DS-Means: Distributed Data Stream Clustering

Alessio Guerrieri and Alberto Montresor *

University of Trento, Italy

Abstract. This paper proposes DS-MEANS, a novel algorithm for clustering distributed data streams. Given a network of computing nodes, each of them receiving its share of a distributed data stream, our goal is to obtain a common clustering under the following restrictions (i) the number of clusters is not known in advance and (ii) nodes are not allowed to share single points of their datasets, but only aggregate information. A motivating example for DS-MEANS is the decentralized detection of botnets, where a collection of independent ISPs may want to detect common threats, but are unwilling to share their precious users' data. In DS-MEANS, nodes execute a distributed version of K-MEANS on each chunk of data they receive to provide a compact representation of the data of the entire network. Later, X-MEANS is executed on this representation to obtain an estimate of the number of clusters. A number of experiments on both synthetic and real-life datasets show that our algorithm is precise, efficient and robust.

1 Introduction

Broadly speaking, *clustering* is the problem of partitioning a set of items into a collection of *clusters*, so that, given a definition of similarity, items in the same cluster are more similar to each other than they are to items in other clusters.

Most clustering algorithms assume that the items to be analyzed are available *here* and *now*, meaning that the entire data set could be easily accessed from a single machine. Sometimes both these assumptions must be relaxed, meaning that items are inherently distributed and not immediately available, for example because they are continuously generated and volatile in nature.

As a potential application area, consider the problem of detecting malicious threats like botnets, DDoS attacks, viruses, etc. [3]. In this setting, a large number of detectors, potentially belonging to different organizations (ISPs, companies, universities) collect large quantity of information about the behavior of a system (for example, by inspecting the networking traffic at routers). In order to associate detection with a (potentially immediate) reaction, data should be analyzed and clustered as it flows through the detectors. In such setting, centralized algorithms cannot be applied. There are several reason for this:

– The data to be analyzed is constituted by a continuous stream of items, and waiting for all of them to be collected in a single machine is infeasible;

* This work is supported by the Italian MIUR Project *Autonomous Security*, sponsored by the PRIN 2008 Programme.

– Forwarding all data to one machine may be too expensive, inducing a large amount of unnecessary traffic;
– If the data is collected by different organizations, privacy issues may prohibit the gathering of the entire data set by a centralized third party.

Based on these considerations, the problem we are trying to solve is a distributed form of data stream clustering [1], where data arrives continuously at multiple nodes, without having the possibility to transmit the entire dataset to a single machine.

The main contribution of this paper is the DS-MEANS algorithm (where DS stands for Distributed Streams), a combination of various known techniques to solve this problem, without the need of previous knowledge about the number of clusters. DS-MEANS first partitions – at each node – data into chunks; then a distributed version of K-MEANS is applied on these chunks. Each time the distributed K-MEANS algorithm is executed, it returns K centroids (points equal to the average of a cluster) that are used, together with centroids obtained from previous executions, as a compact representation of the entire set of streams. Each node then locally runs X-MEANS, a clustering algorithm able to choose the number of clusters, on this representation. An aggregation protocol is executed by all nodes to reach an agreement on the number G of clusters; finally, each node runs a centralized instance of K-MEANS with $K = G$ to get the final clustering.

This approach has two important properties. Given that DS-MEANS works independently on each chunk of data, we can use it in an online setting (in which we want to look only at the last chunks of data) without the need of restarting it from scratch each time new data arrives. Also, the nodes do not directly exchange information about the single points they are trying to cluster, thus achieving a reasonable level of privacy in the presence of sensitive data.

The paper shows the good behavior of the protocol using synthetic datasets. DS-MEANS is able to reach high precision even when it starts without knowing the exact number of clusters; the amount of computation is similar to the centralized version; and finally, DS-MEANS is fault-tolerant and reaches the same level of precision with or without failures (although the computation time may increase in the presence of failures).

