Balancing Loads in Mobile Ad hoc Networks *

Damla Turgut
School of Electrical Engineering and Computer Science
The University of Central Florida
Orlando, FL 32816-2450
E-mail: turgut@cs.ucf.edu

Begumhan Turgut Sajal K. Das Ramez Elmasri
Department of Computer Science and Engineering
The University of Texas at Arlington
Arlington, TX 76019-0015
E-mail: {bturgut, das, elmasri}@cse.uta.edu

ABSTRACT

Mobile ad hoc network consists of freely moving nodes communicating with each other through wireless links. In this paper, we propose a load balancing algorithm for these networks with nodes having different processing powers and thus can perform extensive computations apart from forwarding packets for other nodes. These nodes will also have various degrees of battery powers as well. Due to the heterogeneity of the systems in terms of processing and battery powers, naturally, there will be load imbalance. If the workload is distributed among the nodes in the system based on the resources of individual nodes, the average execution time can be minimized and the lifetime of the nodes can be maximized. Our proposed load balancing algorithm takes into consideration several realistic parameters such as processing and batter powers of each node, and communication cost for the loads being transfered between the overloaded and underloaded nodes. Simulation experiments demonstrate that our proposed algorithm achieves performance improvements in terms of processor utilization, execution time, and balance factor.

Keywords: ad hoc networks, load balancing, energy conservation

1 Introduction

Mobile multi-hop radio networks, also called *ad hoc* or *peer-to-peer* networks have gained prominence in the recent years. These networks plays a critical role in places where a wired backbone is neither available nor economical to build, such as law enforcement operations, battle field communications, disaster recovery situations, and so on. These scenarios demand a network where all the nodes including the base stations are potentially mobile, and communication must be supported untethered between any two nodes. Even though ad hoc networks were initially designed and developed to respond to emergency type situations, it is possible to see more civilian applications which are currently being developed such as conferencing, home networking, automotivePC interaction, personal area networks and bluetooth, embedded computing applications, and so on [4, 6].

Load balancing has been one of the major research interests in both wired and wireless networks. There are various characteristics of nodes that needs to be taken into consideration on the underlying network [1, 5]. As these parameters change in terms of increasing number of nodes, being wireless, node mobility, heterogeneity of nodes, and so on, the process of finding a balanced network becomes more complex. Load balancing is certainly an essential part of an optimal network. For example, if nodes with less processing capabilities have been given large amount of loads and do not have any means to share the load, the process of completing the job will suffer. This could reflect on the network performance just as much as other drawbacks such as congestion would do on the traffic.

It has been generally assumed that the nodes are identical in all respect, that is, they have the same processing speed, memory capacity and transmission power. This assumption on uniformity of devices does not hold, due to the nodes that will constitute

*0-7803-7661-7/03/$17.00©2003 IEEE.

the network are being developed with varying capabilities. For example, the devices which form the network could be wireless phones, palm-tops, laptops, and so on. It is expected that loads in ad hoc network will not only be used for communication but also for extensive computation since these nodes will have high processing powers. Thus, the nodes are responsible for forwarding packets from/to the other nodes and their processing power can be used for computation intensive purposes [3].

Due to the heterogeneity of the systems in terms of computing/processing power, it is likely that there will be load imbalance. This means that some nodes might be idle while some might be overloaded. A powerful node, that is in terms of processing power, is expected to have less or no load at all most of the time simply because it will finish its own work load quickly. Therefore, it is not desirable to have any overloaded nodes while there are underloaded nodes. The processing capability of such powerful nodes can be harnessed by other overloaded nodes if a fraction of their load is distributed to other nodes. If the work load is shared efficiently, not only the average execution time can be minimized but also the battery power of the nodes can be better utilized. As a result, the lifetime of the nodes can be maximized. In addition, long-lived nodes also contribute to the stability of the network topology which is crucial in these types of networks [7]. Achieving stable network topology in ad hoc networks in turn will provide solutions to several problems related to areas such as clustering and routing [1, 2, 3, 8].

Finding a node to which a load can be transfered is non-trivial. The problem would be less complicated if we consider a centralized system where the current load of every node is known to all the nodes. Also, the hop distance between every node pair would be known to the central station/server. A node that needs to distribute its load could easily find the target node to share its load. However, it may not be a good idea for a node to share its load with another node that is several hops away due to the time it may take to obtain the information about the load of that node. Moreover, the information about the load might change by the time this information is assimilated.

