Dynamic Cluster Management In Ad hoc Networks

Puneet Sethi
Ubinetics India Pvt. Ltd.,
Bangalore - 560 027
e-mail: `Puneet.Sethi@ubinetics.co.in`

Gautam Barua
Department of Computer Science and Engineering,
Indian Institute of Technology Guwahati, Guwahati
- 781039
e-mail: `gb@iitg.ernet.in`

Abstract—**Clustering in ad hoc networks provides significant support for implementation of QoS and security, by overcoming inherent network deficiencies (like lack of infrastructure etc.). In a clustered network architecture, the ad hoc network is divided into groups of nodes called** *clusters.* **The clusters are dynamically maintained and reconfigured using specific protocols and algorithms. In this paper, we describe a clustering technique which has been designed as a part of the CRESQ([12]) routing algorithm for ad hoc networks. CRESQ's clustering is segregated into two parts, the** *initial clustering* **and the** *cluster management*. **Initial Clustering creates the clusters during the network formation and cluster management maintains the clusters throughout the network lifetime. The main focus of this paper is the cluster management algorithm which is designed to be distributed in nature, adaptive to practicalities such as loss of packets and tolerant to mobilities during the algorithm execution. The algorithm makes no assumptions about the availability of information concerning transmission range, node mobilities etc. This paper also discusses certain performance metrics for judging the efficiency of clustering in ad hoc networks. We also present simulation results for the cluster management algorithm, along with a comparison with some existing algorithms. Results showing a comparison of CRESQ with existing protocols in terms of** *packet drop ratio* **and** *routing overhead* **are also provided**.

Keywords: **clusters, routing, QoS.**

I. Introduction

An ad hoc network is made up of mobile nodes with wireless interfaces, which communicate with each other over multihop paths, in the absence of any fixed infrastructure. Each mobile node in the network also acts as a router for the network traffic. The main advantages of such networks are rapid deployment and dynamic reconfiguration, which make them the right candidate for military applications, rescue operations and disaster recovery. Owing to their ability to provide a quick and cheap communication link without the need of wired infrastructure, they help in extending the limits of ubiquitous computing and information access for their users. Of late, there has been a need to support QoS and security in ad hoc networks owing to their varied usage scenarios and convergence of different applications' traffic. Clustering provides a solution to support QoS in ad hoc networks, which are usually constrained by low computational and bandwidth capacity of nodes, mobility of intermediate nodes in an established path, and absence of routing infrastructure.

In a clustered network architecture, the whole network is divided into self-managed groups of nodes called *clusters*. All the nodes within a cluster are at most two hops away from each other. These clusters continually adapt themselves to the changing network topology and new cluster configurations that are feasible with the current network topology, are created dynamically. *Master* (or *Clusterhead)* is the node which is only one hop away from all the other nodes in the cluster, and carries certain extra responsibilities.

Much of the existing literature([11], [5], [15], [14], [10]) in this field focuses on the efficient partitioning of a network. The lack of an efficient complete and practical solution for ad hoc networks, particularly concerning the integration of clustering with routing, motivated a study in this area. Clustering in an ad hoc network can be divided into two phases: cluster initialization i.e. the creation of the clusters and cluster management i.e. the maintenance of clusters during the network lifetime. Most of the literature only focuses on the efficiency of the first phase i.e. cluster initialization. The cluster maintenance in [5] is handled by updates; the mobile node checks for the presence of its clusterhead and cluster-partners before making an update, which generates a considerable overhead. The schemes described in [14], [15], [10] assume the availability of, information about node's mobility and transmission range, feedback about the power of received signal, and guaranteed delivery of packets. Such assumptions are very difficult to ensure in a practical implementation. The cluster management in [10] also assumes that the mobile node is aware of its trajectory and its cluster head can predict its future clusterhead. A distributed network partitioning technique is described in ACRPN([7]), which is also used as the initial clustering algorithm in our simulations. ACRPN also describes a quorum system to avoid degradation in performance during frequent cluster changes. Most of the existing schemes fail to deal with issues like maintaining consistent cluster information in the nodes, packet drops, overloading of nodes' resources (due to unbounded cluster size), node movements during algorithm execution.

In this paper we present a clustering technique which is a part of the CRESQ([12]) protocol. We attempt to provide a clustering solution which is distributed in nature, handles the cluster management by taking into account practicalities like packet losses etc. and easily integrates with a routing module. *C*luster based *R*outing for *E*nd-to-end *S*ecurity and *Q*uality of service satisfaction i.e. CRESQ, is a cluster based source routing protocol, which provides adequate support for QoS and security. The clustering in CRESQ is handled by two algorithms, *initial clustering* algorithm and *cluster management* algorithm. The initial clustering algorithm creates the clusters during the formation of the network and the cluster management algorithm maintains the clusters mainly using periodic transmission of HELLO packets. In this paper, we focus on the cluster management algorithm and its evaluation, and other details of CRESQ can be found in [12]. In the next section, we describe the clustering algorithm. In Section III, we discuss certain performance metrics, for clustering in ad hoc networks. In Section IV, we present the results of simulations and comparisons of the cluster management algorithm with some existing algorithms. In Section V, a conclusion of the work is provided.