2 Problem Statement

We consider a distributed network of N computing nodes $p_1, \ldots, p_N$, each of them independently receiving a (possibly unbounded) stream of data items. Although the amount of points each node receives may be different, data items are homogeneous: each of them is a *point* in the same d-dimensional space $\mathbf{R}^d$, taken from a single distribution. Similarity of two points p, q is measured based on their Euclidean distance $d(p, q)$ in $\mathbf{R}^d$: the smaller their distance, the more similar two points are. We use $\mathcal{P}$ to denote the set of points received by all nodes.

Nodes may fail by crashing, meaning that they stop executing the protocol; we do not consider malicious failures, where nodes behave arbitrarily (for example spoofing other nodes with incorrect information about the points they

receive). In other words, nodes participating in the computation are trusted. Nodes are partially synchronized, e.g. through NTP.

Each node may reliably communicate with any other node; to claim a limited degree of privacy, we require that only aggregate information can be shared among nodes, without communicating individual points.

The output of our problem is a collection of G *centroids* $c_1, \ldots, c_G$, again taken from $\mathbf{R}^d$, such that (i) each centroid c_i represents a cluster $C_i \subseteq \mathcal{P}$; (ii) each point $p \in \mathcal{P}$ is assigned to the cluster represented by the closest centroid, denoted $c(p)$; (iii) the average distance

$$\frac{1}{|\mathcal{P}|} \sum_{p \in \mathcal{P}} d(p, c(p))$$

between each point and the center of its cluster is minimized.

The value of G is not known in advance; instead, the correct value must be identified based on the *Bayesian information criterion* [10]. This measure uses the log-likelihood of the dataset according to the model and its number of parameters (in our case the number of centroids multiplied by the number of dimensions) to compute the following metric:

$$\mathrm{BIC}(j) = \log M_j(S) - \frac{1}{2} k_j \log n$$

In this formula j is a model, $M_j(S)$ is the maximal likelihood of dataset S using model j, k_j is the number of parameters of the model and $n = |S|$. To compute the maximal likelihood of the dataset we used the identical spherical Gaussian assumption (see [9] for the exact formulas).

3 Background

Clustering a data set is one of the classical problems in the area of machine learning. This section provides a brief review of the clustering algorithms that are used as building blocks for our own solution.

3.1 K-Means

The most commonly known clustering algorithm is K-MEANS, based on a paper by LLoyd [6]. K-MEANS works as follows: it starts from K centroids (points representing the center of a cluster), repeatedly assigns each point to the closest centroid and recomputes the position of the centroids according to the points assigned to it. This algorithm is guaranteed to converge to a local minima of the *within-cluster sum of squares* (i.e. the average squared distance between each point and the center of its cluster), but the quality is highly dependent on the choice of the starting centroids.

Another drawback is that K needs to be initialized to the correct value of G, the actual number of clusters present in the system. If K-MEANS is given the wrong K (because such value is not known in advance), the algorithm fails to answer precisely.

3.2 X-Means

One algorithm that is able to compute a good clustering even without knowing the number of clusters is X-MEANS [9]. This algorithm uses 2-Means (K-MEANS with $K = 2$) as a subroutine and continues to divide the data set in smaller subsets using the Bayesian information criterion to decide if (and where) does the data need more clusters.

The algorithm will then return the clustering which scores the highest value based on the Bayesian information criterion. We use this algorithm mostly as a subroutine to compute the number of clusters to be created.

3.3 Distributed K-Means

The main building block of our algorithm is the distributed K-MEANS algorithm presented by Bandyopadhyay et al. [1], created to cluster datasets in P2P environments. The algorithm works as follows: each iteration of the K-MEANS algorithm is executed at the same time by all nodes in the network. Each node starts with the same centroids, updates them using its own data and computes an average between its own centroids and the centroids computed by some of its neighbors. This process is repeated until a steady state is reached.

Looking more closely to the algorithm, it appears clearly that it is based on a simple relaxation of the averaging step of K-MEANS. All nodes start from the same centroids and are able to map each of its point to the closest of them. In the centralized version of K-MEANS, we select all the points that have been mapped to a single centroid and compute their mean to get the new position of that centroid. In this distributed algorithm, each node computes the position of each centroid as the mean of the points in the neighborhood that have been mapped to it.

