

A Distributed Clustering Framework in Mobile Ad Hoc Networks

Mohit Garg and R. K. Shyamasundar *Fellow, IEEE*

Abstract—In this paper, we present a completely distributed algorithm for partitioning a given set of mobile nodes into clusters. The proposed algorithm tries to reduce the amount of computational and information overhead while maintaining a stable cluster formation. The algorithm is a distributed version of the *Basic Leader-Follower Algorithm* used in pattern recognition. Our algorithm does not require maintaining a cluster head. The parameters of the algorithm give us ‘good’ control over the cluster properties.

Index Terms—Mobile Ad-Hoc Networks, Distributed Clustering, Distributed Basic Leader-Follower Algorithm.

I. INTRODUCTION

Mobile Ad-Hoc Networks (MANets) were primarily investigated for their use in military applications. But with the burgeoning need for more and more services for the users, MANets have attracted a lot of industry interest recently. Ad-Hoc networks are wireless networks which do not require any fixed infrastructure. They can be quickly set up without the need of any base station or access points as is the case of traditional wireless networks. The temporary nature of Ad-Hoc networks makes them ideal for applications like military communication, disaster relief work, conferences etc.

An ad-hoc network is usually modeled by an undirected graph with each vertex representing a node in the network. An edge between two vertexes implies that the two corresponding nodes can communicate with each other (*i.e.* each link is bi-directional). Such nodes are called *neighbours*. Since the transmission range of each node is limited, this places a limit on the maximum length of each edge in the graph (if the graph is drawn to scale). Thus a node **A** may be able to communicate directly (neighbour) with another node **B**, but not so with a node **C**. Thus, in order for **A** to communicate with **C**,

Mohit Garg is with the Department of Electrical Engineering, Indian Institute of Technology Bombay, Powai, Mumbai 400076, India, email: mohitgarg@iitb.ac.in and R.K. Shyamasundar is with School of Technology and Computer Science, Tata Institute of Fundamental Research, Homi Bhabha Road, Mumbai 400005, India, email: shyam@tcs.tifr.res.in.

it will need to route the packets via another node or set of nodes. This is termed as *multi-hopping* (needed either for energy saving or extending the range).

A. Routing in MANets

Finding and maintaining stable routes in an ad-hoc network is not an easy task. The major stumbling block is the mobility of the nodes. Almost all the approaches in literature can be broadly classified into the following two categories :-

- *Pro-Active routing* involves keeping routes to all possible destinations (which is usually all nodes in the system) and keeping a track of the link parameters so as to find Quality of Service (QoS) enabled paths quickly as and when needed by an application. These techniques require an overhead for maintaining and exchanging information about the state of the network. [9], [10].
- *Reactive routing* on the contrary tries to find paths on demand; *i.e.*, as and when needed by an application, the source node will initiate a route discovery. These algorithms involve relatively lesser overhead for exchanging information about network state but are characterized by larger delays in finding paths. Flooding algorithms can also be clubbed with reactive routing algorithms. [11], [12]

Both pro-active and reactive approaches are not scalable. In a large network, while the former would need a very large overhead to exchange network state information (which will compete with the data packets for the already scarce bandwidth), the latter will lead to large delays especially when the source and destination nodes are separated by a large number of hops. Clearly, some sort of a compromise needs to be achieved between the two strategies. *Clustering* strategies tend to do just that. Clustering divides the given network into discrete partitions. Most of the cluster-based routing algorithms tend to use pro-active approaches within the cluster and reactive routing for inter-cluster routing [1], [4]. This enables one to reach a sort of compromise between pro-active and reactive routing strategies. Our approach also uses the same routing strategy with a slight addition (cf.

Section IV). Clustering also provides one with some sort of a veil with respect to the mobility of a node *i.e.* a node in cluster **X** does not bother about the mobility of a node in cluster **Y**. Only nodes in the respective clusters update their own links and routes when a node (which is in their cluster) moves. We shall discuss clustering in detail in section III.

The remaining part of this paper is organized as follows: In Section II, we discuss the significant works done in the field of cluster-based routing in Mobile Ad Hoc Networks. In Section III, we present our clustering algorithm and discuss the related issues. The simulation results of the clustering technique are presented in Section IV. The paper concludes with a discussion in section V.