II. Clustering Methodology

The proposed cluster management algorithm was designed as a part of the CRESQ ad hoc routing protocol. Clustering in the CRESQ protocol helps in, reducing routing overhead, providing QoS guarantees and reducing QoS re-establishment delay during route failures. The clustering in CRESQ is accomplished by two algorithms namely initial clustering algorithm and the proposed cluster management algorithm. Initial clustering creates the clusters in the ad hoc network at a time when the wireless capable nodes are discovering each other and the cluster management algorithm maintains the clustered architecture by continually adapting to the changing network topology.

A. Terms

- *Cluster* - It refers to a collection of nodes, grouped for the functioning of CRESQ.
- *Master* - Every cluster is characterized by a unique node called its *master.* It has certain extra responsibilities.
- *Bridge* - *Bridge* is a node which belongs to more than one cluster. It thus has more than one master.
- *Slave* - All the cluster nodes other than bridges and master, are called *slaves.* Each *slave* has only one master (denoted by M_S for *slave* S). and hence belongs to only one cluster.
- *State* - A node's *state* describes whether the node is a slave, bridge, master or *none* (*none* means the node is uninitialized, i.e. it does not belong to any cluster).We will also refer to a node as slave, if its *state* is *slave* (similarly for bridge, master, none).

B. Overview of CRESQ

An overview of the CRESQ algorithm is as follows:
1. Initially all the nodes in the network are put in state *none*. Then the whole ad hoc network is clusterized using an initial clustering algorithm described in section II-C.
2. Once the initial clustering is done, then the cluster management algorithm takes over, which is described in section II-D.
3. Whenever a node S wants to establish a connection to a node D, S sends a ROUTE REQUEST packet to its master M_S specifying the destination and required QoS levels. M_S then broadcasts the packet to its bridges.
4. All the bridges of M_S further broadcast the packet to all of their masters, which further broadcast to their bridges. This process continues till a master of D hears the request, which adds D's address in the *Addr_List*[1] of a ROUTE REPLY packet and sends it to the node from which it heard the ROUTE REQUEST.
5. Each of the nodes which receive the ROUTE REPLY packet, sends it to the node from which it heard the ROUTE REQUEST after checking the QoS constraints and adding in the *Addr_List*, its own address (if it is a bridge) or a slave's address (if it is a master). (See section II-B.1)
6. When S receives the ROUTE REPLY, it simply uses the path specified in *Addr_List* to route its data packets. A data packet is routed by source routing at the intermediate nodes.
Note: ROUTE REQUEST loops are avoided by suitable mechanisms. Other details have been omitted in order to maintain clarity.

B.1 Route Discovery & Establishment

Whenever S wants to transmit data to D and does not have a cached route to D, it initiates the route discovery process by sending the ROUTE REQUEST packet. If S is a slave, then it unicasts the ROUTE REQUEST packet to M_S. If S is a bridge or a master or a none, then it broadcasts the ROUTE REQUEST packet. ROUTE REQUEST is flooded by every receiving bridge and master (except M_D i.e. master of D). On receiving the ROUTE REQUEST from A, M_D sends a ROUTE REPLY to A after adding D's address in Addr_List. Every bridge which receives the ROUTE REPLY forwards it to the node from which it heard the ROUTE REQUEST after adding its own address in the Addr_List, if it can provide the QoS. Every master which receives the ROUTE REPLY, forwards it to the previous hop of the ROUTE REQUEST, after adding the address of a slave which can provide the QoS. In this manner ROUTE REPLY travels back to the source S.

After transmitting the ROUTE REQUEST, S waits for REPLY_WAIT_TIME seconds. If it does not receive a ROUTE REPLY within that period, then it may chose to re-transmit ROUTE REQUEST or to inform the upper QoS layer about the network's inability to provide the required QoS, which may then suggest lesser QoS levels.

[1]*Addr_List* is a field in the ROUTE REPLY packet, which will eventually contain the list of nodes to be traversed in order to reach D.

B.2 Route Maintenance

The Route Maintenance process helps in correcting failed routes. Usually, a route failure is caused by the movement of an intermediate node, hence the route is broken only due to a change in a small fraction of the whole route. Unlike other protocols like AODV([3]), DSR([6]) in which the source is informed of the failure and it may have to redo the route discovery hence causing a considerable delay, CRESQ tries to minimize the QoS re-establishment delay by a local perturbation. Whenever a link from A to B fails on a data packet, A informs one of its own masters M using a FAILURE packet. M tries to find a substitute node C which can replace B for that connection and informs A using a FAILURE REPLY packet. If M is unable to find a substitution then it may advise A to send the ROUTE ERROR packet back to the source.

C. Initial Clustering Algorithm

Whenever mobile nodes come into contact with each other to form an ad hoc network, they are in *none* state. Each node has to be in one of the other three states in order to be accessible by the rest of the network. This is accomplished by the initial clustering algorithm. However, this does not place any restriction on the addition of nodes to the network, and new nodes are handled appropriately by the cluster management algorithm, if they enter with state *none*.