It can be safely assumed that a node knows the status of its immediate neighbors, that is, the nodes which are within its transmission range. This is made possible by the continuous transmission of beacon signals and all nodes in the vicinity can receive/hear them. For a node to find the status of other nodes, it sends out query messages, which are heard by its immediate neighbors. These neighbors can in turn relay the query messages and gather the information about nodes that are far away (in terms of number of hops) from the node that started the query process. It can be noted that these kind of query broadcast is expensive and thus should be minimized as much as possible. The problem becomes even more difficult if the nodes are constantly moving and the load of the nodes are changing frequently. A node wishing to distribute its load would require to identify the nodes with which it can share its load. The lightly loaded nodes are identified on a hop-by-hop basis, that is, query messages would first reach to the immediate neighbors (or the 1-hop neighbors), then the 2-hop neighbors, and so on. It is not desirable to share a small chunk of load with a node that is several hops away. This is due to the high communication cost association. Therefore, the goal is to find the nodes, at the right distance, which are capable of sharing the extra load so that there is no or minimum load imbalance in the system.

In this paper, we present a load balancing algorithm that takes into consideration the communication cost for finding the best suited node for load sharing. The load balancing algorithm is invoked only when there is an imbalance with respect to a certain threshold and the performance is evaluated in terms of execution time, battery power and balance factor. The rest of the paper is organized as follows. Section 2 describes our algorithm in detail. A simulation study and experimental results are presented in section 3, followed by the conclusions in section 4.

2 Proposed Load Balancing Algorithm

In this section, we present a load balancing algorithm [7] that is aimed to balance the loads on a given network by calculating the loads and exchanging messages among the nodes to find a way of either receiving or sending loads. We give the preliminaries, balance factor, system activation and algorithm steps.

2.1 Preliminaries

We model the nodes in the network by an undirected graph $G = (V, E)$, where V represents the set of nodes v and E represents the set of links e. Note that the cardinality of V remains the same but the cardinality of E changes with the creation and deletion of links. Associated with every node v, is a value p_v, which represents its processor power and b_v which represents the battery power. Due to heterogenous network, a node (i.e., a laptop) can have very high processing power than another (i.e., a palm-top). It is also assumed that the p_v values do not change whereas the b_v values decrease as the execution of the jobs continues. We defined [2, 7] the neighborhood set of a node v as

$$N[v] = \bigcup_{v' \in V,\ v' \neq v} \left\{ v' | dist(v,\ v') < tx_{range} \right\}$$

where tx_{range} is the transmission range of node v. The neighborhood of a clusterhead is the set of nodes which lie within its transmission range. The notations used are shown in table 1.

2.2 Balance Factor

Load balancing has been dealt with in great depth in distributed systems, where nodes with different computing power are connected among themselves through a known topology that does not change. A node apart from executing its own work load, also has to route (forward) messages for other nodes. Thus, the load handled by a node depends on its own load assignment and the average system load. It is not desirable to share the load with a node whose load is already in close proximity with the average system load.

To quantitatively measure how well balanced the nodes are, we introduce a parameter called *balance factor* (BF). We define the BF as the inverse of the variance of the load of the nodes. Thus,

$$BF = \frac{N}{\sum_v (l_v - \mu)^2}$$

where N is the total number of nodes in the system, l_v is the load of node v, and μ is the average system load, which is computed as

$$\mu = \frac{1}{N} \sum_{v \in V} l_v$$

Clearly, a higher value of BF signifies a better load distribution and it tends to infinity for a perfectly balanced system.

It is difficult to maintain a perfectly load balanced system at *all* times because of the high cost associated with triggering of the load balancing algorithm frequently. The frequency at which the load balancing algorithm is invoked is an important issue. If it is invoked periodically at a high frequency, then the *latest* topology of the system can be used to find the node that would be suitable to share the load. However, this will lead to high computational cost resulting in the loss of battery power or energy. If the frequency of update is low, there are chances that current topological information will be lost, resulting in selection of unsuitable nodes to distribute the load. It is not necessary that the load balancing algorithm will be triggered whenever there is a new load into the system. It will only be invoked when the load falls below a certain threshold. Another measure that we have taken towards reducing the computation cost is to allow only overloaded nodes to distribute loads to other nodes. The underloaded nodes will not request load to execute from others; rather they wait until the request(s) to execute jobs arrive from the overloaded node(s).