II. RELATED WORK

Cluster formation in Mobile Ad-Hoc Networks have attracted considerable attention in recent times [4]–[7]. Most of the current approaches are centered around the idea of a clusterhead which is entrusted the task of controlling the cluster nodes and maintaining the cluster. This leads to unnecessary burden on the clusterhead which itself has limited resources. Also, since most of the routing decisions are taken by the clusterheads and they also serve as routers for the packets [13], these nodes may become bottlenecks in the system. Our scheme does not involve clusterheads and hence, does not suffer from this drawback.

There have been investigations with clusterhead-less strategies as well. Krishna *et al* [8] propose a clusterhead-less scheme with overlapping clusters. A drawback in this algorithm is that every new node leads to formation of new clusters and rearrangement of old ones which may lead to too many changes in a dynamic network. Also, the algorithm requires a three step procedure to update the clusters each time a change in the local topology occurs. Our proposed scheme does not require any iteration.

Lin and Gerla [2] present a simple and elegant distributed algorithm for partitioning the network into clusters (without the need of a clusterhead) with each cluster being at most 2-hops wide (*i.e.*, each node in a cluster is at most 2 hops away from any other node in the same cluster). This algorithm assumes that each node knows *all* of its immediate neighbours at the time of cluster formation. Our algorithm makes no such assumption.

McDonald and Znati [1] propose a novel prediction based clustering strategy in which they use previous information to predict the mobility of each node in a cluster and quantify the stability of a cluster based on it. Their algorithm does not need any clusterhead

and they term it as (α, t) cluster. Their algorithm is computationally expensive and may be a drain on the scarce resources of the mobile nodes. Our algorithm in contrast, does not try any prediction based approach in order to keep things simple and easy to implement. Perkins and Bhagwat note in [10] that “*Simplicity is one of the primary attributes which makes any routing protocol preferred over others for implementation in operational networks.*” Thus, our main goal is to simplify the clustering process and still provide a stable backbone for the network routing protocols. Next we present our clustering algorithm.

III. A NEW DISTRIBUTED CLUSTERING ALGORITHM

In Mobile Ad-Hoc Networks, mobility leads to a lot of issues and link breakage may lead to invalidation of existing paths. This leads to problems in coping up with QoS assurances. One could try and work around the mobility problem in two ways :-

- By using the mobility to our advantage *i.e.*, increased throughputs for mobile networks have been reported in [14] but they are only confined to non-real time applications till now and hence, are not applicable in QoS frameworks.
- By trying to achieve some sort of stability and reducing the cascading effect of node mobility *i.e.*, only large scale mobility of nodes be visible to the nodes far away in the network.

The latter motivates one to look for clustering techniques. Since, an Ad-Hoc network is a completely distributed network with no central authority, it makes sense only to look for algorithms which work in a completely distributed manner. We propose an algorithm which in essence is the distributed version of the *Basic Leader-Follower Algorithm* used in pattern recognition [3]. We call it the *distributed-BLF (d-BLF) Algorithm*. The phrase “Leader-Follower” is misleading here in the sense that our clustering strategy does not require one to maintain a clusterhead or leader. We divide the given network into a set of non-overlapping clusters.

We emphasize that the clustering mechanism is completely independent of the routing algorithm employed. Any routing algorithm which can work on a partitioned network should in principle be able to work with any clustering algorithm.

The basic leader-follower algorithm used for clustering in pattern recognition systems, is ideally suited for Mobile Ad-Hoc networks since it is an on-line algorithm (*i.e.*, it dynamically forms new clusters as and when data points come up) and is a part of the large class

```

1) begin initialize  $\theta$ 
2)    $\mathbf{c}_1 \leftarrow \mathbf{x}$ 
3)   do accept new  $\mathbf{x}$ 
4)      $j \leftarrow \arg(\min_i \|\mathbf{x} - \mathbf{c}_i\|)$ 
5)     if  $\|\mathbf{x} - \mathbf{c}_i\| < \theta$ 
6)       then add  $\mathbf{x}$  to  $c_j$  and update  $c_j$ 
7)     else add new  $\mathbf{c} \leftarrow \mathbf{x}$ 
8)   until no more clusters
9) end

```

TABLE I: Basic Leader Follower Algorithm

of Unsupervised Learning algorithms [3] *i.e.*, no central control is necessary in order to implement it.

