524800	524800	65600	137438953472

Table 5-1: Space usage of different sketches

As shown in Table 5-1, these three sketches are much more compact than array, CMS-MS is even more compact than the other two since it only use *log(log n)* bits to store F_1. The experimental results is aligned with the theoretical analysis in section 2.

5.2 Time Usage Comparison

The average time used by the query and update methods from different sketches is listed in Table 5-2.

Time / Nanoseconds	Default CMS	Conservative CMS	CMS with Morris Counter
Update	805.107365	1524.302566	75538.776964
Query	925.550449	842.385924	1865.810601

Table 5-2: Average time cost of different sketches

From Table 5-2, CCMS's query time is nearly the same as CMS's update/query time, and the update time of CCMS is roughly twice as much as them, since CCMS requires an extra query before every update. Also the update time of CMS-MC is much higher since it might increment each MC at most F_1 times. These results are roughly aligned with the theoretical analysis. However, the query time of CMS-MC is higher than expected, it is because the theoretical analysis "underestimate" the time cost of computing 2^z. Although it takes indeed constant time, it still needs more time than just changing the value of a primitive counter, results in the increase of time cost.

5.3 Error Rate Comparison

Since the normal distribution with std 10^6 and uniform distribution with size 10^6 are quite similar to normal distribution with std 10^8 and uniform distribution with size 10^8, they are discarded for simplicity. Zipf distribution with s=1.6 is removed from (2,3,5,6,9,12,14,15,17,18) since their value is too large (>0.05) to draw on current scale.

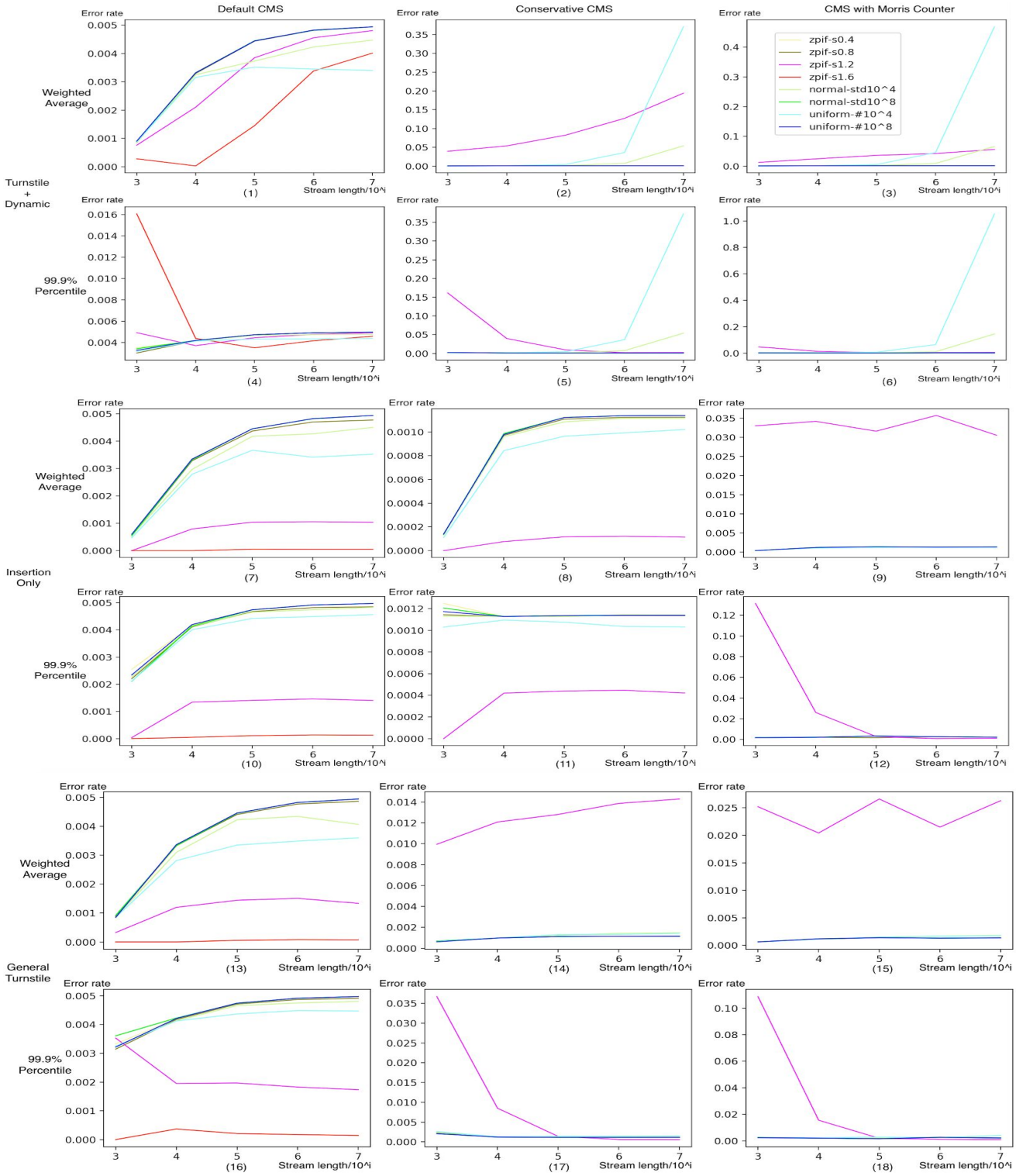

Figure 5-3: Error Rate of sketches on different stream length, distribution and update type

As shown in Figure 5-3, as the stream length grows, most of weighted average and 99.9% percentile errors either slightly increase or fluctuate around some values. Few stream might experience percentile error decreasing when just starts increasing the stream length, because the distinct data amount ($<10^3$) then is too small so that bad estimation is more likely to be included into 99.9% percentile errors.