The initial clustering algorithm is expected to minimize the number of masters in the network, keep the number of bridges at an optimum level, spread the masters evenly over the topography and, be computationally inexpensive and distributed in nature. Any initial clustering algorithm can be used (in fact we may not use any initial clustering algorithm, and let the cluster management algorithm take care of everything), but a good choice of initial clustering algorithm greatly improves the performance of CRESQ and clustering in particular. We suggest the following clustering algorithm, which is used in our simulations:

This distributed algorithm[7] is executed at each network node, after it has discovered its neighbours and two-hop neighbourhood information is available with each node. The algorithm is as follows (Initially all nodes are put in state *none*):

1. If *degree(u)* (i.e. number of neighbours) is higher than that of all its neighbours, then *u* becomes a master and picks MAX_MASTERDEG[2] number of lowest degree neighbours as slaves. It also checks if the *effective degree* (i.e. number of neighbours with state *none)* of any of the remaining neighbours is zero. If so, then that node is made a master.
2. Else
The following steps are repeated for every higher degree neighbour, *v*.
 (a) If *degree(v)* is the highest in the neighbourhood of *v*,
 i. If *u* is among the MAX_MASTERDEG smallest degree neighbours of *v*, then *u* becomes a slave of *v*. Each lower degree neighbour *w* of *u*, which is of the greatest degree in the neighbourhood of *w*, is made a master.
 ii. Else *u* repeats the algorithm for the remaining none neighbours.
 (b) Else
 i. If the *degree(u)* is the highest in the neighbourhood of *u* (excluding the edge from *v* to *u)*, then *u* becomes a master.
 ii. If the *degree(v)* is the highest in the neighbourhood of *u*, then *u* becomes a slave of *v*.

Notes:

1. In Step 2, a node becomes a slave only if it does not become master by any other rule.
2. A node which receives a "become master" message always becomes a master, irrespective of its current state.
3. A node which becomes slave to more than one node, automatically get marked as bridge (and keeps number of masters less than MAX_MASTERS).

We can see that the algorithm tries to minimize the number of masters in step 1, where the node with the highest degree is made the master and it takes the lowest degree neighbours as its slaves. The cluster management algorithm is independent of the initial clustering algorithm. So, the network can also use a different initial clustering algorithm, which takes into account other information like position, mobility, range etc. provided that such information are available.

D. Cluster Management Algorithm

The cluster management algorithm can be divided into two parts for each node**:** one part is executed on triggering of *HelloEvent*[3] and the second is executed during the receipt of any packet. All the nodes in the cluster periodically transmit HELLO packets, and update their data structures on receiving packets from other nodes. Each node executes a different sequence of steps on triggering of HelloEvent and on receipt of any packet, depending upon its state. Each slave or bridge maintains a list of its masters and each master maintains lists of its slaves and bridges.

We present the cluster management algorithm, as different sequence of steps executed by the node, based on its state*:*

[2]maximum number of slaves that a master can have.

[3]HelloEvent is triggered in each node after every HELLO_INTERVAL seconds. Some randomization is also introduced between the triggering of two consecutive HelloEvent events.

Slave:

- On triggering of HelloEvent.
 1. If its master is still *alive*[4],
 (a) If it has not transmitted any packet in previous HELLO_INTERVAL[5] seconds,
 i. then broadcast a HELLO packet with *Hello_Type*[6] set to GENERAL_HELLO.
 2. Else,
 (a) mark self as none.
- On receiving a packet 'p' from node A,
 1. If A is its master,
 (a) then restamp master, to update the last time it was heard of.
 2. Else, if A is a master or a none,[7]
 (a) If 'p' is a HELLO packet and Hello_Type == BE_MY_SLAVE,
 i. then add A as one of its masters, and become a bridge.
 (b) Else, send a HELLO packet to A with Hello_Type set to CAN_I_BE_YOUR_SLAVE[8].

Bridge:

- On triggering of a HelloEvent,
 1. bridge removes the masters which have expired and changes its state to *slave* or *none*, if required.
 2. If it has not transmitted any packet in last HELLO_INTERVAL seconds,
 (a) then it broadcasts a HELLO packet with Hello_Type set to GENERAL_HELLO.
- On receiving a packet 'p' from A,
 1. If A is any one of its masters,
 (a) then restamp that master.
 2. Else, if A is a master or a none,
 (a) If 'p' is a HELLO packet and Hello_Type == BE_MY_SLAVE and numOfMasters[9] $<$ MAX_MASTERS,
 i. then add A as master.
 (b) Else, send a HELLO packet to A, with Hello_Type *set to* CAN_I_BE_YOUR_SLAVE.

Master:

- On triggering of a HelloEvent,
 1. master removes the slaves and bridges which have expired. If no slaves and bridges are left in its cluster, then state is changed to *none*.
 2. If it has not transmitted any packet in last HELLO_INTERVAL seconds,
 (a) then it broadcasts a HELLO packet with Hello_Type set to GENERAL_HELLO.
- On receiving a packet 'p' from node A,

[4]If no packet is received from a cluster partner for (1+ALLOWED_HELLO_LOSS) $\times$HELLO_INTERVAL seconds, then it is assumed to be dead or expired(and is expelled from the cluster).