Table I lists the steps involved in clustering the data points into clusters. Note that this algorithm is single pass and does not require iteration and hence converges much faster than many other clustering algorithms. In order to adapt it to our case, we need to :-

- 1) First define the metric on the basis of which we will be adding nodes to clusters and the threshold involved. Note that the standard BLF algorithm uses the Euclidean distance as the measure of closeness since it is only concerned with the static case. We will need to define an appropriate measure to be able to capture the mobility. (Stability Metric)
- 2) Second, we need to adapt the algorithm so that clustering can be performed in a distributed manner and each node on the basis of local information can join the cluster that leads to a “globally best” partition. (Distributed-BLF Algorithm)

The Distributed-BLF algorithm will have the following phases: Cluster Formation, Cluster Maintenance, and Cluster Disintegration. In the following, we shall discuss these phases keeping the simple algorithm described and issues in view. The presentations will be informal for lack of space.

1) *Stability Metric*: Optimality of the algorithm can be derived from the optimality of the BLF algorithm but that will be valid only for the static case, *i.e.*, the network topology being fixed and nodes not moving. The notion of “optimal clustering” for a dynamic topology is difficult to define and hence, the cluster formation and stability will be determined by the metric that we choose in the BLF algorithm. It should be clear that clusters in which nodes are mobile are less likely to remain stable for longer times than those in which they are stationary. Hence, each cluster should have some measure of node mobility in it. One could attempt a prediction based metric (as in [1], [4]) but all such algorithms, apart from being too complex, tend to be computationally prohibitive and hence, are not likely to be used in actual implementations.

A compromise can be reached if each node monitors the number of new nodes in the cluster *i.e.*, the time for which they have been up – new nodes can be classified as transient (liable to move away soon) and those which are old can be classified as stable nodes. Thus, each node is initially kept in a transient state (for $T = T_{transient}$). The motivation behind this is that newer nodes are more likely to move away in the near future (since the node may be mobile). *The fraction of the nodes in the cluster which are stable determine the stability of the cluster and is taken as the stability metric in our approach*¹.

Thus, this metric is known to all nodes in the cluster and an incoming node can inquire about this from them and make a decision about its joining that particular cluster or not. We wish to emphasize that this metric is only used for an incoming node that wishes to join the clusters. Cluster disintegration does not take place if the above mentioned stability metric goes below a threshold. The exact conditions and mechanism of cluster disintegration is discussed later.

2) *Distributed BLF Algorithm*: We now present the d-BLF algorithm. We assume that there exists a MAC protocol which takes care of collisions and that a channel access policy is in place. We also limit the size (number of nodes) of each cluster to be less than *MAX_CLUS_SIZE* and larger than *MIN_CLUS_SIZE*. This keeps a check on the number of nodes in a cluster. The lower limit is not a strict limit *i.e.*, there may exist clusters smaller than that size but they will tend to disintegrate.

Let us say that a node currently not a member of any cluster need to indicate its willingness to join a cluster through the flag Switch-ON. It periodically broadcasts a *CLUS_FIND* message which if heard by a node which is currently a member of a cluster it sends back a *CLUS_INFO* reply containing the cluster name, the number of members in the cluster and its stability metric. The new node, keeps accumulating all such messages for a given amount of time, chooses the cluster which it wants to join and sends back an *CLUS_JOIN_REQUEST* message. The cluster node which had earlier sent the *CLUS_INFO* message, on receiving the *CLUS_JOIN_REQUEST* message adds the node to the cluster, sends *CLUS_JOIN_ACCEPT* message back to the requesting node and broadcasts the updated routing table within the cluster. The node now becomes a member of the cluster and is included in all future routing table updates. All the above message exchanges will be accompanied by timeouts so as to

¹We have also simulated a Gaussian stability metric and the results are presented in Section IV.

prevent a node from waiting ad infinitum for a particular reply.