An interesting property to be emphasized is the fact that each node only communicates with the others using aggregate data. In the communication rounds the nodes only send their centroids to some of their neighbors. The actual points are not shared in the network and each node can avoid sharing information with the other nodes. This is highly desirable when the data is distributed between nodes representing different companies and they want to collaborate to get a good clustering without having to directly share their points with the competition.

4 The Algorithm

The original distributed K-MEANS algorithm works on static datasets and needs to know the number of clusters that better represent the data distribution in advance, or it will fail to produce a good clustering [1]. We use this algorithm as a subroutine and we create a novel distributed framework to discover the correct number of clusters in data streams.

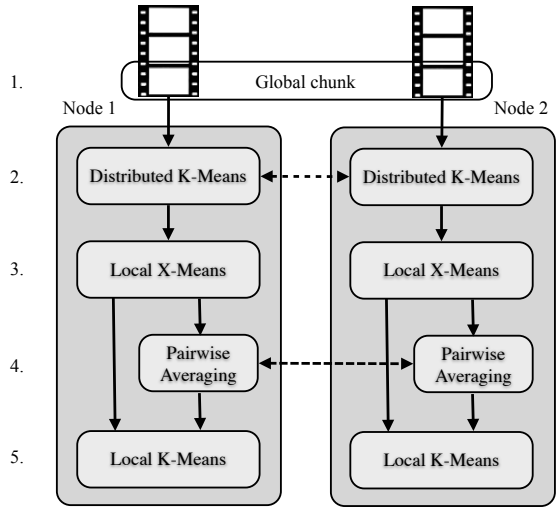


Fig. 1. Overview of DS-MEANS

We provide here an overall view of the algorithm, while implementation details are provided in the following subsections. The algorithm is divided in the following five steps, numbered (1)–(5), illustrated in Figure 1 and whose pseudocode is outlined in Figure 2:

1. Each node p_i receives a stream of data points p, which are collected in variable C. The execution is divided into approximately synchronized *epochs* of duration Δ; at the beginning of each epoch t, a *chunk* of data denoted containing the points collected in the last Δ time units is stored in variable $chunk^i(t)$ and variable C is emptied.
2. The set of chunks, one for each node, that have been created at epoch t is called *global chunk* and is denoted $chunk(t)$:

$$chunk(t) = \{chunk^1(t), chunk^2(t), \ldots, chunk^N(t)\}$$

   The nodes execute an independent instance of the distributed K-MEANS algorithm on each global chunk $chunk(t)$, using an arbitrary value of K. Each node p_i generates a collection of centroids $centroids^i(t)$ of size K. The new centroids are added to the set $centroids^i$, which contains all centroids generated so far by node i. The original data points are discarded.
3. Each node p_i execute the centralized version of X-MEANS on $centroids^i$ to obtain the number G^i of centroids that better represent $centroids^i$.
4. The resulting number of centroids could be different at different nodes, so a pairwise averaging algorithm [5] is run on such values. All nodes will obtain the same average number $G = 1/N \sum_{i=1}^{N} G_i$.
5. Finally, each node runs a local version of K-MEANS on $centroids^i$, using K equal to the number of clusters resulting from the previous step ($K = G$). The resulting set $centroids^i_G$ is the output of the algorithm.

```
    on receive p
(1)   C ← C ∪ {p}

    repeat every Δ time units
        t ← t + 1
(1)     chunk^i(t) ← C
        C ← ∅
(2)     centroids^i(t) ← DistKMeans(chunk^i(t), K)
        centroids^i ← centroids^i ∪ centroids^i(t)
(3)     G^i ← XMeans(centroids^i)
(4)     K ← distributedAverage(G^i)
(5)     centroids^i_G ← KMeans(centroids^i, K)
```

Fig. 2. Algorithm executed by node p_i

DS-MEANS can be transformed into a dynamic algorithm that quickly adapts to new data arriving in the system. Assuming that the last estimate of the number of clusters is K_{old}, the framework will react as follows:

- All nodes in the network will start an instance of the distributed K-MEANS algorithm only on the new global chunk, using $K = K_{old}$.
- When the distributed K-MEANS algorithm has terminated, all nodes update their list of centroids by adding the new ones. In a "sliding window" model, they also discard the centroids generated w epochs ago, where w is the size of the sliding window (in number of epochs):

$$centroids^i \leftarrow centroids^i \cup centroids^i(t) - centroids^i(t-w)$$

- Each node will run the local X-MEANS algorithm giving it an upper bound on the number of clusters based on K_{old}.
- The pairwise aggregation step and the local K-MEANS step is repeated as usual, thus obtaining a new estimate of the number of clusters and a new clustering.

