Load-Balancing Clusters in Wireless Ad Hoc Networks

Alan D. Amis Ravi Prakash
Department of Computer Science
University of Texas at Dallas
Richardson, Texas 75083-0688
Email: aamis@telogy.com, ravip@utdallas.edu

Abstract

Ad hoc networks consist of a set of identical nodes that move freely and independently and communicate with other node via wireless links. Such networks may be logically represented as a set of clusters by grouping together nodes that are in close proximity with one another. Clusterheads form a virtual backbone and may be used to route packets for nodes in their cluster. Nodes are assumed to have non-deterministic mobility pattern. Clusters are formed by diffusing node identities along the wireless links. Different heuristics employ different policies to elect clusterheads. Several of these policies are biased in favor of some nodes. As a result, these nodes shoulder greater responsibility and may deplete their energy faster, causing them to drop out of the network. Therefore, there is a need for load-balancing among clusterheads to allow all nodes the opportunity to serve as a clusterhead. We propose a load-balancing heuristic to extend the life of a clusterhead to the maximum budget before allowing the clusterhead to retire and give way to another node. This helps to evenly distribute the responsibility of acting as clusterheads among all nodes. Thus, the heuristic insures fairness and stability. Simulation experiments demonstrate that the proposed heuristic does provide longer clusterhead durations than with no load-balancing.

1. Introduction

Ad hoc networks (also referred to as packet radio networks) consist of nodes that move freely and communicate with other nodes via wireless links. One way to support efficient communication between nodes is to develop a wireless backbone architecture [1, 2, 3, 6]. While all nodes are identical in their capabilities, certain nodes are elected to form the backbone. These nodes are called clusterheads and gateways. Clusterheads are nodes that are vested with the responsibility of routing messages for all the nodes within their cluster. Gateway nodes are nodes at the fringe of a cluster and typically communicate with gateway nodes of other clusters. The wireless backbone can be used either to route packets, or to disseminate routing information, or both.

Nodes in ad hoc networks are powered by batteries because of their mobile nature. Communications or transmissions cause the batteries to be depleted. Therefore, the amount of communications should be kept to a minimum to avoid a node dropping out of the network prematurely. Clusterhead batteries are depleted faster because they are usually involved in every communication within their cluster. Therefore, there is a need to distribute the responsibility of being a clusterhead to all nodes (load-balancing). The proposed heuristic provides load balancing among clusterheads to insure a fair distribution of load among clusterheads.

2. System Model

In an ad hoc network all nodes are alike and all are mobile. There are no base stations to coordinate the activities of subsets of nodes. Therefore, all the nodes have to collectively make decisions. All communication is over wireless links. A wireless link can be established between a pair of nodes only if they are within wireless range of each other. We will only consider bidirectional links. It is assumed the MAC layer will mask unidirectional links and pass only bidirectional links. Beacons could be used to determine the presence of neighboring nodes. After the absence of some number of successive beacons from a neighboring node, it is concluded that the node is no longer a neighbor. Two nodes that have a wireless link will, henceforth, be said to be 1 wireless hop away from each other. They are also said to be immediate neighbors. Communication between nodes is over a single shared channel. The Multiple Access with Collision Avoidance (MACA) protocol [13] may be used to allow asynchronous communication while avoiding collisions and retransmissions over a single wireless channel.

MACA utilizes a *Request To Send/Clear To Send (RTS/CTS)* handshaking to avoid collision between nodes.

Other protocols such as spatial TDMA [7] may be used to provide MAC layer communication. Spatial TDMA provides deterministic performance that is good if the number of nodes is kept relatively small. However, spatial TDMA requires that all nodes be known and in a fixed location to operate. In ad hoc networks the nodes within each neighborhood are not known *a priori*. Therefore, spatial TDMA is not a viable solution initially. We suggest that MACA be used initially for this heuristic to establish clusterheads and their associated neighborhoods. Then the individual cluster may transition to spatial TDMA for inter-cluster and intra-cluster communication.

All nodes broadcast their node identity periodically to maintain neighborhood integrity. Due to mobility, a node's neighborhood changes with time. As the mobility of nodes may not be predictable, changes in network topology over time are arbitrary. However, nodes may not be aware of changes in their neighborhood. Therefore, clusters and clusterheads must be updated frequently to maintain accurate network topology.

