Load Balanced Link Reversal Routing in Mobile Wireless Ad Hoc Networks

Nabhendra Bisnik, Alhussein Abouzeid, Costas Busch
Rensselaer Polytechnic Institute
Troy, New York
bisnin@rpi.edu, abouzeid@ecse.rpi.edu, buschc@cs.rpi.edu

Abstract

*Link reversal routing (*LRR*) is a local, distributed, and low-overhead technique used to maintain loop free routes in mobile wireless ad hoc networks. We explore the problem of load balancing the packet traffic in the network when using the LRR created routes. We study a fundamental LRR algorithm and identify usual situations which cause the load to be unbalanced. We propose three modifications to the LRR algorithm such that the load is distributed in a more uniform manner. The modifications preserve all the desirable qualities of LRR algorithms such as loop free routes, local response to topology change, low overhead and are completely distributed in nature. We perform simulations in order to evaluate the performance of the proposed modifications for both single-path and multi-path scenarios. The simulation results show that the modifications enable LRR to provide a much improved load balancing and ensure higher network lifetime.*

1 Introduction

Designing efficient routing algorithms for mobile wireless ad hoc networks is a very challenging and interesting problem. The routing algorithms designed for the wired networks (OSPF, RIP) do not work efficiently in the mobile wireless ad hoc networks because they are inherently different from the wired networks. To begin with, the ad hoc networks lack the regular infrastructure features, such as switches and routers, that are fairly common in the wired networks. In addition the wireless nodes have limited communication range which allows them to directly communicate with only a few nodes. If a node wishes to send a packet to a node that does lie in its communication range then the packet has to be forwarded by the intermediate nodes until it reaches the destination. Thus in an ad hoc network each node shares the responsibility of appropriately routing the packets. Also the mobile wireless ad hoc networks are characterized by variable link qualities and frequently changing topology, which may lead to routing loops and large overheads if the routing algorithms of regular wired networks are used. Finally in most of the deployments of wireless ad hoc networks, particularly wireless sensor networks, the wireless nodes are battery powered. Therefore, the routing algorithm must ensure that the load of forwarding packets is as evenly distributed as possible otherwise the nodes that carry most of the burden of forwarding would run out of power quickly which may lead to disconnected network.

A lot of research effort has been dedicated to the development of efficient routing algorithms for mobile wireless ad hoc networks. It is desirable that these routing protocols must converge quickly, ensure high packet delivery ratio, be scalable and energy efficient. The routing algorithms may be classified into *reactive* and *proactive*. In reactive routing algorithms, a node maintains information about the route to another node only if it has an active communication session with that node (e.g. AODV [14], DSR [9]). On the other hand in proactive routing algorithms a node maintains information about route to every other node in the network, irrespective of the current sessions (e.g. DSDV [15]). The proactive routing algorithms typically concede higher overhead in order to maintain consistent states but lead to lower delays in comparison to the reactive routing algorithms.

Link reversal routing (*LRR*) [5] is a technique for routing algorithms in mobile ad hoc networks that can be used in both proactive and reactive modes. LRR generates and maintains a destination oriented directed acyclic graph (DAG) on top of the underlying undirected connectivity graph. In order to achieve loop free paths, LRR uses a "height" mechanism such that each node has a height, and links are directed from higher heights to lower heights. The heights of nodes along a path to the destination are in strictly decreasing order. In proactive mode, LRR maintains DAGs for all possible destinations at all times, while in reactive mode, LRR generates and maintains a DAG for a destination only when required. The routing control information that is required for maintaining DAGs is local, and is restricted to one hop information, thus, it incurs low overhead and scalable. The loop-free nature of LRR ensures a high packet delivery ratio. Also LRR algorithm maintains redundant routes, thus making the network robust against link errors and topology changes. LRR algorithms are particularly efficient in networks with moderate rate of topology changes. TORA [12] is a well studied link reversal based

routing protocol for mobile ad hoc networks.