[5]If the node has sent any packet (ROUTE REQUEST, ROUTE REPLY, DATA etc.) in previous HELLO_INTERVAL seconds, then it may not send the HELLO packet. This greatly reduces bandwidth wastage.

[6]This field in a HELLO packet, describes the type of the packet. It can take three values i.e. GENERAL_HELLO, CAN_I_BE_YOUR_SLAVE and BE_MY_SLAVE.

[7]Packets may contain information about node's state, list of its masters/slaves/bridges, and other information like QoS abilities.

[8]Nodes which are sent requests like CAN_I_BE_YOUR_SLAVE, are not resent similar requests for a particular duration.

[9]Variable that contains the number of masters that the bridge has currently.

```
1. If A is its own slave or bridge,
   (a) then restamp and update information about A's state and other details like QoS capabilities etc.
2. Else, if 'p' is a HELLO packet and Hello_Type == CAN_I_BE_YOUR_SLAVE and numOfSlaves[10] + numOfBridges[11] < MAX_MASTERDEG,
   (a) then, add A in the list of slaves/bridges, send a HELLO packet to A with Hello_Type set to BE_MY_SLAVE.
3. Else, if A is none,
   (a) then broadcast a HELLO packet with Hello_Type set to GENERAL_HELLO.
4. Else, if A is a master,
   (a) If 'p' is a HELLO packet and Hello_Type == BE_MY_SLAVE,
       i. then become a slave of A.
   (b) Else, if the receiving node has moved from its original cluster[12],
       i. then send a HELLO packet to A with Hello_Type set to CAN_I_BE_YOUR_SLAVE.
```

None:

```
• On triggering of a HelloEvent,
  1. Broadcast a HELLO packet, with Hello_Type set to GENERAL_HELLO.
  2. Increment numHelloTries.
• On receiving a packet 'p' from A,
  1. If 'p' is a HELLO packet and A is a master and Hello_Type == BE_MY_SLAVE,
     (a) then become a slave, and add A as master.
  2. Else, if A is master,
     (a) then send a HELLO packet to A with Hello_Type set to CAN_I_BE_YOUR_SLAVE.
  3. Else,
     (a) If A is a slave or bridge or none and 'p' is a HELLO packet and numHelloTries > MAX_HELLO_TRIES and Hello_Type == CAN_I_BE_YOUR_SLAVE,
         i. then, mark self as master, add the node as bridge and send a HELLO to A with Hello_Type set to BE_MY_SLAVE.
     (b) Else, if A is none,
         i. then send a HELLO to A with Hello_Type set to CAN_I_BE_YOUR_SLAVE.
```

The above algorithm is run at each node. Every node avoids transmitting HELLO packets, if it is able to make its presence felt through some other packets. Whenever a packet is received from a node S, which is not a member of the cluster, then appropriate steps contingent upon S's state are taken to include S in the cluster. It can be seen that a node goes into *none* state, when it discovers that it no more belongs to any cluster.

The efficiency of the cluster management algorithm can be greatly improved with the help of lower layers. Whenever lower layers like 802.11 report link failure to the routing layer, the node can use this information to identify the present state of the cluster. This part of the cluster management algorithm can be clubbed with the Route Maintenance process.

[10] Variable that contains the number of slaves that the master has currently.

[11] Variable that contains the number of bridges that the master has currently.

[12] Detecting whether the receiving node has moved or A has moved, is done on the basis of constituent nodes of the cluster and their timestamps, as reflected by the current data structures of the two nodes. Details can be found in [12], [13].

The cluster management algorithm can be made sensitive to the needs of routing. For example when a data source or an intermediate becomes none, then the value of MAX_HELLO_TRIES can be decreased for that node. Also, when a slave A of a master M is handling a connection, then a required change in the state of M can be postponed until M comes to know that A is out of reach .

III. Performance Metrics

We discuss the following criteria, for judging the performance of a clustering algorithm in ad hoc networks. The discussion encompasses description of the metrics' relevance in judging both the efficiency and optimality of clustering algorithms for use with ad hoc routing protocols.

Number of masters

One of the important ways to judge the efficiency of a cluster management algorithm will be through the number of masters it creates. The number of masters in an ad hoc network considerably affects the performance of a routing protocol. If the number of masters is large, then the average cluster size will be small hence inhibiting the routing protocol from exploiting the advantages of clustering such as minimal route discovery and recovery delays. If the number of masters is small, then it actually overloads the resources of a master which now has to serve more nodes. It is also important to study the dependence of the number of masters to changes in node mobility, transmission range, and number of nodes. Clustering algorithms usually tend to make more masters during higher node mobility, greater number of nodes and lesser transmission range.

Number of Cluster Changes

The number of cluster changes refers to the total number of times any node in the network changes its cluster. This parameter determines the stability of the clustered architecture. A higher value for this parameter is usually observed in scenarios of high node mobility and moderate transmission range. Ideally a lower value of this parameter is required for better performance of routing protocols, but in order to accomplish this certain algorithms([2]) try to postpone the cluster change. This results in an inefficient clustered architecture and prohibits the discovery of efficient routing paths. The CRESQ's cluster management algorithm avoids the cluster change only if it is desired (explained in section II-D, last paragraph).