3. Previous Work and Design Choices

There are two heuristic design approaches for management of ad hoc networks. The first choice is to have all nodes maintain knowledge of the network and manage themselves [5, 9, 10]. This circumvents the need to select leaders or develop clusters. However, it imposes a significant communication responsibility on individual nodes. Each node must dynamically maintain routes to the rest of the nodes in the network. With large networks the number of messages needed to maintain routing tables may cause congestion in the network. Ultimately this traffic may generate huge delays in message propagation from one node to another. This approach will not be considered in the remainder of this paper.

The second approach is to identify a subset of nodes within the network and vest them with the extra responsibility of being a leader (clusterhead) of certain nodes in their proximity. The clusterheads are responsible for managing communication between nodes in their own neighborhood as well as routing information to other clusterheads in other neighborhoods. Typically, backbones are constructed to connect neighborhoods in the network. Past solutions of this kind have created a hierarchy where every node in the network was no more than 1 hops away from a clusterhead [1, 3, 7].

Furthermore, some of the previous clustering solutions have relied on synchronous clocks for exchange of data between nodes. In the Linked Cluster Algorithm [1], LCA, nodes communicate using TDMA frames. Each frame has a slot for each node in the network to communicate, avoiding collisions. For every node to have knowledge of all nodes in it neighborhood it requires $2n$ TDMA time slots, where n is the number of nodes in the network. A node x becomes a clusterhead if at least one of the following conditions is satisfied: (i) x has the highest identity among all nodes within 1 wireless hop of it, (ii) x does not have the highest identity in its 1-hop neighborhood, but there exists at least one neighboring node y such that x is the highest identity node in y's 1-hop neighborhood. Thus, LCA has a definite bias towards higher id nodes while electing clusterheads. A pathological case exists for LCA where a group of nodes are aligned in monotonically increasing order. In this case all of the nodes in the ordered sequence will become a clusterhead, generating a large number of clusterheads.

Later the LCA heuristic was revised [4] to decrease the number of clusterheads produced in the original LCA and to decrease the number of clusterheads generated in the pathological case. In this revised edition of LCA (LCA2) a node is said to be *covered* if it is in the 1-hop neighborhood of a node that has declared itself to be a clusterhead. Starting from the lowest id node to the highest id node, a node declares itself to be a clusterhead if among the *non-covered* nodes in its 1-hop neighborhood, it has the lowest id. So, LCA2 favors lower id node while electing clusterheads.

Definition 1 (d-hop Clusters) - *A d-hop cluster is one where no node in a cluster is more than d hops away from its clusterhead.*

The LCA heuristics were developed to generate 1-hop clusters and intended to be used with small networks of less than 100 nodes. In this case the delay between node transmissions is minimal and may be tolerated. However, as the number of nodes in the network grows larger, LCA will impose greater delays between node transmissions in the TDMA communication scheme and may be unacceptable. Additionally, it has been shown [11] that as communications increase the amount of skew in a synchronous timer also increases, thereby degrading the performance of the overall system or introducing additional delay and overhead.

The Max-Min heuristic [12] was developed to extend the notion of 1-hop clusters and generalizes cluster formation to d-hop clusters. The rules for Max-Min heuristic are similar to those for LCA but converges on a clusterhead solution much faster at the network layer, $2d$ rounds of messages exchanges. Once again a node x becomes a clusterhead if at least one of the following conditions is satisfied: (i) x has the highest identity among all nodes within d wireless hop of it, (ii) x does not have the highest identity in its d-hop neighborhood, but there exists at least one neighboring node y such that x is the highest identity node in y's d-hop neighborhood. Max-Min and LCA generate different solu-

tions because in case (ii) for Max-Min if a node becomes a clusterhead it will consume all nodes that are closer to it than any other elected clusterhead. This is a major difference between the two heuristics. However, like LCA, Max-Min also favors higher id nodes while electing clusterheads.

Other solutions base the election of clusterheads on degree of connectivity [8], not node id. Each node broadcasts the nodes that it can hear, including itself. A node is elected as a clusterhead if it is the highest connected node in all of the *uncovered* neighboring nodes. In the case of a tie, the lowest or highest id may be used. As the network topology changes this approach can result in a high turnover of clusterheads [6]. This is undesirable due to the high overhead associated with clusterhead change over. Data structures have to be maintained for each node in the cluster. As new clusterheads are elected these data structures must be passed from the old clusterhead to the newly elected clusterhead. Re-election of clusterheads could minimize this network traffic by circumventing the need to send these data structures. So Degree based heuristic has no bias towards any particular nodes while electing clusterheads.