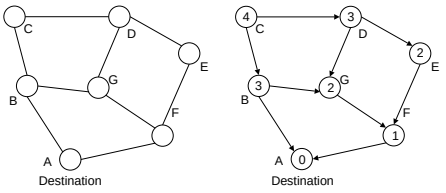
LRR naturally fits the communication demands of mobile wireless sensor networks that are involved in the task of surveillance or environmental monitoring. In such networks, all the nodes deployed in the surveillance field forward the sensed data to a sink node. For LRR this translates to maintaining a single DAG for the sink. For similar reasons LRR is suitable for mobile wireless ad hoc networks where nodes connect to the Internet through a single access point. In such networks the wireless nodes are typically battery powered and it is desired that the load is evenly spread across the nodes in order to ensure longevity of the network lifetime.

In this study we inspect the problem of load balancing in a basic LRR algorithm, the *full LRR algorithm* [5]. Our load balancing analysis can be extended in a straightforward manner to other LRR algorithms as well. The motivation for load balancing is to ensure high network lifetime and high traffic throughput (through the avoidance of hot spots). We investigate the various causes of uneven load distribution in LRR. We show that LRR may lead to traffic hot spots resulting from the occurrence of unbalanced DAGs. We formally formulate the load balancing problem and propose three distributed heuristic mechanisms that improve the load balancing in LRR, namely: (i) Selfish-node based mechanism (SNBM) (ii) Proactive decrease in height (PDH) and (iii) Reactive increase in height (RIH). We use the *network lifetime*, *balance factor* and *squared sum* metrics in order to quantify the performance of the proposed load balancing mechanisms. We perform extensive simulations in order to evaluate the improvements achieved by the three mechanisms compared to the basic LRR algorithm. In the simulations we consider both multi-path and single-path routing, so as to evaluate the performance gains for varying packet traffic forwarding strategies.

The rest of this paper is organized as follows: A brief overview of related work is presented in Section 2. In Section 3, we briefly describe the LRR technique and the basic full LRR algorithm. In Section 4 we discuss the causes of load unbalance in LRR and define the load balancing problem and metrics, which is followed by a description of the heuristic mechanisms for distributing load in Section 5. The simulation results are presented in Section 6. Section 7 concludes the paper with a discussion of the future work.

2 Related work

Gafni-Bertsekas (GB) [5] give the first LRR algorithms for creating routes in mobile ad hoc networks. The GB algorithms are the full LRR algorithm (described in Section 3 and used in our study) and the *partial* LRR algorithm, which are both very simple LRR algorithms. However, the GB algorithms fail to converge when the graph is partitioned into disconnected components. The *lightweight mobile routing* (LMR) [4] and *temporally-ordered routing algorithm* (TORA) [12], are LRR algorithms which alleviate the convergence problem of the GB algorithms. A detailed discussion on LRR algorithms and routing protocols based on it can be found in [16] and [18].



(a) Connected network graph. (b) Resulting DAG.

Figure 1. Route generation phase.

Many studies have focused on developing mechanisms and analytically studying the load balancing problem in mobile wireless ad hoc networks. Load-Balanced Ad Hoc Routing (LBAR) protocol is proposed in [7] which is very similar to AODV, except that while forwarding the route request (RREQ) messages each node appends its current activity status to the RREQ. The destination uses this extra information in order to relay the RREQ message along the path with least load so that the load is evenly spread across the nodes. In [8], a load balancing algorithm is proposed for wireless access networks which maintains a load-balanced backbone tree rooted at the access point. The access point is responsible for updating the backbone tree and information about load distribution required at the access point. It is assumed in [8] that nodes have multiple-antennas, so wireless connections between neighbors is modeled as isolated point-to-point links, which makes the topology very similar to the wired-networks. In [17], the authors propose a mechanism of load balancing in ad hoc wireless networks that relies on dissipation of load-distribution information throughout the network. If a node is overloaded (serving more load than the average load in the network), then it queries its neighborhood for under-loaded nodes, and transfers its load to the under-loaded node. However the dissipation of load information increases the overhead and may cause instability.

In [13], the authors study that how alternate path routing (APR), which is a popular load balancing mechanism in wired networks, performs in a wireless ad hoc network. It is observed that the interference, caused by channel sharing, significantly reduces the performance of APR in wireless ad hoc networks. An analytical model of load distribution in multi-path routing is presented in [6]. It is shown that the multi-path routing does not provide any benefits in ad hoc networks, unless the number of paths is large. In [10] the authors show that multi-path routing may achieve significant improvement over single-path routing provided the various paths involved in multi-path routing are sufficiently disjoint. They use the number of interfering links existing between the nodes of two paths as the measure of correlation between the paths and choose the paths with minimum "interference correlation". It is shown in [20] that optimum load balancing is a NP-hard problem even for a simple network topology.