Inaccessibility time

This parameter refers to the total duration for which any network node remains inaccessible for the rest of the network. The node becomes inaccessible in two situations: when no other node is in its range, and when it is going through a transition period. The transition period refers to the period starting from the point when a node discovers that it has left its cluster to the point when it joins a new cluster. In the scenarios of limited transmission range, most of the existing algorithms designate the inaccessible node as a master. This is not desirable, with regard to the integration with a routing module. When a node's state is *master* (or *bridge* or *slave*), it implicitly carries a meaning that it is a part of the whole cluster configuration. This may lead to incorrect interpretations at the routing layer and may prevent the routing protocol from taking any corrective steps even if it so desires. This may also lead to inefficient clustering. The node could be made to wait instead of being designated as a master, so that an efficient clustering is made later. The second type of inaccessibility due to transition is largely ignored by the current algorithms, but it remains a practical problem. In CRESQ's cluster management algorithm we introduce a state *none* and a node in state *none* means that it is not accessible by the network.

IV. Results and Discussions

We have implemented the CRESQ routing module in the ns-2([9]) simulator. The values of parameters such as MAX_MASTERDEG, MAX_MASTERS, HELLO_INTERVAL etc. were estimated to better the performance of the routing and details of these simulations can be found in [12]. Here, we focus on the simulations results for evaluation of the performance of CRESQ's cluster management algorithm. However to provide a glimpse of the total picture, we also provide a comparison of CRESQ's performance with other protocols.

In all our simulations IEEE 802.11([8]) was used as the MAC layer and the radio model used characteristics similar to a commercial radio interface, Lucent's WaveLAN([1]).

A. Movement and communication model

Our simulations were run for N(30, 50, 100) nodes in a L$\times$ L(L=100 to 800) area, and the movement model used was the random way-point model. In this model, nodes are initially placed at random positions and then they start moving towards a random destination with a speed ranging from 0 to a maximum value. On reaching the destination, they wait for a *pause time* and again select a new destination. Nodes' mobility vary inversely with the pause time.

B. Number of Masters

We ran simulations to evaluate the efficiency of the algorithm in terms of average number of masters, in scenarios of varying mobility, number of nodes and transmission range. We present the results for network of 50 nodes in L $\times$ L (L = 600 m) for studying the dependence on node mobility. Figure 1 shows the number of masters that exist in the network at different times, in a moderate mobility scenario. It also shows the number of masters that would be created if instead of the cluster management algorithm, initial clustering algorithm is made to run at each time click. The node mobility chosen for this simulation was chosen to be moderate (pause time set to 60). For the first 60 seconds of the network lifetime, there is no deviation shown by the cluster management algorithm from the initial clustering algorithm. However, afterwards it starts showing deviation as the nodes start to move towards their random destinations, hence causing changes in clusters. Figure 2 shows the average number of masters that are created in the network at different node mobilities. It can be seen that clustering becomes efficient as the mobility decreases. Figure 2 also shows the deviation in the number of masters that the cluster management algorithm shows, from a hypothetical situation in which instead of the cluster management algorithm, the initial clustering algorithm will run at each instant of evaluation. We can see that the deviation vanishes as the mobilities decrease and the network becomes more stable. Figure 3 shows the dependence of the number of masters on the number of nodes in the network. The number of masters increase almost linearly with the increase in the number of nodes. An important observation is that the deviation also increases with the number of nodes, but the deviation per node approximately remains the same. In these simulations, as the number of nodes increased we also increased the value of L, so that the density of nodes per unit area remains the same as that of the earlier 50 node scenario.

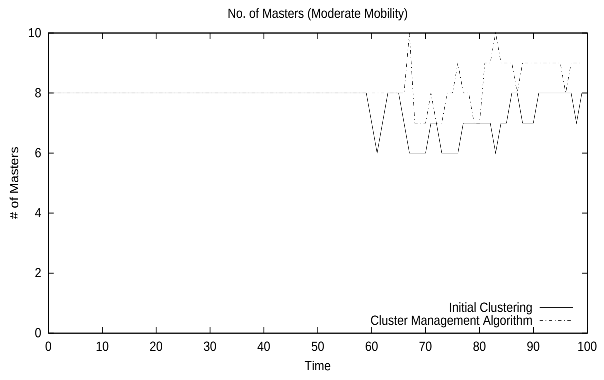

Fig. 1. Performance of cluster management algorithm in moderate mobility.