The new node will select the cluster based on the stability metric and the size of the clusters. The primary criteria will be the largest stability metric (greater than the threshold: *STABILITY_THRESH*) but in case the cluster already has a size equal to *MAX_CLUS_SIZE*, it will proceed with other options. An exception to the above rule will be when the node finds a cluster with size=*MIN_CLUS_SIZE*-1. In this case addition of this node to that cluster will make it a stable cluster and hence it will join this cluster if its stability metric is above the threshold irrespective of the stability metrics of other larger sized clusters. A tie in the exception can be broken by the larger stability metric criteria.

In the above scenario, it is possible that the node – a) does not find a cluster around it, b) does not find one satisfying both the stability and size criteria. In both these cases the node forms its own cluster. In this case, its size is less than *MIN_CLUS_SIZE* (if *MIN_CLUS_SIZE* > 1) and it keeps working according to cluster disintegration mechanism as explained in subsection III-C.

Also, at every point in time, every cluster has an ID which is same as the ID of the lowest ID member of the cluster.

A. Cluster Maintenance

There are four topology changes that will require cluster updating:-

- 1) *A Node Switches ON*: This will be handled in exactly the same way as cluster formation. This case includes the case when a new node comes up which is not currently a member of any cluster.
- 2) *A Node Switched OFF (or fails)*: This may lead to one or more links inside a cluster being broken. The nodes neighbouring to the failed node will detect the link loss and will send the routing table updates inside the cluster. Every node will update their respective tables and will find that no intra-cluster path exists to this node. Hence, it will be deleted from the cluster and the cluster parameters (*i.e.*, cluster ID, stability metric, size) will be updated. In case the failed node was a border node, the nodes in the adjacent cluster will also note a loss of link to it and update their respective routing tables.
- 3) *A new link (with a node already bound to a cluster) is detected*: This means that an existing node (say, Node **A**) has moved into the hearing range of another node (Node **B**). Both nodes **A**

```

start
if myself_unclustered then
  broadcast(clus_find)
  waitfor(responses)
  parse_responses()
  if suitable_cluster_exists()
    send(clus_join_request)
    waitfor(clus_join_reply)
    if clus_join_accept then updatemyclus()
    else form_own_clus()
  endif
endif
endif
if size(myclus) < MIN_CLUS_SIZE then
  if time_for_disintegrate_broadcast() then
    broadcast(clus_find)
  endif
endif
endif
do_other_work()
if received(clus_info) then
  check_suitability()
  if suitable_cluster_exists then
    send(clus_join_request)
    waitfor(clus_join_reply)
    if clus_join_accept updatemyclus()
  endif
endif
endif
goto start
end

```

TABLE II: Algorithm to be followed by each node

and **B** will find out the cluster affiliations of each other, update their own routing tables and broadcast this information in their respective next routing table updates. Note that both these nodes may now become border nodes (if both of them are in different clusters).

- 4) *A link loss is detected*: This scenario is characterized by a node not receiving the routing table update from its neighbour for a particular interval of time. It will then, wait for another larger interval of time and declare the link as lost if it does not hear from the node again. The next routing update will reflect this new information. Note that a link loss may also occur between border nodes which can again be detected by a border node not receiving any adjacent-cluster-routing-table update from its neighbour in an adjacent cluster.

In all the above cases, each node in the cluster on receiving information about the change, will update the cluster parameters (*i.e.*, cluster ID, stability metric, size etc.) independently of each other.

One can see that the above algorithm inherently assumes that the topology changes are “slow enough” so as to allow the updates due to an earlier topology change to settle down. Such an assumption is inherent

in any pro-active algorithm and in case the rate of topological changes in the network are too high, flooding and reactive algorithms may be the only option left.

B. Cluster Disintegration

In order to push the network towards forming larger and hence lesser number of clusters, we keep a minimum limit on the cluster sizes (MIN_CLUS_SIZE). Thus clusters with sizes less than the minimum size are unstable and will tend to disintegrate. Note that this is the **ONLY** condition under which cluster members are allowed to defect from the cluster in case they find better and larger clusters to join. Clusters do not disintegrate even if their stability metrics go below the $STABILITY_THRESH$. This threshold is only provided for the incoming nodes to select the cluster they want to join (if they want to join or else form their own cluster).