Table 1: Load Balancing Algorithm Notations

Notation	:	Meaning
N	:	Total number of nodes
tx_{range}	:	Transmission range
m_v	:	Total number of jobs of node v
l_v	:	Load of node v
p_v	:	Processing power of node v
b_v	:	Battery power of node v
ς^i	:	ith job of node v
μ	:	Average System Load
t_v	:	Threshold value of node v

2.3 Algorithmic Execution

We generate the number of nodes with random positions. These nodes are given maximum displacement (*max_disp*) and they can move in all directions. The range of (min, max) processing and battery powers (min, max) are assigned to these nodes. It is assumed that nodes with high processing power also have high battery power – that is, a laptop would have a higher battery power than a PDA. A number of jobs are created in each node. The job size is measured in terms of million instructions. Each node then calculates its current load by summing all the jobs assigned to it. The construction of multi-hop neighbors starts in every node until all the nodes are processed. The average system load is calculated and every node is made aware of it. The overhead parameter is defined and used for returning one job per hop in every node. Since our load balancing algorithm is distributed, each node collects the information about the current load of the other nodes by a repetitive query messaging. After gathering the loads of all the nodes ($l_v \forall v$) each node computes the average system load μ as defined earlier. If the load of a node is above a defined threshold value, it tries to distribute the extra load to its 1-hop neighbors. If its 1-hop neighbors are also overloaded, the node requests its 2-hop neighbors to receive the load and so on. In our algorithm, an overloaded node means that its load value is higher than a range, set by a threshold value. The range is set to be between $(1-t_v) \times \mu$ and $(1+t_v) \times \mu$, where t_v is the threshold value for node

v. The value of t_v is a tunable system parameter. If a node is below a threshold value, it waits for overloaded nodes to distribute the jobs. This algorithm always tries to move the load from overloaded nodes to underloaded neighbors (from 1-hop to multi-hop). If the load is distributed to a multi-hop neighbor, the overhead, which includes job instructions, is added to every return hop to the original node.

2.4 Load Balancing Steps

The procedure consists of 5 steps as described below.

Step 1: Find the immediate (1-hop) neighbors of each node v (i.e., nodes within its transmission range) and let this denoted by $N_1[v]$. Thus,

$$N_1[v] = \sum_{v' \in V,\ v' \neq v} \left\{ dist(v,\ v') < tx_{range} \right\}$$

Similarly, find the 2-hop neighbors, $N_2[v]$. This process is continued for all the nodes in the network. Note that $N_1[v]$ is the same as $N[v]$, which was defined in section 2.1.

Step 2: For every node, calculate the total load by summing the each job sizes, c_v^i as

$$l_v = \sum_{i=1}^{m_v} c_v^i$$

where

$$c_v = p_v \times b_v$$

and p_v and b_v are the processing and battery powers of node v.

Step 3: Calculate the average load of the system as

$$\mu = \frac{1}{N} \sum_{v \in V} l_v$$

Step 4: A node is *underloaded* if

$$l_v < (1 - t_v) \times \mu$$

In this case, the node waits to process the job(s) distributed from overloaded nodes.

A node is *overloaded* if

$$l_v > (1 + t_v) \times \mu$$

In this case, the node starts screening its immediate (1-hop) neighbors to distribute the load. If an underloaded node is not found in its 1-hop neighbors, the node searches its 2-hops neighbors, and so on.

Step 5: Repeat Step 4 until there are no overloaded or underloaded nodes. If the nodes lie within the range $[(1 - t_v) \times \mu, \leq (1 + t_v) \times \mu)]$, the loads are considered already balanced.

The execution of this algorithm finds the target nodes capable of sharing the extra load based on the processing and battery powers of those nodes. It tries to distribute the load to the node's immediate neighbors. If none of the 1-hop neighbors are available to share the load, then the 2-hop neighbors are queried, and so on. Since the load of the nodes changes dynamically, the average system load changes accordingly. The algorithm always uses the up-to-date average system load by querying the loads of all the nodes in the network.