Figure 5 shows the variation of the number of masters with respect to transmission power and it also gives a comparison with some existing algorithms. CRESQ's cluster management algorithm places an upper bound on the number of slaves/bridges in a cluster i.e. MAX_MASTERDEG, which limits the cluster size. An upper bound on the number of masters for bridges is also placed. These upper bounds were placed keeping in mind the practicalities like nodes' resources etc. However, we can see that CRESQ with no upper bounds outperforms the Clusterhead scheme([11]), which also does not have any bounds. Bounded CRESQ's cluster management also performs as expected, as it creates approximately five masters for 30 nodes (MAX_MASTERDEG =9, so even in the best possible scenario three masters will be made) in the regions of interest(moderate transmission range). We can see that WCA([10]) which also has a bounded cluster size in terms of a load balancing factor, has lesser number of masters than CRESQ's algorithm. WCA outperforms CRESQ because it assumes availability of information regarding nodes' transmission power and mobilities. It is also to be noted that we can obtain better results for CRESQ, if we assume guaranteed packet delivery as other schemes do. An important observation is that lesser number of masters are created by CRESQ's algorithm in regions of low transmission range. This does not mean that CRESQ's cluster management algorithm is able to form clusters, but it implies that it keeps the nodes in the *none* state instead of designating them as masters. This is corroborated by figure 6, where it is shown that nodes spend considerable time in the *none* state, if the transmission range is less. These simulations depicting the comparisons were run for 30 nodes in 100 $\times$ 100 m area, with near zero pause time.

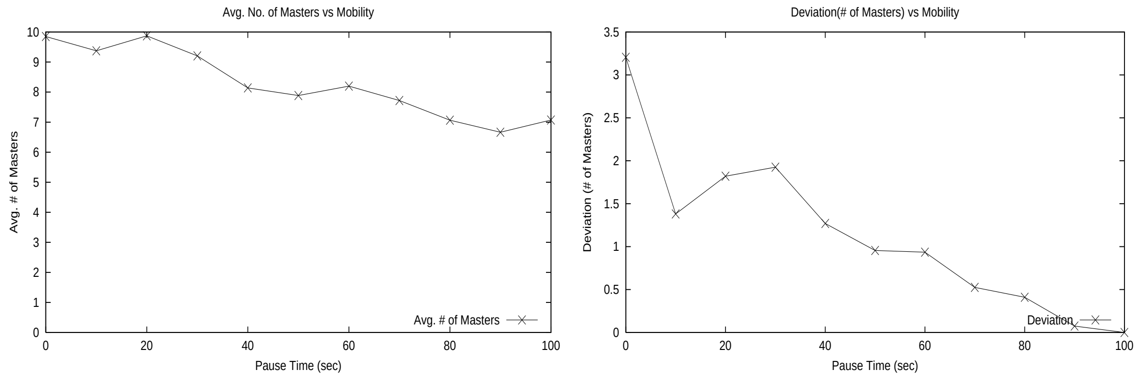

Fig. 2. No. of masters Vs. Node mobility.

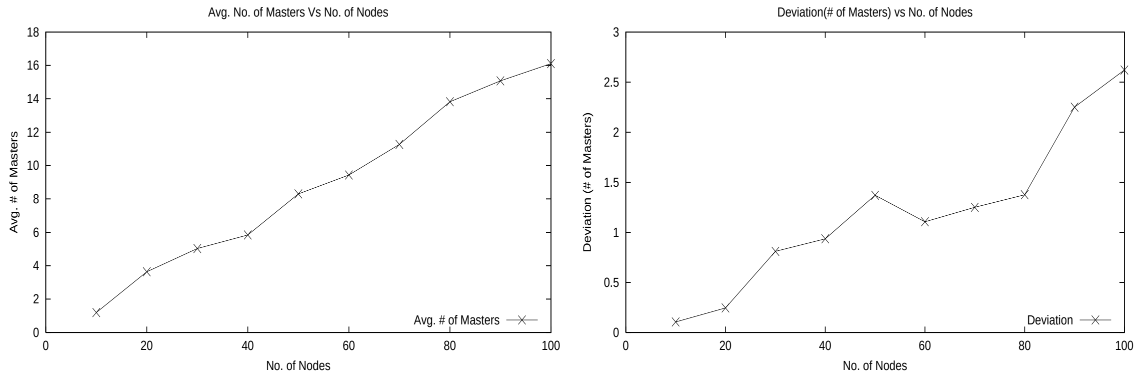

Fig. 3. No. of Masters Vs No. of Nodes.

C. Cluster Changes

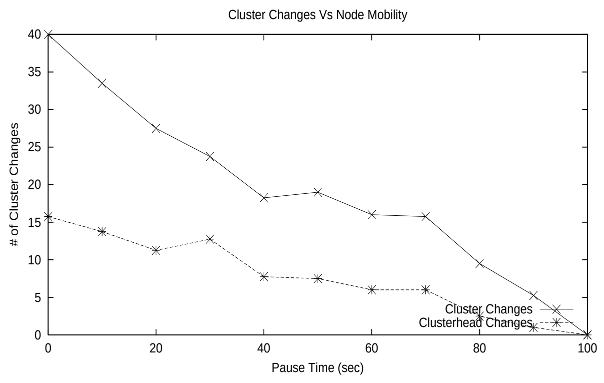

Fig. 4. Cluster Changes Vs Node mobility.

Figure 4 shows the variation in cluster changes and cluster head changes with respect to node mobility. We can see that lesser number of changes occur in lower mobilities, as the network is more stable in those scenarios. The simulations for these results were carried with N = 50 and L = 600. Figure 5 shows a comparison with existing clustering algorithms i.e. Adaptive technique([5]) and Clusterhead scheme([5], [11]). We can see that CRESQ's cluster management provides a much more stable architecture than the other two schemes. The simulation scenario chosen, was similar to the one chosen for comparison in terms of number of masters.