Hence, if a cluster has a size lower than the minimum allowed, each member in the cluster is allowed to look for “greener pastures” independently and join one of them if it finds a better cluster(s). *i.e.*, each node periodically broadcasts a $CLUS_FIND$ similar to the one sent by a new node in the cluster formation process. A similar exchange of messages occur and the node joins the new cluster if it is better than its current cluster. In this case apart from the cluster stability metric (which is the primary filtering criteria), the node will also look only for clusters with a size greater than or equal to the size of its current cluster. In case the node finds only clusters around it which have the same size as its own cluster, it will compare the cluster IDs and will decide to join the new cluster (whose ID is the smallest of the new cluster IDs), only if the new cluster ID is smaller than its current cluster ID. If this case does not happen, then the selection (after using both the stability and size metrics) of the new cluster will happen based on the cluster formation rules as discussed in Section III-A.

IV. SIMULATIONS

We have extensively simulated the algorithm using *pthread*s, a POSIX Threads library for Linux. Representing each node by a thread allowed us to simulate inter-node communication in a distributed manner.

The simulation environment consisted of a time-slotted MAC layer and 75 nodes in an area of 100×100 units. Each node had a transmission range of 15 units. Nodes were switched on at a constant rate in the initial part of the simulations. Clustering was performed dynamically and movement was imparted to a random number (with mean half of the nodes ON at

that time) of nodes with exponential velocities (mean=50 units/iteration).

Each reading was averaged over 10 random graphs each run for 5000 iterations. All results used a $step$ stability metric with a $STABILITY_THRESH$ of 0.3 except for Figure 1. We used $T_{transient} = 50$ everywhere.

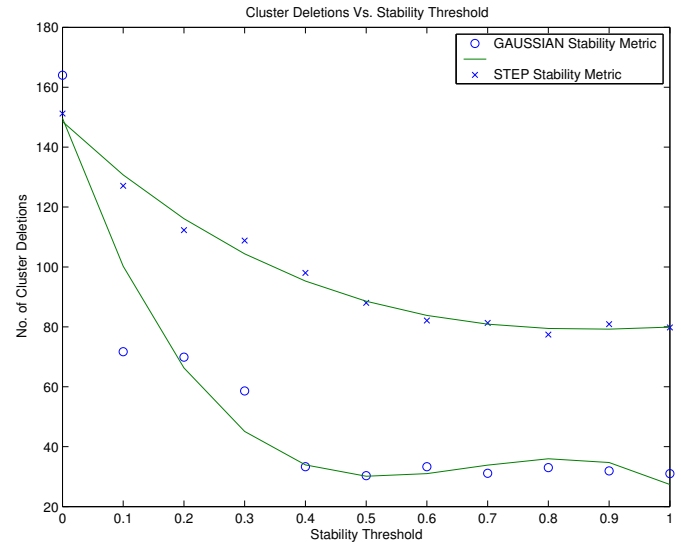


Fig. 1: No. of Cluster Deletions Vs. Stability Threshold

Fig. 1 shows the variation of number of cluster deletions in the system with the stability threshold. (‘Cluster Deletion’ refers to the event when all members of the cluster are lost, *i.e.* the number of members in the cluster go down to zero.) We clearly see a downward trend in the cluster deletions as the stability threshold is increased. We used two different stability metrics: a $step$ metric which is the fraction of nodes in the cluster who have remained in the cluster for a minimum amount of time (‘cluster age’ of a node). The second one was an $Gaussian$ metric in which nodes were given Gaussian weights w.r.t. their ‘cluster age’. The Gaussian metric gives lower weights to ‘young’ and ‘old’ nodes while giving higher weight-age to number of ‘middle aged’ nodes in the cluster. The defining time $T_{transient} = 50$ was used in both cases.

The following Stability Metrics were used: Step Metric,

$$stab = \frac{\sum_i^{N_c} I((T_i - CLK + T_{transient}) \leq 0)}{N_c}$$

Gaussian Metric,

$$stab = \frac{\sum_i^{N_c} e^{-(T_i - CLK + T_{transient})^2}}{N_c}$$

where, N_c is the number of elements in the cluster, and T_i is the time at which the i^{th} node joined the cluster and CLK is the global system clock.