In [3], a LRR algorithm is presented that is a modification of the TORA algorithm, which shows how to improve the lifetime of the nodes by increasing the heights of low

power nodes. However, that algorithm has not been analyzed with simulations, and also incurs a high overhead due to the requirement of a power threshold mechanism. The load balancing mechanisms proposed in this paper are also built on top of a link reversal routing algorithm. The mechanisms use the notion of height in order to attract or repel traffic. By doing so, it is ensured that the mechanisms are completely distributed and flow of control information is localized. We give simulation results to show the effectiveness of our mechanisms.

3 Link reversal routing algorithm

In this section we first describe the general idea behind LRR algorithms and then we describe the GB full LRR algorithm. We assume that the network is fully connected and that each node i has a unique node ID v_i (the IDs are taken from an ordered set). The basis of an LRR algorithm is to assign heights to nodes, where the heights are chosen from some ordered domain. The *height* of a node i with respect to destination k is denoted by h_i^k. The heights of nodes may frequently change during the lifetime of the network. The LRR algorithm ensures loop freedom by maintaining an invariant that node i forwards a packet, destined for node k, to node j only if the following condition is satisfied:

$$h_i^k > h_j^k \vee (h_i^k = h_j^k \wedge v_i > v_j). \quad (1)$$

The LRR algorithm preserves the above invariant by maintaining a directed acyclic graph (DAG) for each destination. Let Γ^k denote the DAG for destination k. The link $i \rightarrow j$ would exist in Γ^k, only if i and j are within communication range of each other and Condition (1) is satisfied. A node is said to be a sink if it has not outgoing links in the DAG. It is desired that node k is the only sink in the Γ^k, in which case, we say that Γ^k is destination oriented. Also at all times it holds that $h_k^k < h_j^k$, $\forall\ j$. For the sake of simplicity, in the rest of this paper we consider only a single destination k. Thus we prefer to omit the superscript (k) in order to avoid notational clutter. The same analysis applies if we had more than one destinations.

Every link reversal algorithm consists of two phases: (i) route generation phase and (ii) route maintenance phase. In the route generation phase the connected graph of a wireless ad hoc network is transformed into a destination oriented directed acyclic graph (DAG) by assigning heights to the nodes, as illustrated in Figure 1. Any path in the resulting DAG proceeds in the direction of decreasing height, thus avoiding any loop formation. This is achieved by the broadcast of route request packets towards the destination and propagation of route reply packets along the reverse path [4]. The route generation phase involves forwarding of routing control packets over several hops and thus it may result in large overhead. However this phase is short lived and therefore it does not make significant contribution to the overall overhead. A node can communicate with the destination as long as it has at least one outgoing link in the DAG. The route maintenance phase is initiated whenever a node, other than the destination, becomes a sink (has no outgoing links). A node may become a sink if all its possible successors move out of its communication range. In such a situation the height of the sink node is smaller than the height of all its neighbors. A sink node is said to be in a *bad state* as it has no feasible route to the destination. The sink reacts to the situation by increasing its height which causes the reversal of some or all of its adjacent links. The link reversals of one sink may cause other nodes in the network to become sinks. The network eventually stabilizes to a destination oriented DAG. We note the the destination is the only sink that is not allowed to reverse.

There are several kinds of LRR algorithms, which are distinguished by the way the heights are updated when nodes become sinks. In the GB full reversal algorithm, the height is a single integer, and a sink updates its height so that all of its adjacent links become outgoing. In the GB partial reversal algorithm, the height is a 2-tuple of integers (heights are ordered lexicographically) and a sink reverses only links toward the part of the network that has not been affected by reversals (the second value in the height serves as a memory for previously reversed links). It is expected that the partial reversal algorithm performs less number of link reversals, and thus is faster to stabilize, than the full reversal algorithm. However, in [2], it is shown that in worst case the partial reversal algorithm may take much higher time to stabilize than the full reversal algorithm. So in this work we focus only on the full reversal algorithm. The other known LRR algorithms (see Section 2) are based on the same principles as the two aforementioned GB algorithms, so our work can be generalized to those algorithms as well. The only problem of the GB algorithms is that they don't stabilize on disconnected components of the network when partitions occur due to the mobility of the network nodes. However, in order to keep the presentation in the paper simple, we handle partitions in the simulations in a special way so they don't cause instability. The results in this paper can be easily extended in a straightforward manner to LRR algorithms that handle partitions. We now describe the full reversal algorithm.