D. Inaccessibility Time

Inaccessibility time for a node denotes the period for which it is not accessible by the network and in the CRESQ's case it will actually refer to the time spent in *none* state. From figure 6, we can see that inaccessibility decreases with lower mobility and with greater transmission range. The reasons for inaccessibility are however, different in both the cases. In a greater mobility scenario, node transitions make the major contribution to inaccessibility but in a lesser transmission range case, it is the limitation of range which is the cause.

E. Comparison of CRESQ with existing routing protocols

Figure 7 shows a comparison of CRESQ performance with routing protocols like AODV([3]), DSR([6]), DSDV([4]) and TORA([16]). It is also important to note that for cluster maintenance CRESQ uses only one HELLO packet per second per node in the worst case (when no other packets are transmitted in the network).The same number of packets are used by protocols like AODV to maintain their list of neighbours. There is no extra overhead in CRESQ which is also depicted by its efficient performance. In terms of routing overhead, CRESQ outperforms all others due to its clustering which limits the flooding of ROUTE REQUESTS. In terms of packet drop ratio, CRESQ, AODV and DSR perform better than others in low mobilities. In moderate mobilities CRESQ outperforms DSR(even though both are source routing) due to its ability to locally correct routes. Further details of the comparison can be found in [12], [13].

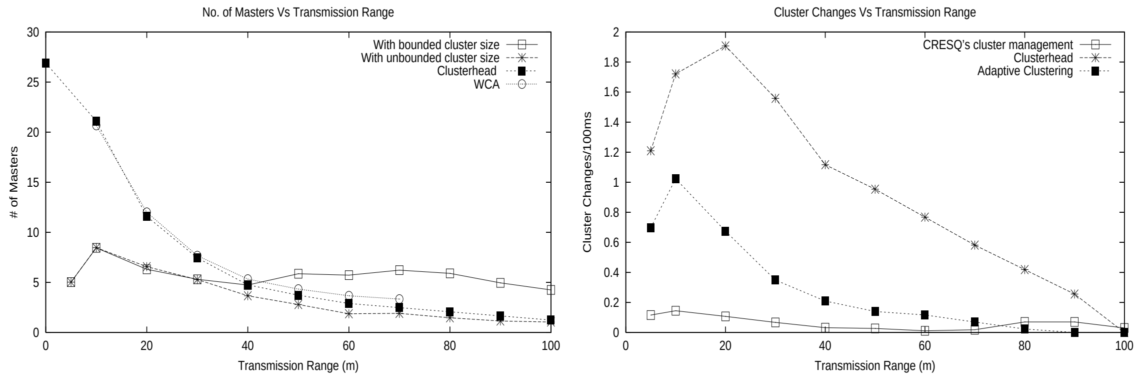

Fig. 5. Comparison with existing algorithms

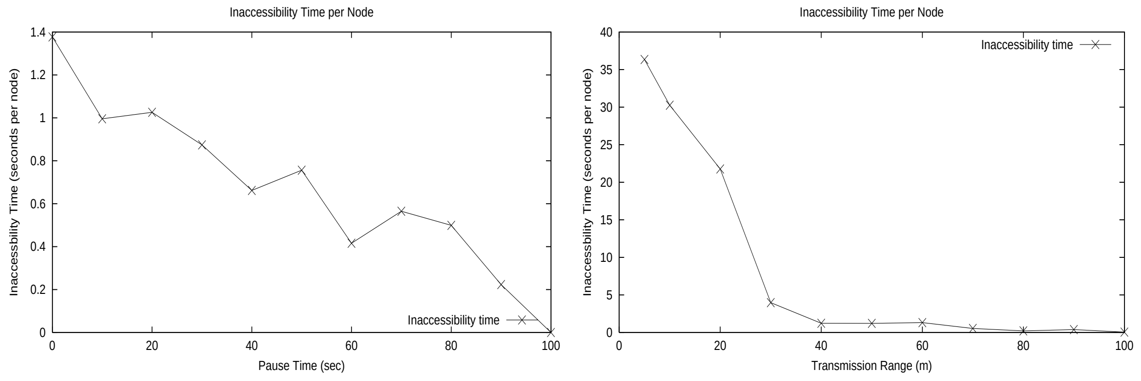

Fig. 6. Inaccessibility Time

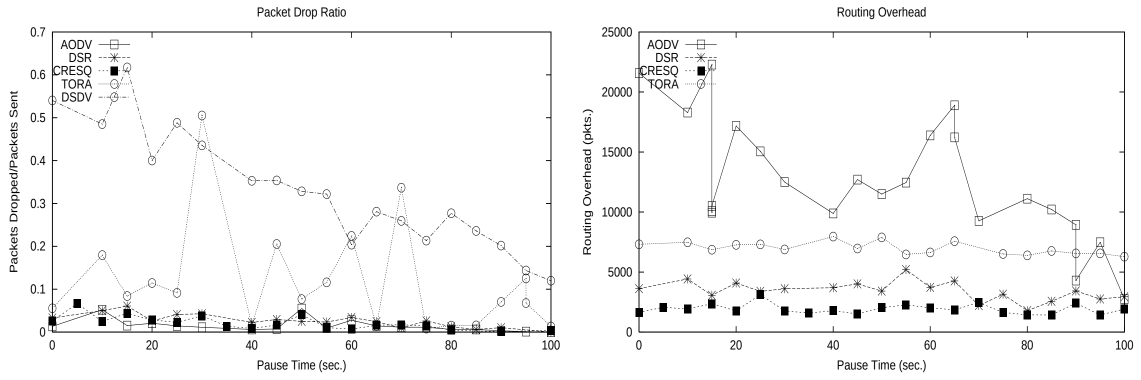

Fig. 7. Comparison of CRESQ with existing protocols

V. Conclusion

We have presented a new cluster management algorithm, which has been designed to suit the needs of ad hoc networks. It has been specifically designed as a part of the CRESQ routing module, which supports routing with QoS and security in ad hoc networks. The proposed cluster management algorithm is distributed in nature, computationally inexpensive and has low overhead (as explained in Section IV-E). The cluster management algorithm makes no assumptions about the transmission range and nodes' mobilities and is also tolerant to packet losses and node movements during algorithm execution. The algorithm has been integrated with the CRESQ routing module and has been implemented in the ns simulator. Simulations done with the clustering algorithm and CRESQ routing protocol prove the worth of the scheme and the routing protocol. CRESQ's clustering algorithm creates lower number of masters than the existing algorithms, which helps in reducing the latency during route discovery. It also provides a much more stable clustered architecture than the existing schemes.

REFERENCES