3 Simulation Study

The simulation model consists of N nodes that are randomly scattered on a square (100×100) grid. The nodes can move in all possible directions with displacement varying uniformly between 0 to a maximum displacement value (*max_disp*), per unit time. Also, the node of nodes N or the node density (nodes per unit area) is varied. Each node generates loads with a certain rate, the average of which is λ loads per node. A fraction of the newly generated load by a node will get distributed among other nodes depending on the current load of the node and the average load of all the nodes in the system. The nodes are assigned varying processing and battery powers. Since a node with a high processing power naturally posses a high battery power, the relationship between these two has been maintained during the random generation of the values. To measure the performance of our system, we identify three metrics: (1) execution time improvement, (2) standard deviation, and (3) balance factor. These three parameters are studied for varying number of nodes (N) in the system with respect to the average job size.

In our simulation experiments, N was varied between 100 and 300 with an increment of 50. The average job size was within the range of 100 to 1000 millions of instructions. The nodes moved randomly in all possible directions. Each node knows, not only its own current load, but also the loads of its multi-hop neighbors as well as the average system load, since our algorithm is distributed in nature. Before distributing jobs to other nodes, a node checks if its current load is outside the specified threshold. The threshold value used in these experiments is set to 20% which can be set to any other value appropriately. The overhead propagated for the jobs coming back from neighbors (1- or multiple-hops away) is 0.01% per hop traveled. The overhead simply adds the extra number of instructions per hop. Note that these values are arbitrary at this time and can be adjusted according to a specific system requirements.

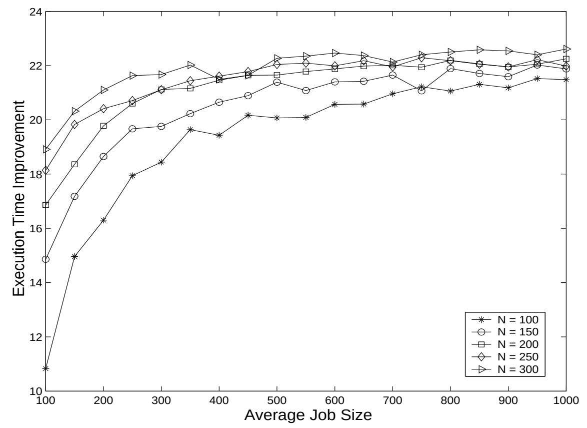

Figure 1: Execution Time Improvement After Load Balancing

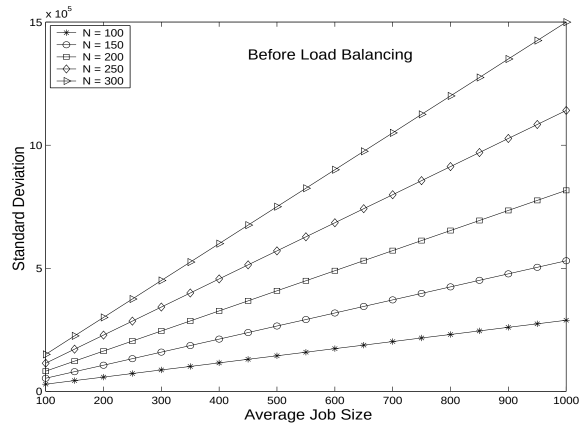

Figure 2: Standard Deviation Before Load Balancing

3.1 Experimental Results

Figure 1 shows the execution time improvement with respect to the average job size. It is noted that execution time gives better results for both large job sizes and a large number of nodes. It is expected to improve in the execution time when more nodes are present in the system, since more nodes ultimately means more allocation of jobs to other nodes. Figures 2 and 3 respectively present standard deviation with respect to the average job size before and after the load balancing algorithm is used. The standard deviation represents how well the excess loads of different nodes have been distributed. A small number means that the nodes have used similar percentage of their processing and battery powers. In this sense, the nodes exist in the system in longer periods of time. As it can be seen from figure 3, the standard deviation has shown significant improvement (as much as three fold) after the load balancing. Figures 4 and 5 show balance factor with respect to the average job size. Higher numbers in the balance factor signifies the improvement as it can be seen from figure 5. The balance factor gives better results for resource-poor networks.