The choice to partition data into (global) chunks helps us in avoiding repeating unnecessary computations in the distributed K-MEANS step. Reusing the old number of clusters as K helps in making even less important the value of K given to DS-MEANS at startup.

Note that the nodes communicate only during two steps: for distributed K-MEANS and for pairwise averaging. During the former, only centroids are sent over the network, while in the latter the only shared value is the number of centroids. Actual points are never shared across the network, giving us a sufficient (although limited) degree of privacy.

4.1 Dividing data into chunks

We derive the idea of dividing data into chunks and working on each of them separately by the work of Guha et al. [4]. The easiest way to divide the data is

based on the timestamp (creating a new chunk for each interval of time). Even if the nodes receive data at different rates and therefore create chunks of different size, the distributed K-MEANS algorithm makes sure that each node has similar results. We force each node to generate the same number of chunks, so that each global chunk will contain exactly one chunk from each node.

4.2 Distributed K-Means

The nodes in the network need to execute the distributed K-MEANS algorithm on each global chunk. We start an execution of the distributed K-MEANS algorithm any time a new global chunk arrives, meaning that each node has a new chunk to work on. Each execution is completely independent and chooses its own starting centroids. At the end of each execution, K new centroids are created (roughly the same in all nodes), which are added to the list of centroids computed so far. This list acts as a compact representation of all data in the network.

Each execution selects a set of starting centroids, common to all nodes, as follows: each node chooses randomly K centroids and a real number c in the range $[0, 1]$. A round-based epidemic protocol is executed to reconcile the different sets of centroids [2]. At each round, each node communicates with a random subset of neighbors and inherits both the centroids and c from the neighbor having the greatest value of c. Thanks to this epidemic protocol, $O(\log n)$ rounds are sufficient to have all nodes knowing the same set of starting centroids.

Another important part of this algorithm is the distributed termination condition. In the centralized K-MEANS, the algorithm terminates when there are no more updates to the centroids and a steady state has been reached. This approach is not viable in a decentralized setting due to the lack of a global view of the system, so we need to define a local condition to be checked by individual nodes. Each node keeps track of the last set of centroids it has generated and, at each iteration, checks if they have been changed by measuring the distance between the centroids and checking if the average change is more than a given threshold δ. A node terminates the execution and outputs its centroids when both the local centroids and the ones of the contacted neighbors have not changed in the last τ iterations.

4.3 X-Means

In this step, each node executes its own instance of X-MEANS on the list of centroids obtained from the previous step. Different runs of X-MEANS will result in different clustering and, in unlucky cases, in a number of clusters far from the correct answer. By running X-MEANS separately on each node in the network, we use redundancy to discard unlucky instances of X-MEANS.

Since we give as input to the X-MEANS algorithm a set of points $centroids^i$, each of them representing a subset of the original data points, we make each node compute the variance of each centroid in the list using its own data and we use this value as a "lower bound" on the cost of making a cluster containing that centroid.

4.4 Pairwise averaging

The number of clusters identified by the local execution of X-MEANS can vary significantly between different nodes, while it would be desirable to obtain in all nodes a clustering with the same number of clusters. To obtain this we use a simple, epidemic-based pairwise averaging algorithm [5]. After $O(\log N)$ epidemic rounds all nodes know the average number of clusters computed by the different instances of X-MEANS.

4.5 Local K-Means

When the pairwise averaging step is over, all nodes have the same estimate of the number of clusters in the underlying data. Now each node can run the centralized K-MEANS algorithm on the local list of centroids using the value K computed in the previous steps. We can reuse some of the centroids found by the local X-MEANS as the starting centroids for the local K-MEANS, since they work on the same dataset $centroids^i$.