CMS achieves the error guarantee on all kinds of streams and usually the error rate is just half of the error constraint. It is not surprising that CMS performs quite well on highly-skewed data, such as Zipf distribution with s 1.2(purple line) and 1.6(red line). Also, normal and uniform distribution data with small std(10^4) and universe size(10^4) have smaller error than other non-highly-skewed streams. It is because the number of distinct items in highly skewed and these two kinds of stream is much smaller than others, so that collisions are less likely to happen, CMS are more likely to obtain good estimation. Lastly, CMS does not vary a lot when using different stream update type, when the stream is both turnstile and dynamic (constant result), CMS performs slightly worse than insertion-only and general turnstile stream.

CCMS only keeps the error guarantee on insertion-only stream, especially when the stream is highly-skewed, on insertion-only stream CCMS could reduce the error rate to approximately ⅛ of the error rate using CMS. However, unlike CMS, CCMS does not suitable for all kinds of highly-skewed stream, mainly because it can not deal with negative updates, therefore CCMS performs badly on skewed stream with negative updates. But the error rate of other distributions (slightly skewed, normal, uniform) on turnstile stream is actually better when using CCMS rather than CMS, because the items do not appear as often as skewed data, so estimation could still benefits from the “compensation mechanism” of CCMS and won’t be disturbed by lots of negative updates.

CMS-MC does not have the error guarantee and performs badly on basically all kinds of highly-skewed turnstile streams no matter negative updates are allowed or not, since it not only neglect negative updates, but also might output underestimations. Meanwhile, unlike CCMS, CMS-MC is not be able to accurately estimate streams with small amount of distinct items, since the variance of MC is proportional to item value, frequently appeared items tend to have larger variance in MC. On the Contrary, CMS-MC could give a reasonably good estimation on non-skewed stream data with large amount of distinct items, since each item only has few updates, the variance of each MC then is fairly low and the influence of negative updates is negligible.

6. Conclusion

Count-Min Sketch and its variants, Conservative Count-Min Sketch and Count-Min Sketch with Morris Counter, are all very compact data structure that supports fast querying and updating. CMS-MC replaces primitive counters with Morris Counter, therefore is more compact than the other two sketches. Meanwhile, the query and update time of CMS and CCMS, together with the update time of CMS-MC, are roughly aligned with their theoretical analysis. That is, CCMS and CMS-MC requires more time on updating since the previous one adds an extra query before every update, and the later one will increment each MC at most F_1 times. However, although theoretically the query speed of CMS-MC should be the same as the query speed of the other two, it is slower in reality because the 2^z computation introduce extra constant time cost.

As the stream size increases, the error rate either slightly grows or fluctuates around some values. CMS performs quite well on skewed data and slightly better on data with less distinct items, also it is very stable as its performances does not vary from stream type. CCMS achieves even better accuracy over insertion-only stream, but when negative updates is allowed, CCMS will perform badly on skewed stream since it will ignore lots of negative updates. Similarly, CMS-MC can not handle negative updates and will underestimate the value, so generally it gives worse estimation than the other two sketches on skewed stream. However, over the none or slightly skewed stream, the two variants of CMS performs even slightly better than CMS, since the number of distinct items is very large and items appear infrequently, so the impact of negative updates is negligible, meanwhile the sketches could benefit from the low variance of MC and “compensation mechanism” of CCMS.

7. References

[1] Estan, C. and Varghese, G., 2003, June. Data streaming in computer networks. In *Proceedings of Workshop on Management and Processing of Data Streams*.

[2] Henzinger, M.R., Raghavan, P. and Rajagopalan, S., 1998. Computing on data streams. *External memory algorithms*, *50*, pp.107-118.

[3] Zhu, Y. and Shasha, D., 2002, January. Statstream: Statistical monitoring of thousands of data streams in real time. In *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases* (pp. 358-369). Morgan Kaufmann.

[4] Cormode, G. and Muthukrishnan, S., 2005. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, *55*(1), pp.58-75.

[5] Holland, W. and Wirth, T. (n.d.). *Lecture Notes Weeks 1-7*. [online] Melbourne: University of Melbourne. Available at: https://app.lms.unimelb.edu.au/bbcswebdav/pid-7745980-dt-content-rid-67197046_2/courses/COMP90056_2019_SM2/90056-lnotes_w1-w7_v2.pdf [Accessed 14 Sep. 2019].

[6] Estan, C. and Varghese, G., 2002. *New directions in traffic measurement and accounting* (Vol. 32, No. 4, pp. 323-336). ACM.

[7] Commons Math. (2016). Apache Commons.

[8] Sedlacek, J. and Hurka, T. (2019). *VisualVM*. Oracle.

[9] Cormode, G. and Muthukrishnan, S., 2005, April. Summarizing and mining skewed data streams. In *Proceedings of the 2005 SIAM International Conference on Data Mining* (pp. 44-55). Society for Industrial and Applied Mathematics.

[10] Breslau, L., Cao, P., Fan, L., Phillips, G. and Shenker, S., 1999, March. Web caching and Zipf-like distributions: Evidence and implications. In *Ieee Infocom* (Vol. 1, No. 1, pp. 126-134). INSTITUTE OF ELECTRICAL ENGINEERS INC (IEEE).