We would like to combine the non-biased selection of Degree based heuristics with the stability of Max-Min and LCA(2) heuristics.

4. Contributions

The main objective was to develop an enhancement for existing heuristics to provide a contiguous balance of loading on the elected clusterheads. Once a node is elected a clusterhead it is desirable for it to stay as a clusterhead up to some maximum specified amount of time, or budget. The budget is a user defined constraint placed on the heuristic and can be modified to meet the unique characteristics of the system, i.e., the battery life of individual nodes. Some of the goals of the heuristic are:

1. Minimize the number and size of the data structures required to implement the heuristic,
2. Extend the clusterhead duration budget based on an input parameter,
3. Allow every node equal opportunity to become a clusterhead in time,
4. Maximize the stability in the network,

5. Load-Balancing (Node ID)

5.1. Data Structures

The data structures necessary for the heuristic consist of two local variables: Physical ID (PID), and Virtual ID (VID). The PID is the initial id given and is unique for each individual node. Initially, the VID is set equal to the PID for each node. However, this changes with time to represent the electability of a node. The VIDs for various nodes may be the same at certain times.

5.2. Basic Idea

The *node id* load-balancing heuristic operates on the principle of a circular queue. That is, the virtual ids of each non-clusterhead node cycles through the circular queue at a rate of 1 unit per run of the load-balancing heuristic[1]. The circular queue has a minimum value of 1 and a maximum value of MAX_COUNT. Upon reaching MAX_COUNT a node will rotate to a value of 1 on the next cluster election heuristic run. As the cluster election heuristics run they will use the VIDs to determine the clusterheads of the network. In cases where the VIDs are the same the PID is used as a tie breaker. Once a node is determined to be a clusterhead, its VID is promoted to a value larger than MAX_COUNT (MAX_COUNT + VID). A clusterhead will maintain this value until it has exhausted its clusterhead duration budget. At this point it will set its VID to 0, i.e., less than any other node, and become a normal node.

There are cases where two elected clusterheads A and B may move within close range of each other causing B to give way to A prior to B exhausting its clusterhead budget. In this case clusterhead B lowers its VID to the value it would have achieved had it not become a clusterhead. That is, B's VID just prior to becoming elected a clusterhead + the number of times the cluster election heuristic has run since B became a clusterhead. This will place B back into a place of high electability to insure a quick return as a clusterhead. One would probably immediately notice that this may not be the desired response. One of the main goals of this heuristic is to provide stability in the network. Therefore, we may not want node B to make a quick return as a clusterhead. Alternately, node B's VID may be set to 0 just as if it had successfully used its clusterhead budget. This should provide a more dampened response than having the clusterheads bounce back simply to use up their clusterhead budget.

5.3. Load-Balancing Pseudo Code

```
**********************************************
Clusterhead Load-Balancing
If a node is a clusterhead check to see if the budget is
exceeded. If it is then set the VID to 0 and become and
ordinary node, otherwise increment the VID value.
**********************************************
```

[1] For clarity, the load-balancing heuristic is an enhancement to the cluster election heuristic and will run whenever the cluster election heuristic is triggered to run.

```
cluster_load_balance()
if (Clusterhead == My_Node_Id)
        if (Budget >= Max_Budget)
            VID = 0;
            Clusterhead = FALSE;
            Budget = 0;
        else
            Budget += Work;
else
        ++VID;
```

6. Load-Balancing (Degree)

6.1. Data Structures

The data structures necessary for the heuristic consist of one local variable: Elected Degree. The Elected Degree represents the degree of the node when it was initially elected a clusterhead. This value will be used for comparison purposes described below.

6.2. Basic Idea

Because the Degree based heuristic elects clusterheads with a different policy than LCA and Max-Min heuristics, we must use a different load-balancing policy. The *degree* load-balancing heuristic monitors the amount of change, or delta, in the degree of an elected clusterhead from the time it is elected a clusterhead. That is, on each run of the heuristic the difference is take between the current degree of the clusterhead and the Elected Degree. If the absolute value of the difference exceeds an input value MAX_DELTA, then the clusterhead is demoted to an ordinary node. The Degree based heuristic, much like the LCA2 heuristic, does not allow for adjacent clusterheads. Therefore, there may be cases where an elected clusterhead will give way to another clusterhead based on the heuristic and not on exceeding the MAX_DELTA parameter.