A more efficient implementation of this algorithm could share additional information between the two algorithms. If we wanted to compute some of the data structures commonly used to speed up clustering algorithms (like KD-Trees [8]), we could reuse them in both algorithm, thus avoiding unnecessary computations.

5 Evaluation

DS-MEANS has been tested on PeerSim [7], a P2P simulator. In this section, first the experimental framework used to evaluate DS-MEANS is presented, then the results are shown.

5.1 Experimental Framework

The input of the evaluation is a list of data points, each labeled with the correct group to which they belong. The output is a similar list, with points labeled by the clusters discovered through DS-MEANS.

We compare DS-MEANS against the centralized version of K-MEANS (instructed with the correct value of K) and the centralized version of X-MEANS. Four figures of merit are considered: *Clustering quality* is measured through the F-score, the harmonic mean of precision and recall, and through the within-cluster sum of squares. *Execution time* is measured in number of communication iterations in the simulation. Note that comparing execution time of centralized and decentralized algorithms is hardly significant, because the latter strongly depends on the underlying network. The fourth figure of merit is thus *total computational work*, measured as the number of times that the distance function (used to compare two data points) is called across the entire execution.

An artificial dataset is generated as follows. First of all, G different *mean points* are randomly chosen from a d-dimensional space, one for each of the

groups in which data points are correctly subdivided. Then, each point in the dataset is created in two steps: (i) one of the groups is selected uniformly at random, and the point is labeled with it; (ii) the actual point coordinates are generated following a standard Gaussian distribution (in each of the d dimensions) centered in the corresponding mean point.

A couple of observations are in order:

- Given that mean points are independently generated, groups may overlap, inducing errors when a clustering algorithm is applied. All potential clustering algorithms are affected in the same way by this problem, so our comparison against centralized algorithms is fair.
- Data points are equally divided among nodes, and then divided in chunks. We do not ensure that a chunk will contain data from all of the original groups; this most likely occurs when the number of points in a chunk is small.

Parameter	**Symbol**	**Value**
Network size	N	100
Groups	G	20
Data dimensionality	d	2
Threshold iterations	τ	3
Threshold variance	δ	0.5
Size of the space	M	100

Table 1. Simulation parameters

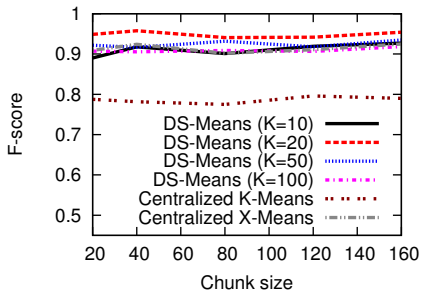
Each experiment is repeated 20 times; variance is so small that it is not shown in the figures. Unless explicitly stated otherwise, our results have been obtained with the parameters listed in Table 1.

5.2 Experimental Results

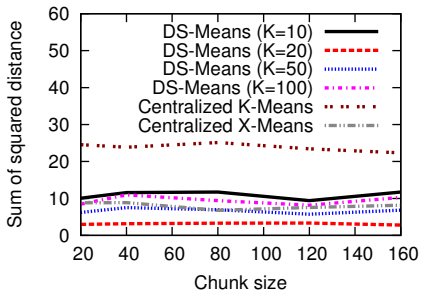
Figure 3(a) compares the F-score of DS-MEANS and the two centralized clustering algorithms. We can see that the centralized K-MEANS algorithm obtains a clustering of lower quality, even when it is given the correct value of G. This is caused by the usual problem of the presence of local minima, that are better avoided by X-MEANS. DS-MEANS obtains very good F-scores not only when the correct K is used to initialize to the algorithm, but it even obtains results comparable with X-MEANS with the wrong value of K.

Figure 3(b) shows the average within-cluster sum of squares distance of DS-MEANS. As before, the experiments show results equivalent to the centralized X-MEANS.

Another interesting property of DS-MEANS is the amount of computation (measured by the number of calls to the distance function) that is needed to complete the execution. Figure 4(a) shows that our clustering algorithm is always comparable to the centralized algorithm.