Full reversal algorithm In the full reversal algorithm, the height of each node is an integer. The destination node has height 0. Let $N(i)$ denote the set of nodes that are neighbors of node i and

$$h_{\max_i} = \max_{j \in N(i)} h_j,$$

i.e. $h_{\max_i}$ is the maximum height in the neighborhood of node i. A node, i becomes a sink when its height is less than that of all of its neighbors. So in order to get out of the bad state the sink node adjusts its height, h_i, according to

$$h_i = h_{\max_i} + 1,$$

and reverses the direction of all its incoming links. This brings the node i out of the bad state. However, this may cause other nodes to become sinks. These nodes may again adjust their heights and reverse the direction of their incoming links. The full reversal algorithm is illustrated in Figure 2.

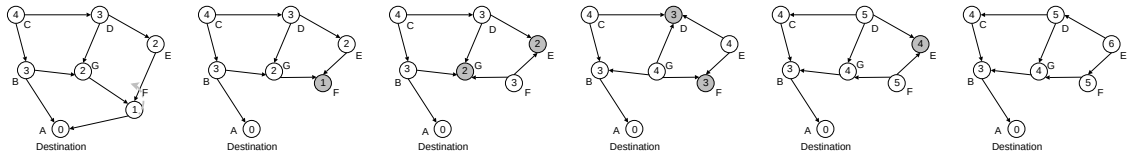


Figure 2. Execution of full reversal algorithm. The nodes shaded gray are sinks.

Henceforth, by "LRR algorithm" we will mean the full reversal algorithm. It should be noted that the exchange of routing information during the route maintenance phase is localized. Whenever a node becomes a sink it adjusts its height, which is a local decision, and informs its one hop neighbor about the change in the height which leads to the change in the direction of the links of the underlying DAG. Due to the localized nature of control information and presence of redundant paths, the overhead of the LRR algorithm is expected to be less than the DV routing algorithms. Also at every instant the DAG maintains the invariant condition (1) and hence the network is free of loops at all times. However a downside of the LRR algorithm is that large amount of state information (heights and link directions for DAGs corresponding to each destination) has to be maintained at each node. Also the heights of nodes is a non decreasing function of time and thus may become arbitrarily large.

The nodes do not have any information about the topology of the network other than the knowledge of their neighbors. Therefore, if a node has more than one outgoing link then it has no extra information available in order to choose the *optimal* next hop when forwarding packets. One possible forwarding way is to randomly choose an outgoing link and send all the packets over that link. Another possible approach is to evenly distribute all the packet load over all outgoing links. The load balancing behavior of the LRR algorithm is explored next.

4 The load balancing problem

Although the LRR algorithm is very effective in maintaining loop-free routes without incurring high overheads, it may lead to unbalanced load distribution in the network. Unbalanced load reduces the lifetime of highly loaded node in battery powered networks, apart from causing high delays and packet losses due to congestion. In this section we first illustrate the problem of unequal load distribution with the help of an example. We then discuss the reasons that cause such an unbalance and then formally define the load balance problem. The insights obtained from the discussion would be helpful in devising solutions to the problem.

4.1 An example of unbalanced load distribution

Consider a network whose associated DAG for destination A is shown in Figure 3(a). In this case all the traffic

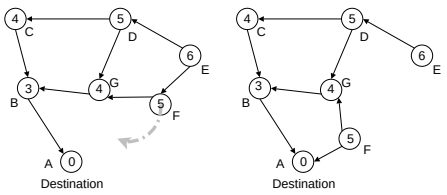


Figure 3. An example of unequal load distribution in the LRR algorithm.

destined to the destination is routed over the link $B \rightarrow A$, which causes B to be responsible for forwarding large packet traffic load. However, this situation cannot be avoided since there are no alternate paths. Now suppose that the node F moves towards node A as shown in Figure 3(a). As a result of this movement the node F comes within communication range of node A but moves out of the communication range of node E. This is reflected in formation of a new link $F \rightarrow A$ and disappearance of link $E \rightarrow F$ from the DAG, as shown in Figure 3(b). In the new topology, node F has a link to the destination and can route some portion of the traffic to the destination and thus relieve B of some traffic. But the DAG, constructed and maintained using the LRR algorithm, does not allow this because node F has no incoming links. So although an alternate path is available in the underlying undirected graph, most of the traffic will still be routed through node B. This is a clear situation of unbalanced load distribution caused in the LRR algorithm.