6.3. Load-Balancing Pseudo Code

```
**********************************************
Clusterhead Load-Balancing
If a node is a clusterhead check to see if it has exceeded
MAX_DELTA. If it has then set it to and ordinary node.
**********************************************

cluster_load_balance()
if (Clusterhead == My_Node_Id)
   if (ABS(My_Degree-Elected_Degree)>=MAX_DELTA)
      Clusterhead = FALSE;
```

7. Simulation Experiments and Results

We conducted simulation experiments to evaluate the performance of the proposed heuristic. The load-balancing heuristic was implemented into 4 cluster election heuristics: the Max-Min heuristic [12], the Linked Cluster Algorithm (LCA) [1], the revised LCA (LCA2) [4], and the Highest-Connectivity (Degree) [8, 6] heuristic. These simulation results were then compared against similar results produced by the cluster election heuristics running without load-balancing. We assumed a variety of systems running with 100, 200, 400, and 600 nodes to simulate ad hoc networks with varying levels of node density. Two nodes are said to have a wireless link between them if they are within communication range of each other. The performance was simulated with the communication range of the nodes set to 20, 25 and 30 length units. Additionally, the span of a cluster, $i.e.$, the maximum number of wireless hops between a node and its clusterhead (d) was set to 2 and then 3 for each of the simulation combinations above. The entire simulation was conducted in a 200×200 unit region. Initially, each node was assigned a unique node id and (x, y) coordinates within the region. The nodes were then allowed to move at random in any direction at a speed of not greater than 1/2 the wireless range of a node per second. The simulation ran for 2000 seconds, and the network was *sampled* every 2 seconds. At each sample time the proposed load-balancing and cluster election heuristic was run to determine clusterheads and their associated clusters. For every simulation run a number of performance metrics were measured for the 2000 seconds of simulation. The main simulation metric measured was *Clusterhead Duration*, and provided a basis for evaluating the performance of the proposed load-balancing heuristic.

Definition 2 (Clusterhead Duration) - *The mean time for which once a node is elected as a clusterhead, it stays as a clusterhead. This statistic is a measure of stability, the longer the duration the more stable the system.*

As described in Section 5.2, the load-balancing heuristic has several customizable approaches. For example, once a clusterhead loses its leadership role before exhausting its clusterhead budget, it may set the node's new virtual id to 0, or its Old Virtual id + Number of runs of the heuristic since it was elected a clusterhead. Additionally, the clusterhead budget may be a function of (i) contiguous times elected as a clusterhead, (ii) maximum amount of work performed or load, (iii) minimum amount of work performed or idle, (iv) or any combination of these. Work is calculated to be the summation of the number of nodes in the cluster times the sample period. That is,

$$Work = \sum_{i=1}^{n} sample\ period\ duration_i * cluster\ size_i$$

For the purposes of these simulations we have set the clusterhead budget to be a function of the maximum amount of work it performs (5000 units of Work). That is, once a clusterhead becomes a clusterhead it will remain a clusterhead until it has exhausted it maximum work load, or until it loses out to another clusterhead based on the rules of the cluster election heuristic. Once a clusterhead does lose its leadership role to another clusterhead its new virtual id is set to its old virtual id + the number of runs of the heuristic since becoming a clusterhead.

Figure 1 shows the clusterhead durations for the Max-Min heuristic with and without load-balancing applied. The load-balancing makes a noticeable difference in the clusterhead duration (ranging from 14% to 28%). Furthermore, the variance of the clusterhead duration without load-balancing applied is more than 500% greater than with load-balancing. This shows that while the load-balancing heuristics generates longer clusterhead durations, it also produces much tighter and more deterministic responses (stability). These results are not surprising, as mentioned earlier the Max-Min heuristic favors the election of larger ids. Therefore, once a clusterhead is elected it will stay a clusterhead for a maximum of the programmed budget. This will provide the longer clusterhead durations that we see. The load-balancing heuristic is continuously rotating ordinary nodes into the position of becoming a clusterhead. Therefore, once a clusterhead budget is exceeded, a different clusterhead is elected and the process repeats. This provides the load-balancing effect of distributing the responsibility of being a clusterhead among all nodes.