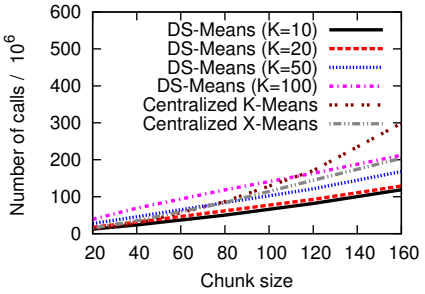


(a) Average F-score against two centralized algorithms

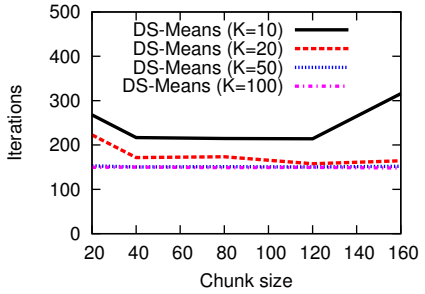


(b) Average within-cluster sum of squares against two centralized algorithms

Fig. 3. Precision of DS-MEANS using different values for K.



(a) Total computational work against two centralized algorithms

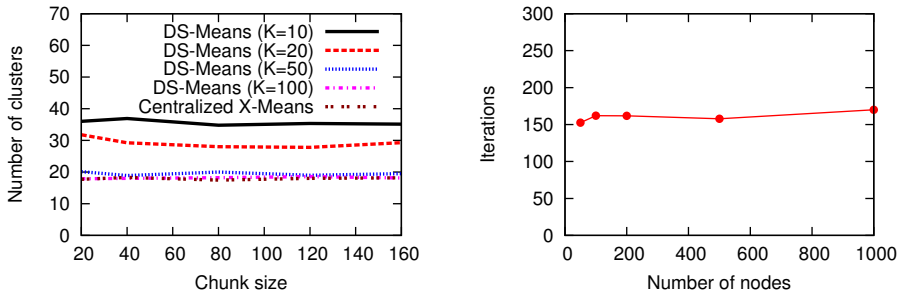


(b) Average execution time

Fig. 4. Computational work and execution time of DS-MEANS using different values for K.

Figure 4(b) shows that, even if the amount of computation is larger, the execution time (in which we only take into account the communication costs) seems to become lower when the algorithm is given a bigger K.

While we have seen that the precision of DS-MEANS is high even when given the incorrect number of clusters, it could be interesting to see what is the actual number of clusters obtained from it. Figure 5(a) shows the number of clusters found by DS-MEANS when given different values of K, when $G = 20$ different groups have been created. The results are quite interesting: the algorithm is closer to the correct answer when it is given a value of K bigger than necessary, while when it is given the correct K the number of clusters it creates is bigger than necessary. This behavior is easily explained: giving a bigger K to the distributed K-MEANS algorithm results in a bigger number of centroids generated and thus a better representation of the data. When K is small then it is more likely that in the distributed K-MEANS algorithm more than one cluster is mapped to a single centroid, thus creating a new point in the list of centroids that does not correspond to any of the 20 original groups. The reason the algo-



(a) Average number of clusters (b) Scalability: execution time

Fig. 5. Average number of clusters and scalability given $G = 20$ groups and an initial value $K = 40$.

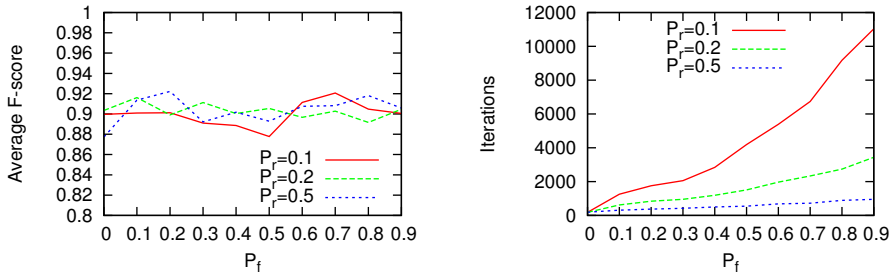


Fig. 6. Robustness of DS-MEANS with different recovery probabilities

rithm is still very precise is that these centroids, being quite far from the real distributions, will not be used when mapping the data of the entire network to the clusters.