4.2 Causes of load unbalance

The network is said to be in an *unbalanced state* if the DAG does not allow routing of packets through alternate paths that are available in the underlying undirected connected network graph, thus resulting in some nodes being highly loaded. In such a scenario the overloaded nodes may be *relieved* if the direction of some links of the DAG is reversed. For example in figure 3(b), node B is overloaded although there exists an alternate path through F. This situation may be corrected if the direction of link $F \rightarrow G$ is

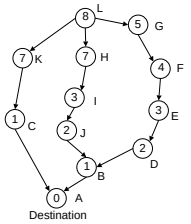


Figure 4. A selfish node (L) may lead to load unbalance.

reversed.

The main reason for the load unbalance is that the height of the nodes is a non-decreasing function of time. Each time a node becomes a sink, the full reversal algorithm leads to an increase in its height. So over a period of time this may create a situation where a node could possibly be capable of relieving the traffic (due to available adjacent links) but does do so because its height is greater than the height of all its neighbors. Such a node would only have outgoing links and is referred to as a *selfish node* since it does not forward packets of other nodes and simply pumps its own traffic into the network. A selfish node essentially partitions a DAG into *isolated routing components* which may obstruct load balancing: sets S_1 and S_2 are said to be isolated routing components if no node in S_1 has a node from the set S_2 on its path to the destination and vice-versa. In Figure 3(b), node F is selfish which leads to the overload of node B. The overloading caused due to the presence of selfish nodes is further illustrated in Figure 4. Node L is a selfish node that partitions the DAG into isolated routing components $\{B, D, E, F, G, H, I, J\}$ and $\{C, K\}$. This overloads node B although some traffic could have been routed through node C. Since the heights of the nodes never decrease with time, a node would remain selfish as long as some of its neighbors do not become sinks (and reverse directions of adjacent links toward the selfish node) or a node with a higher height does not enter the neighborhood of the selfish node.

Another implication of the non-decreasing nature of the heights is that the nodes that maintain *stable* routes over a long period of time (nodes whose adjacent links don't reverse for long periods of time), have comparatively lower heights than the rest of the nodes. If the height of a node is lower than most of the nodes in the network then it is very likely that most of the traffic would be directed to this node. Thus, in a way the LRR algorithm leads to overloading the stable nodes.

4.3 Load balancing definition and metrics

The load balancing quality in the LRR algorithm is characterized by (i) the destination oriented DAG, Γ, and (ii) the forwarding strategy used for the traffic, denoted by $\mathcal{S}_\Gamma$. The forwarding strategy governs how a node distributes the incoming load to the outgoing links of the DAG. One approach may be to equally distribute the load over all outgoing links while another approach may be to choose a single outgoing link and forward all incoming traffic over that link.

The packet traffic (bits/sec) forwarded on a link l of DAG Γ for the forwarding strategy $\mathcal{S}_\Gamma$ is denoted by $x_{\mathcal{S}_\Gamma}(l)$. The set of outgoing links at node i in a DAG Γ is denoted by $E_\Gamma(i)$. Thus, for a given DAG and forwarding strategy, the total traffic forwarded by node i, $\mathcal{F}_{\Gamma,\mathcal{S}_\Gamma}(i)$, is given by:

$$\mathcal{F}_{\Gamma,\mathcal{S}_\Gamma}(i) = \sum_{l \in E_\Gamma(i)} x_{\mathcal{S}_\Gamma}(l).$$

We define two metrics for characterizing the load balance, called *balance factor* and *squared sum*:

1. *Balance Factor (BF)*: Let $\overline{\mathcal{F}}_{\Gamma,\mathcal{S}_\Gamma}$ denote the mean traffic load handled by a node in the network. The BF for a given Γ and $\mathcal{S}_\Gamma$, denoted by $\mathcal{B}(\Gamma, \mathcal{S}_\Gamma)$, is defined as

$$\mathcal{B}(\Gamma, \mathcal{S}_\Gamma) = \left(\frac{1}{N} \sum_{i \in V} \left(\mathcal{F}_{\Gamma,\mathcal{S}_\Gamma}(i) - \overline{\mathcal{F}}_{\Gamma,\mathcal{S}_\Gamma} \right)^2 \right)^{-1} \quad (2)$$

   In other words BF is the inverse of load variance. The higher the BF is, more balanced is the traffic load. So from an optimization point of view, the load balance problem for the LRR algorithm is to find the optimal DAG for the underlying connected graph, $\Gamma^\star$, and the optimal forwarding strategy for the DAG, $\mathcal{S}^\star_\Gamma$, such that

$$\mathcal{B}(\Gamma^\star, \mathcal{S}^\star_\Gamma) = \max_{\Gamma, \mathcal{S}_\Gamma} \mathcal{B}(\Gamma, \mathcal{S}_\Gamma) \quad (3)$$

2. *Squared Sum (SS)*: The SS for a given Γ and $\mathcal{S}_\Gamma$, denoted by $\mathcal{C}(\Gamma, \mathcal{S}_\Gamma)$, is defined as

$$\mathcal{C}(\Gamma, \mathcal{S}_\Gamma) = \sum_{i \in V} \mathcal{F}^2_{\Gamma,\mathcal{S}_\Gamma}(i) \quad (4)$$

   While BF indicates that how much the load is balanced, the purpose of SS is to serve as a metric for the amount of load actually served by nodes in the network. The lower the SS is, the better is the load distribution.

It is important to understand that why a linear sum (LS), $\sum_{i \in V} \mathcal{F}_{\Gamma,\mathcal{S}_\Gamma}$, may not be a good indication of the load balancing quality. Suppose there are two forwarding nodes A and B. Consider scenario 1 where they forward 4 Mbps and 10 Mbps while in scenario 2 they forward 7 Mbps each. Clearly, scenario 2 is more desirable. However, this is not reflected in LS, which is same for both the scenarios. However SS for scenario 1 is 116, while for scenario 2, SS is 98, which indicates that scenario 2 is better than scenario 1. Moreover, when optimizing BF, If there is more than one solution to (3), we choose the solution with least SS.

The solution of load balancing thus consists of two parts: (i) constructing an optimal destination oriented DAG over connected network graph and (ii) devising an optimal forwarding strategy for routing traffic on the DAG. As shown in [20], the load balancing problem is NP-hard even for simple topologies. Although distributed solutions are very appealing in ad hoc networks, the distributed solution of optimal load balance problem is even harder and may require exchange of large amount of information among nodes. Therefore, in this work we will present some heuristic modifications to the route maintenance phase of LRR.

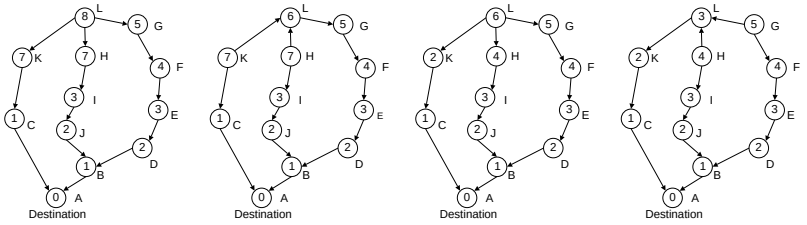


Figure 5. An example showing how the selfish node based mechanism (SNBM) works.

5 Heuristic mechanisms for load distribution

In this section we present three heuristic mechanisms for obtaining more uniform load balancing in the LRR algorithm for mobile wireless ad hoc networks. The basic idea behind all the mechanisms is that they try to make the distribution of heights in the network more uniform. The mechanisms differ in the way that they handle the events that trigger the update of the heights.