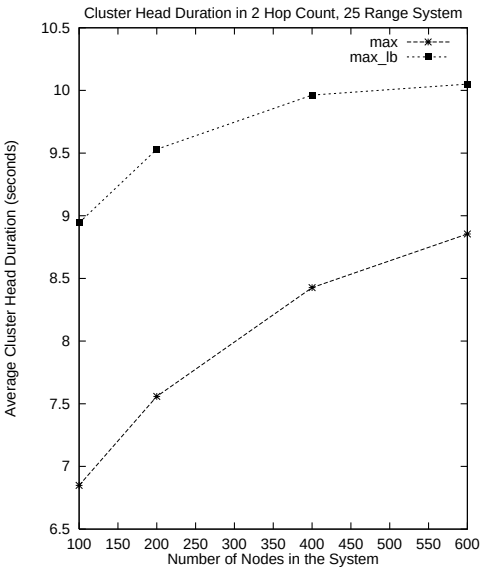
Figure 2 shows the clusterhead durations for the LCA heuristic with and without load-balancing applied. The clusterhead duration for LCA is only slightly greater without load-balancing than with it. At first glance this may appear that the load-balancing is a hindrance rather than a help. However, we see that the variance of clusterhead duration for LCA without load-balancing is 400% larger than with load-balancing. Therefore, while LCA with load-balancing produces slightly shorter clusterhead durations, load-balancing once again, provides a much tighter and more deterministic response (stability).

Figure 3 shows the clusterhead durations for the revised LCA heuristic (LCA2) with and without load-balancing applied. The clusterhead duration with load-balancing show a 25% improvement over without load-balancing. Furthermore, the variance with load-balancing applied is greatly improved over without load-balancing, showing an improvement of better than 500%. Therefore, load-balancing once again, provides a larger clusterhead duration and a much tighter and more deterministic response (stability). Again, just as with the Max-Min heuristic, the LCA2 heuristic is biased towards certain node ids (lower node ids). Therefore, it is not surprising that the LCA2 heuristic produces results quite similar to that of the Max-Min heuristic.

Figure 4 shows the clusterhead durations for the Degree based heuristic with and without load-balancing applied. For this simulation we have set the MAX_DELTA value to 10. The load-balancing has produced a noticeable improvement in the clusterhead duration (better than 200%). However, the variance of the clusterhead duration without load-balancing is less than that with load-balancing. By examining the clusterhead duration without load-balancing we see that it is between 2 and 3. This happens to be approximately equal to the sample rate of the simulation, 2 seconds. Therefore, what we are observing is that clusterheads exist for only a single snapshot of the simulation and then give way to another node. The variance will be small because each clusterhead is only serving as a clusterhead for one snapshot, on the average. While we want each node to serve as a clusterhead, we would like for each node to stay a clusterhead for some period of time. In this case longer than one snapshot. With this knowledge, the variance of the Degree based heuristic with load-balancing is actually quite good and provides a much needed larger clusterhead duration.

8. Conclusion

Two clusterhead load-balancing heuristics have been proposed for ad hoc networks. The first heuristic is for cluster election heuristics that favor the election of clusterheads based on node id. Here the heuristic places a budget on the contiguous amount of time that a node stays a clusterhead. As seen from the simulation results, this heuristic produces larger clusterhead durations while decreasing the variance, increased stability. The second heuristic is for cluster election heuristics that favor the election of clusterheads based on the degree of connectivity. A clusterhead stays a clusterhead as long as its degree of connectivity is within a specific range. The Degree based heuristic was simulated with this load-balancing heuristic. The simulation results show a much needed increase in clusterhead duration while still maintaining a low variance.