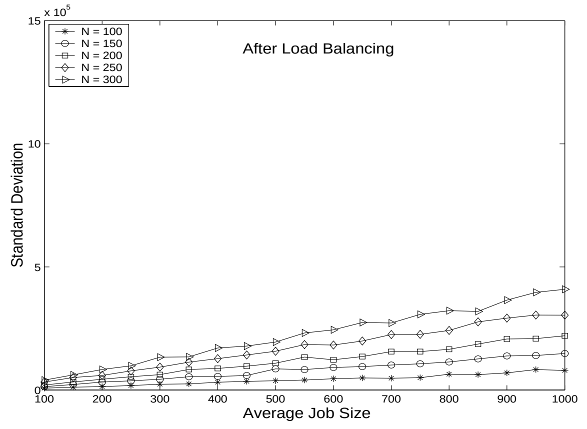

Figure 3: Standard Deviation After Load Balancing

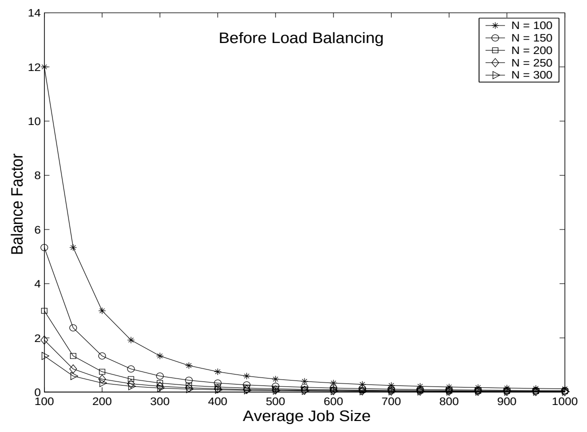

Figure 4: Balance Factor Before Load Balancing

4 Conclusions

In this paper, we described our load balancing algorithm for mobile ad hoc networks where each of the nodes has varying degree of processing and battery powers. It is desirable for nodes with higher processing power to handle loads that require more

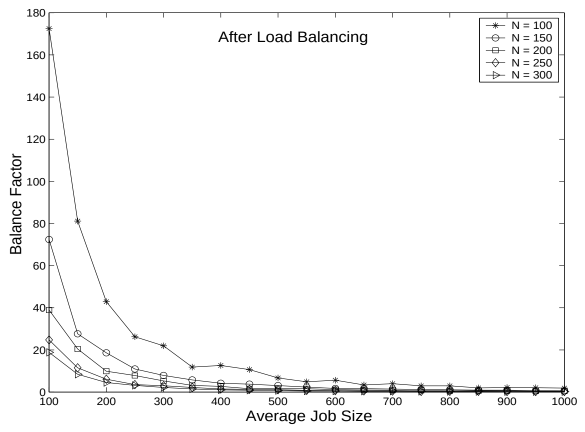

Figure 5: Balance Factor After Load Balancing

computational power such that the execution time of the loads is minimized. Basically, the goal is to obtain a balanced ad hoc network to achieve better performance in terms of execution time of jobs and throughput. Simulation results have shown significant improvements in the execution time, standard deviation (that is, utilization), and balance factor. Our load balancing algorithm tries to distribute the load in such a way that the system remains in a equilibrium resulting in uniform consumption of battery.

References

[1] A. Amis and R. Prakash, "Load-Balancing Clusters in Wireless Ad Hoc Networks," *Proceedings of ASSET 2000*, Richardson, Texas, March 2000, pp. 25-32.

[2] M. Chatterjee, S.K. Das and D. Turgut, "WCA: A Weighted Clustering Algorithm for Mobile Ad hoc Networks", *Journal of Clustering Computing, (Special Issue on Mobile Ad hoc Networks)*, Vol. 5, No. 2, April 2002, pp. 193-204.

[3] H. Hassanein and A. Zhou, "Routing with Load Balancing in Wireless Ad hoc Networks," *Proceedings of the 4th ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, Rome, Italy, July 15-20, 2001, pp. 89-96.

[4] C.E. Perkins, Ed, *Ad Hoc Networking*, Addison Wesley, 2001.

[5] M. Singhal and N.G. Shivaratri, *Advanced Concepts in Operating Systems*, McGraw Hill, 1994.

[6] C-K Toh, *Ad Hoc Mobile Wireless Networks Protocols and Systems*, Prentice Hall, 2002.

[7] D. Turgut, *Efficient Algorithms and Protocols for Stability Management in Mobile Ad Hoc Networks*, PhD Dissertation, University of Texas at Arlington, May 2002.

[8] D. Turgut, S.K. Das and M. Chatterjee, *Longevity of Routes in Mobile Ad hoc Networks*, VTC Spring 2001.