Note that a higher stability metric leads to a larger number of clusters which may not be desirable. Thus, we can choose the suitable stability threshold and optimize the performance of the clustering technique.

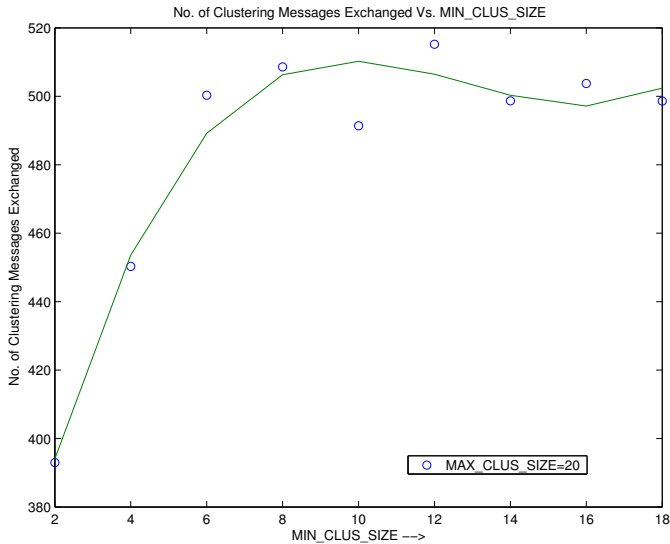


Fig. 2: No. of Clustering Messages Exchanged Vs. MIN_CLUS_SIZE

In Fig. 2, we can see the variation in the number of cluster maintenance messages exchanged with increasing MIN_CLUS_SIZE . As expected, more messages are exchanged as MIN_CLUS_SIZE is increased since ‘smaller’ clusters will tend to disintegrate and join bigger clusters and MIN_CLUS_SIZE defines the limit of being ‘small’. The higher the limit, higher is the number of clusters that come in the ‘small’ category thereby explaining the observations.

In Figures 3 and 4, we have plotted cluster size distributions under the simulation conditions. It is clear from the plots that for a fixed MAX_CLUS_SIZE , as MIN_CLUS_SIZE increases, clusters tend to be bigger. One can see that (as expected) the number of clusters decreases as larger clusters are allowed. It is important to note that since we chose random graphs, in many cases there were isolated nodes in the network which did not have any neighbours with which they could form larger clusters. Hence there were unavoidable singleton clusters in some cases. Thus both the parameters can be chosen and varied to get the desired clustering behaviour.

V. CONCLUSION

In this work, we have proposed a new, completely distributed, adaptive clustering algorithm for Mobile Ad