[1] Bruce Tuch. Development of WaveLAN, as ISM band wireless LAN. AT&T Technical Journal, 72(4):27-33, July/Aug 1993.

[2] C.-C. Chiang, H.K. Wu, W. Liu and M. Gerla, "Routing in Clustered Multihop, Mobile Wireless Networks", In the *Pro-*

ceedings of GLOBECOM'98, Nov. 1998, pages 1817 -1822.

[3] Charles E. Perkins, Elizabeth M. Royer, "Ad-hoc On-Demand Distance Vector Routing", *In the 2nd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'99).*

[4] Charles E. Perkins, Pravin Bhagwat, "Highly dynamic Destination-Sequenced Distance-Vector routing (DSDV) for mobile computers", *In the Proceedings of SIGCOMM'94 Conference on Communications Architectures, Protocols and Applications, August 1994.*

[5] C. R. Lin and M. Gerla, "Adaptive Clustering for mobile wireless networks", *Journal on Selected Areas in Communications 15,* 7(Sept. 1997), pages 1265-1275.

[6] David B. Johnson, David A. Maltz, John Broch, "DSR: The Dynamic Source Routing for Multihop Wireless Ad hoc Networks", *Adhoc Networking 2001.*

[7] I. Chakraborty, G. Barua, "ACRPN: A New Routing Protocol for Adhoc Wireless Networks", *Unpublished Manuscript, Indian Institute of Technology Guwahati 2001.* http://www.iitg.ernet.in/engfac/gb/acrpn.pdf

[8] IEEE Standards Department. Wireless LAN medium access control (MAC) and physical layer (PHY) specifications, IEEE standard 802.11-1997, 1997.

[9] Kevin Fall and Kannan Varadhan, editors. ns notes and documentation. The VINT project, UC Berkeley, LBL, USC/ISI and Xerox Parc November 1997. http://www.isi.edu/nsnam.

[10] M. Chatterjee, S. K. Das and D. Turgut, "WCA: A Weighted Clustering Algorithm for Mobile Ad hoc Networks", *Journal of Cluster Computing,* Special Issue on Mobile Ad hoc Networking, No. 5, 2002, pages 193-204.

[11] M. Gerla and J.T.C Tsai, "Multicluster, Mobile, Multimedia Radio Network", *Wireless Networks 1,* 3 (1995), pages 255-265.

[12] Puneet Sethi, "CRESQ: Providing QoS and Security in Ad Hoc Networks", *Undergraduate Thesis, Department of Computer Science and Engineering, Indian Institute of Technology Guwahati,* May 2002, http://www.iitg.ernet.in/engfac/gb/cresq.pdf

[13] P. Sethi, G. Barua, "CRESQ: Providing QoS and Security in Ad hoc networks", *Unpublished Manuscript, Indian Institute of Technology Guwahati 2001.* http://www.iitg.ernet.in/engfac/gb/cresq_router.pdf.

[14] S. Basagni, "Distributed and Mobility-Adaptive Clustering For Multimedia Support in Multi-hop Wireless Networks", In the *Proceedings of Vehicular Technology Conference,* VTC 1999-Fall, Vol.2, pages 889-893.

[15] S. Basagni, "Distributed Clustering for Ad hoc Networks", Proceedings of International Symposium on Parallel Architecture, Algorithms and Networks, June 1999, pages 310-315.

[16] Vincent D. Park, M. Scott Corson, "Temporally-Ordered Routing Algorithm(TORA) version 1: Functional Specification", INTERNET-DRAFT, *draft-ietf-manet-tora-spec-00.txt, November 1997.*

[17] http://www.bluetooth.com/