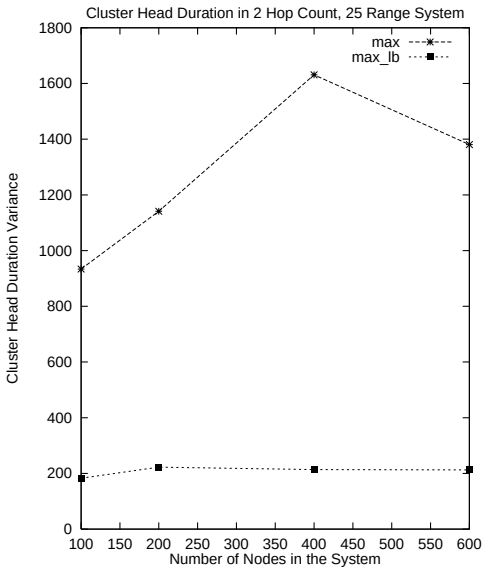
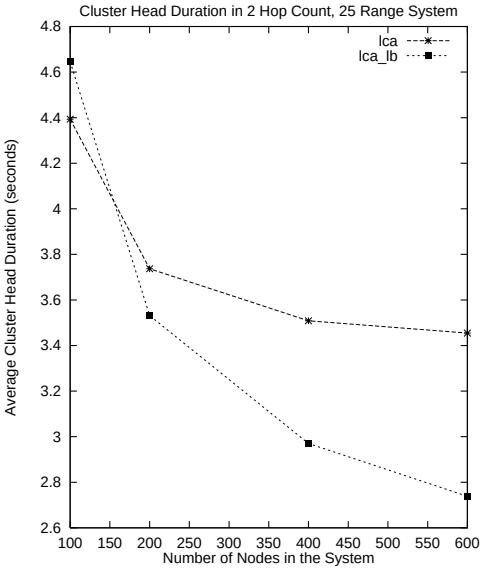


Figure 1. Clusterhead duration and variance, Max-Min.



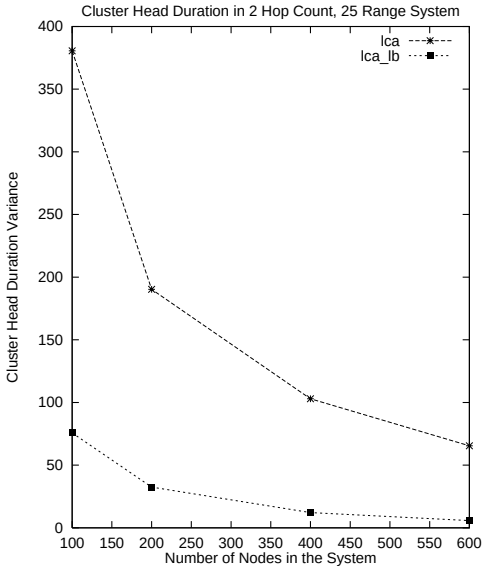
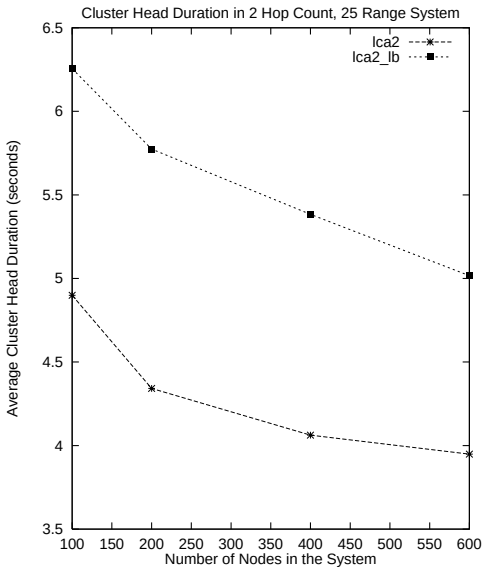


Figure 2. Clusterhead duration and variance, LCA.



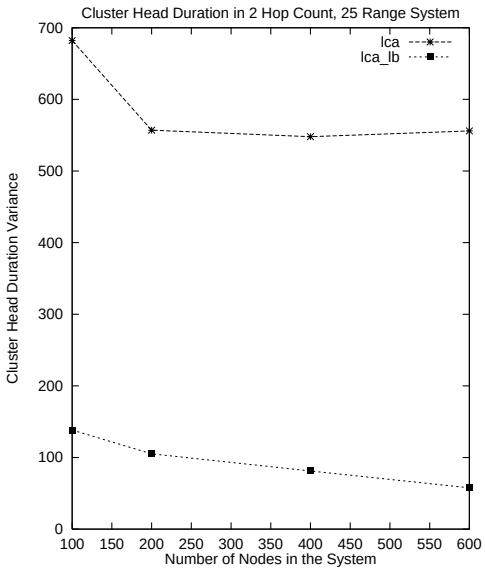
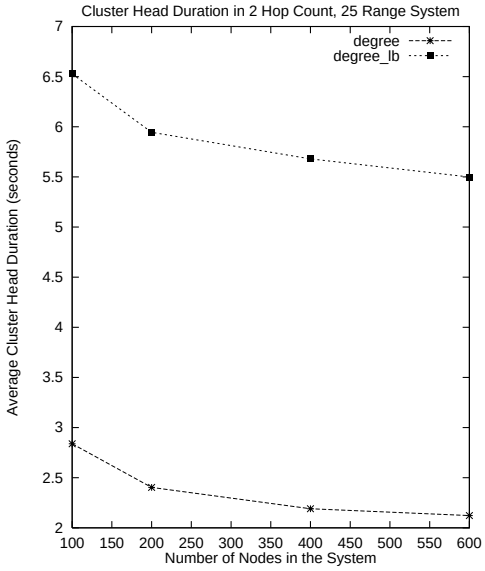