As the next step we want to see the behavior of our distributed clustering algorithm when the network grows in size. In Figure 5(b) we see the execution time of our clustering algorithm against the number of nodes in the network. We see that the differences are very small and that we need a similar amount of communication iterations to complete the algorithm. This fact was expected since the nodes in the network only look and communicate to their local neighbors and the data is distributed homogeneously in the network.

Finally, an important aspect to be analyzed is the robustness of our approach. In our model, nodes may go down for a period of time before getting repaired and rejoining the distributed algorithm. Nodes that are down do not communicate with the rest of the network and do not work on their data. They are still able to receive a new chunk and store it until the node goes up. This model has been chosen to make sure that each node has the data on which it has to work.

Our model for robustness uses two parameters: the probability of failure (P_f) and the probability of recovery (P_r). All nodes start the computation in the up state and then, at each iteration of the simulation, each node in the up state

goes down with probability P_f, while each node in the down state goes up with probability P_r.

In Figure 6(a) the F-score of our algorithm with different values of these two parameters is shown. In this simulation, while $G = 20$, the algorithm has been initialized with $K = 40$. Even with very high failure percentage and low recovery rate the algorithm is able to reach roughly the same F-score. Note that in the case in which the probability of recovery is 0.1 and the probability of failure is 0.9, on average 90% of the nodes in the network are down at all times. The algorithm is still able to reach a good clustering even in these extreme conditions, at the price of an increase in the number of communication iterations (see Figure 6(b)).

6 Conclusion

In this paper, we have presented DS-MEANS, a novel algorithm for clustering distributed data streams. Preliminary results with synthetic datasets are promising; we are now evaluating the system with realistic data for botnet detection.

References

1. Bandyopadhyay, S., Giannella, C., Maulik, U., Kargupta, H., Liu, K., Datta, S.: Clustering distributed data streams in peer-to-peer environments. Information Sciences 176(14), 1952 – 1985 (2006)
2. Demers, A., Greene, D., Hauser, C., Irish, W., Larson, J., Shenker, S., Sturgis, H., Swinehart, D., Terry, D.: Epidemic algorithms for replicated database maintenance. In: Proceedings of the sixth annual ACM Symposium on Principles of distributed computing. pp. 1–12. ACM (1987)
3. Gu, G., Perdisci, R., Zhang, J., Lee, W.: BotMiner: Clustering analysis of network traffic for protocol-and structure-independent botnet detection. In: Proc. of the 17th USENIX Security Conference. pp. 139–154 (2008)
4. Guha, S., Meyerson, A., Mishra, N., Motwani, R., O'Callaghan, L.: Clustering data streams: Theory and practice. IEEE Transactions on Knowledge and Data Engineering pp. 515–528 (2003)
5. Jelasity, M., Montresor, A., Babaoglu, O.: Gossip-based aggregation in large dynamic networks. ACM Transactions on Computer Systems (TOCS) 23(3), 219–252 (2005)
6. Lloyd, S.: Least squares quantization in PCM. IEEE Transactions On Information Theory 28(2), 129–137 (1982)
7. Montresor, A., Jelasity, M.: PeerSim: A scalable P2P simulator. In: Proc. of the 9th Int. Conference on Peer-to-Peer (P2P'09). pp. 99–100. Seattle, WA (Sep 2009)
8. Pelleg, D., Moore, A.: Accelerating exact k-means algorithms with geometric reasoning. In: Proc. of the 5th Int. Conference on Knowledge Discovery and Data Mining (KDD'99). pp. 277–281. ACM, San Diego, California, United States (1999)
9. Pelleg, D., Moore, A.W.: X-means: Extending k-means with efficient estimation of the number of clusters. In: Proc. of the 17th Int. Conference on Machine Learning (ICML'00). pp. 727–734. Morgan Kaufmann, San Francisco, CA, USA (2000)
10. Schwarz, G.: Estimating the dimension of a model. The Annals of Statistics 6(2), 461–464 (1978)