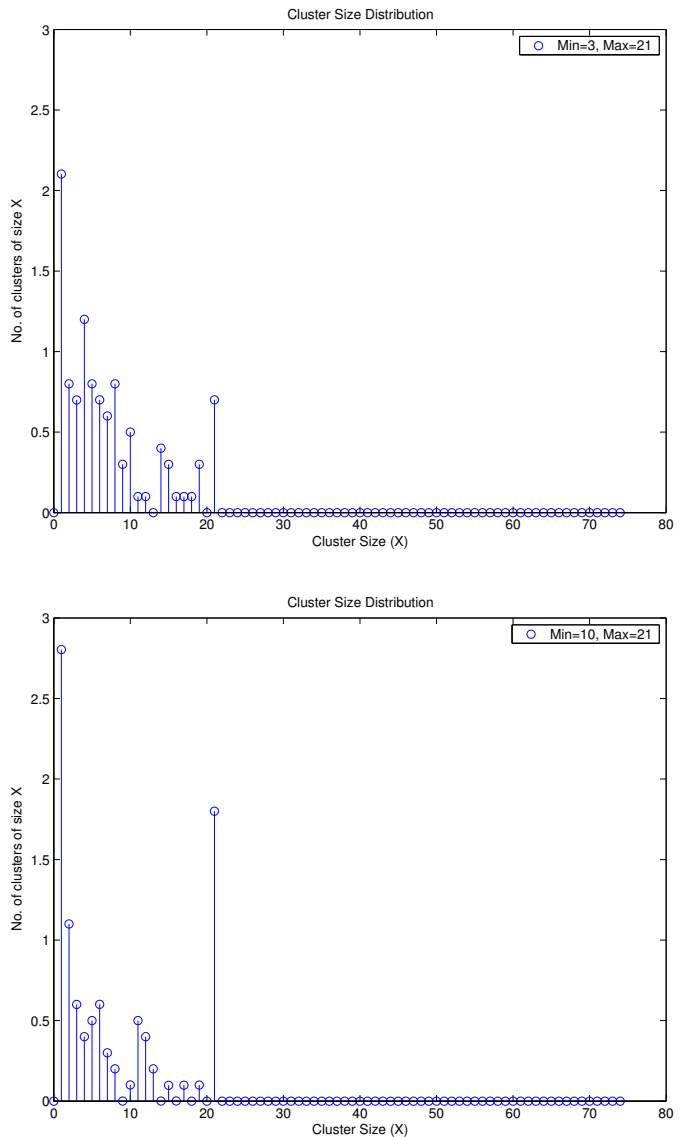


Fig. 3: Cluster Size Distribution ($MAX_CLUS_SIZE=21$)

Hoc Networks. The algorithm is a distributed version of the Basic Leader-Follower Algorithm and we term it as the d-BLF algorithm. The size of the clusters is upper bounded and the number of clusters and their dynamic behaviour can be controlled by using the $STABILITY_THRESH$, MAX_CLUS_SIZE and MIN_CLUS_SIZE parameters.

The proposed approach tries to maintain a stable cluster topology by considering a ‘node age’ based stability metric and needs little resources in terms of computational effort.

ACKNOWLEDGMENTS

We thank Prof. V. S. Borkar (TIFR) and Prof. D. Manjunath (IIT, Bombay) for many discussions. The

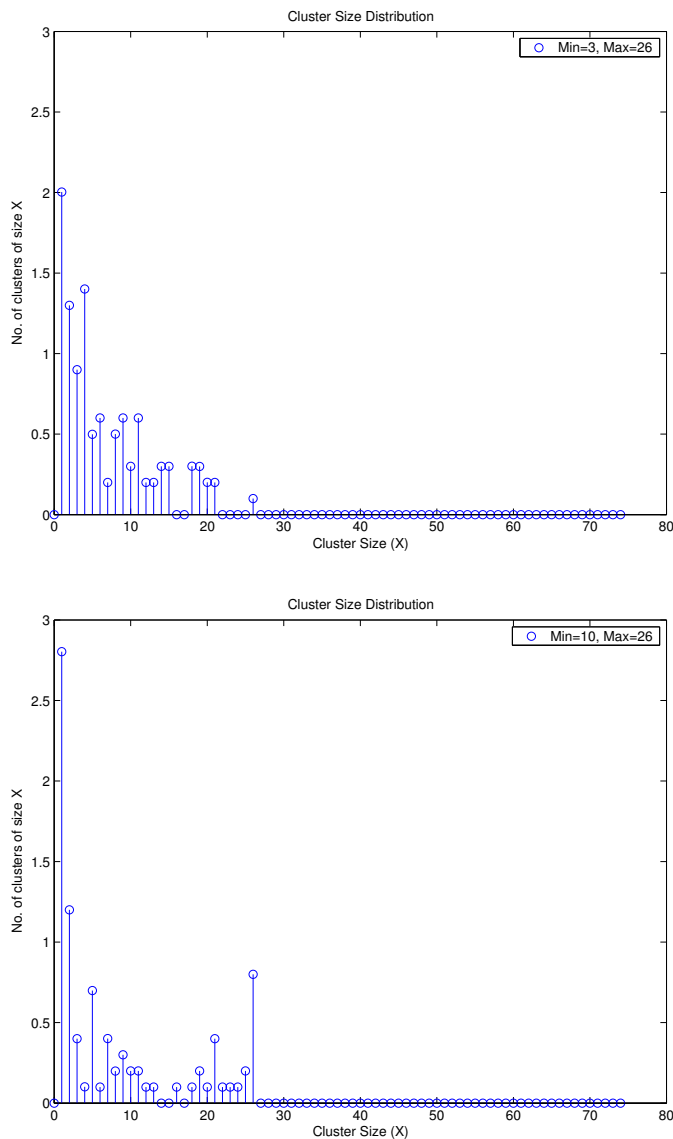


Fig. 4: Cluster Size Distribution (MAX_CLUS_SIZE=26)