5.1 Selfish node based mechanism (SNBM)

As pointed out in Section 4.3 and illustrated in Figure 4, a selfish node may lead to load unbalance. Thus a simple way to preclude this possibility is to prevent the formation of selfish nodes whenever it is possible. In SNBM each node periodically checks the number of incoming links. If the number of incoming links is zero, then this implies that the node has become selfish. When a nodes discovers that it is selfish, it updates (decreases) its height in the following manner. Let $h_{\max}$ and $h_{\min}$ denote the maximum and minimum heights of the neighbors of the selfish node. If $h_{\max} - h_{\min} < 2$, then the selfish node does nothing, since in such a situation there is the danger that the selfish node may become a sink if it decreases its height and thus become useless. If $h_{\max} - h_{\min} \geq 2$, then the selfish node updates its height to be an integer between $h_{\max}$ and $h_{\min}$. Note that by doing so, the selfish node does not become a sink and also none of the neighbors become sinks either. However, by changing directions of links new nodes may become selfish which also need to adjust their height, and so this process may propagate further in the network.

SNBM leads to a reduction in the height of the selfish node and ensures that the direction of at least one outgoing link is reversed. The amount by which the *selfishness* of the node decreases depends on the value of the new height. The lower the new height is, the more outgoing links of the selfish node are reversed. However if node i sets its height below $h_{\min_i}$, then the direction of all the links would be reversed and i would become sink, and thus not forwarding any traffic from neighbors. (This is why a selfish does not do anything if all its neighbors have the same height or it has only one neighbor.) The aggressiveness of SNBM depends on how the heights of the selfish nodes are updated. Let h_{new} denote the updated height. SNBM is highly aggressive if $h_{\text{new}} = h_{\min} + 1$ and least aggressive if $h_{\text{new}} = h_{\max} - 1$. In the former approach SNBM updates the height to be the minimum height while ensuring that the node does not become a sink. In the latter approach SNBM updates the height to be the maximum height while ensuring that the node is no longer a selfish node. A less extreme approach could be setting h_{new} equal to a random integer between $h_{\min}$ and $h_{\max}$.

The execution of selfish node based mechanism using the most aggressive approach ($h_{\text{new}} = h_{\min} + 1$) is shown in Figure 5. Initially the network has unbalanced load distribution as shown in figure 5(a). Node B is responsible for forwarding packets from nodes $\{B, D, E, F, G, H, I, J\}$ while node C is responsible for forwarding packets from nodes $\{C, K, L\}$ only. This is because node L is a selfish node which partitions the network into isolated routing components. According to the SNBM mechanism, node L updates its height to 6, and reverses the direction of links $L \rightarrow H$ and $L \rightarrow K$, which makes H and K selfish (figure 5(b)). Now node H updates its height to 4 and reverses the direction of $H \rightarrow L$. Similarly node K updates its height to 2 and reverses the direction of link $K \rightarrow L$. This again makes L a selfish node (figure 5(c)). Now L sets its height to be equal to 3 and reverses the direction of links $L \rightarrow H$ and $L \rightarrow G$. This makes nodes H and G selfish. H and G cannot update their height since the neighbors have height difference less than 2. The resulting network, as shown in figure 5(d) is more balanced than the initial network. The node C may be responsible for forwarding the packets from $\{C, K, L, H, G\}$ while node B may be responsible for forwarding the packets from $\{D, E, F, I, J\}$. It should be noted that the mechanism makes a more critical node (L) un-selfish and makes other appropriate nodes selfish (in this case H and G), thus balancing the load.

The selfish node based mechanism terminates because at each step the height of some node is reduced by at least 1 and the lower bound of the height of any node is 1. Also loop freedom is ensured during the execution of the mechanism since Condition (1) is never violated. A nice side effect of this mechanism is that it would prevent the height of the nodes from becoming arbitrarily large with time.

Although this mechanism solves the problem of unbal-

anced load caused by selfish nodes, all cases of load unbalance do not involve selfish nodes. For example, Figure 6(a) shows an unbalanced load scenario (caused by node L) where there is no selfish node. So there is a need of mechanisms that do not rely on the presence of selfish nodes.

5.2 Proactive decrease in height (PDH)

We have earlier observed that the load unbalance is caused because a node that is capable of relieving the highly loaded nodes has higher height than the surrounding nodes so that it does not attract enough traffic. This is because of the fact that in the LRR algorithm the height of a node is a non-decreasing function of time. So one way solving this problem is to decrease the height of the nodes whenever possible. This is the idea behind the next load balancing mechanism, called the *proactive decrease in height (PDH)* mechanism.