Figure 3. Clusterhead duration and variance, LCA2.



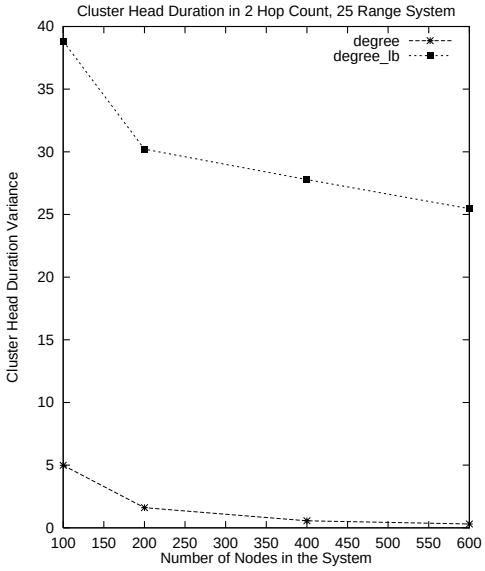


Figure 4. Clusterhead duration and variance, Degree.

References

[1] D. J. Baker and A. Ephremides. The Architectural Organization of a Mobile Radio Network via a Distributed Algorithm. *IEEE Transactions on Communications*, COM-29(11):1694–1701, November 1981.

[2] D.J. Baker, A. Ephremides, and J. A. Flynn. The Design and Simulation of a Mobile Radio Network with Distributed Control. *IEEE Journal on Selected Areas in Communications*, pages 226–237, 1984.

[3] B. Das and V. Bharghavan. Routing in Ad-Hoc Networks Using Minimum Connected Dominating Sets. In *Proceedings of ICC*, 1997.

[4] A. Ephremides, J. E. Wieselthier, and D. J. Baker. A Design Concept for Reliable Mobile Radio Networks with Frequency Hopping Signaling. *Proceedings of IEEE*, 75(1):56–73, 1987.

[5] E. Gafni and D. Bertsekas. Distributed Algorithms for Generating Loop-free Routes in Networks with Frequently Changing Topology. *IEEE Transactions on Communications*, pages 11–18, January 1981.

[6] M. Gerla and J. T.-C. Tsai. Multicluster, mobile, multimedia radio network. *ACM Baltzer Journal of Wireless Networks*, 1(3):255–265, 1995.

[7] L. Kleinrock and J. Silvester. Spatial Reuse in Multihop Packet Radio Networks. *Proceedings of the IEEE*, 75(1):156–167, January 1987.

[8] Abhay K. Parekh. Selecting Routers in Ad-Hoc Wireless Networks. In *ITS*, 1994.

[9] V. D. Park and M. S. Corson. A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks. In *Proceedings of IEEE INFOCOM*, April 1997.

[10] C.E. Perkins and P. Bhagwat. Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers. In *Proceedings of ACM SIGCOMM Conference on Communication Architectures, Protocols and Applications*, pages 234–244, August 1994.

[11] Jennifer Lundelius and Nancy Lynch. An Upper and Lower Bound for Clock Synchronization. *Information and Control*, Vol. 62 1984.

[12] A. Amis, R. Prakash, T. Vuong, and D.T. Huynh. Max-Min D-Cluster Formation in Wireless Ad Hoc Networks. In *Proceedings of IEEE INFOCOM*, March 1999.

[13] A. Tanenbaum. *Computer Networks(3^{rd} Edition)*. Prentice Hall, Upper Saddle River, N.J., 1996.