work was done under support from the project *Futuristic Technologies: Security of Information and Network Infrastructures* from the Ministry of Information Technology, Govt. of India.

REFERENCES

- [1] A. B. McDonald and T. F. Znati, *A Mobility-Based Framework for Adaptive Clustering in Wireless Ad Hoc Networks*, IEEE Journal on Selected Areas in Communications, Vol. 17, No. 8, pp. 1466-1487, Aug. 1999.
- [2] C. R. Lin and M. Gerla, *Adaptive Clustering for Mobile Wireless Networks*, IEEE Journal on Selected Areas in Communication, Vol. 15, No. 7, pp. 1265-1275, Sept. 1997.
- [3] R. O. Duda, P. E. Hart and D. G. Stork, *Pattern Classification*, 2nd ed., New York: John Wiley & Sons, 2001.
- [4] S. Sivavakeesar and G. Pavlou, *A Prediction-Based Clustering Algorithm to Achieve Quality of Service in Multihop Ad Hoc Networks*, London Communication Symposium, 2002, <http://www.ee.ucl.ac.uk/lcs/papers2002/>.
- [5] S. Basagni, *Distributed Clustering for Ad Hoc Networks*, Proceedings of the 1999 International Symposium on Parallel Architectures, Algorithms and Networks (I-SPAN'99), IEEE Computer Society, pp.310-315, Australia, Jun 23-25,1999.
- [6] B. Das, R. Sivakumar and V. Bhargavan, *Routing in Ad Hoc Networks Using a Spine*, Sixth International Conference on Computer Communications and Networks (ICCCN '97), Las Vegas, NV, Sept 22 - 25, 1997.
- [7] S. Basagni, I. Chlamtac, A. Faragó, *A Generalized Clustering Algorithm for Peer-to-Peer Networks*, Workshop on Algorithmic Aspects of Communication, satellite workshop of ICALP'97, invited paper, Bologna, Italy, Jul 11-12, 1997.
- [8] N. H. Vaidya, P. Krishna, M. Chatterjee and D. K. Pradhan, *A Cluster-based Approach for Routing in Dynamic Networks*, ACM SIGCOMM Computer Communication Review, Vol. 27, No. 2, Apr. 1997.
- [9] S. Chen and K. Nahrstedt, *Distributed Quality of Service Routing in Ad-Hoc Networks*, IEEE Journal on Selected Areas in Communication, Vol. 17, No. 8, Aug. 1999.
- [10] C. R. Perkins and P. Bhagwat, *Highly Dynamic Destination Sequenced Distance Vector Routing (DSDV) for mobile computers*, Proceedings of ACM SIGCOMM, pp. 234-244, Oct. 1994.
- [11] C. E. Perkins and E. M. Royer, *Ad-hoc On-demand Distance Vector Routing*, Proceedings of the Second IEEE Workshop on Mobile Computing Systems and Applications 1999, pp. 90-100, 25-26 Feb. 1999.
- [12] D. B. Johnson, D. A. Maltz, *Dynamic Source Routing in Ad Hoc Wireless Networks*, Mobile Computing, edited by Tomasz Imielinski and Hank Korth, Chapter 5, pp. 153-181, Kluwer Academic Publishers, 1996.
- [13] P. Sinha, R. Sivakumar and V. Bhargavan, *CEDAR: A Core Extraction Distributed Ad Hoc Routing Algorithm*, IEEE Infocom 99, New York, NY, Mar. 1999.
- [14] M. Grossglauser and D. Tse, *Mobility Increases the Capacity of Ad-Hoc Wireless Networks*, IEEE Transactions on Information Theory, Vol. 48(6), Jun. 2002.