The PDH mechanism is very simple. Periodically each node (selfish or not) compares its height with the height of its neighbors. If the height of a node is greater than $h_{\min}+1$, then the node sets its height equal to $h_{\min}+1$. In other words, each node (selfish or not) maintains the smallest possible height that is guaranteed to not make it a sink. The height update is accompanied by appropriate adjustments in the directions of its links such that Condition (1) is maintained.

The execution of the algorithm is illustrated in Figure 6. As shown in Figure 6(a), initially the load in the network is unbalanced. Note that this kind of load unbalance cannot be fixed by the selfish node based mechanism since there is no selfish node in the network (except M, but it has only one neighbor). Node C initiates the PDH mechanism and sets its height equal to 1, as shown in Figure 6(b). Then nodes K and H execute the PDH mechanism and set their height equal to 2 and 4 respectively (Figure 6(c)). Similarly node L executed PDH and sets its height equal to 3. This results in reversal of the direction of links $L \rightarrow H$ and $L \rightarrow G$. Nodes G and M also update their heights to 4. The resulting load balanced network is represented in Figure 6(d).

The PDH mechanism ensures that gradient of the heights is low and uniform throughout the DAG. Since property (1) is never violated, loop freedom is maintained throughout the execution of the mechanism. The PDH mechanism ensures that the height of the nodes do not become arbitrarily high. The PDH mechanism terminates since in each step the height of some node decreases at least by 1 and the lower bound of height of a node is 0.

5.3 Reactive increase in height (RIH)

Both the SNBM and PDH mechanisms are proactive in nature. This means that the nodes that employ these mechanisms perform some actions when certain local conditions are true. The node hopes that the actions performed by it would prevent load unbalance in the network. However, in the proactive mechanisms, a node may take an action even when there is no load unbalance, or even when there is no threat of load unbalance. This wastes the network resources and may lead to high overheads. For example consider figure 6(a). In this figure although the DAG is unbalanced but some of the nodes that have paths through B may never transmit any packets to the destination. Thus node B may actually not be overloaded and therefore there might be no need for invoking load balancing mechanisms. This leads to a rather trivial but a critical observation, that is, only nodes that know that the load in the network is unbalanced are the nodes that are actually overloaded. This is the basis of the mechanism presented in this section, called *reactive increase in height (RIH)* mechanism.

The RIH mechanism achieves load balancing by driving traffic away from an overloaded node. As observed earlier, a node attracts traffic if it has low height and stable route to the destination. Thus RIH repels traffic from an overloaded node by increasing the height of the node.

The RIH mechanism is initiated by a node when it detects that it has become overloaded. A node can detect that it is overloaded if the rate of traffic forwarded by it exceeds a certain threshold or if the buffers of the node overflow. If a node discovers that it is overload, then it sets its height equal to $h_{\max}+1$ and reverses all the incoming links. If none of the neighbors of the node become sinks then no traffic needs to be forwarded by the node and it can *relax*. However, if this leads to formation of some sink nodes then those nodes execute the full reversal algorithm in order to establish paths to the destination. This discovery for new routes leads to a more even distribution of load, whenever possible. In the worst case, if the overloaded node is the only node in the connected component that has a route to the destination, then the initial configuration is restored after the full reversal algorithm terminates, since there is no other alternative. If the node remains overloaded after an execution of RIH, then the node can re-initiate RIH only after $T_{\min}$ seconds, where $T_{\min}$ is chosen randomly from the range of fixed parameters T_1 and T_2 (specified in the particular implementation or simulations).

The execution of RIH is illustrated in Figure 7. Suppose that node B discovers that it is overloaded, so according to RIH it sets its height equal to 5 and reverses the direction of links $C \rightarrow B$ and $G \rightarrow B$. As a result of this change, nodes C and G become sinks, as shown is Figure 7(b). So nodes C and G execute the full reversal algorithm in order to come out of the bad state. As a result D becomes sink, as shown in Figure 7(c). The network finally settles in the final state as shown in Figure 7(e). In the final state nodes D, E, G have an alternate path to the destination through F. Thus the load may be more uniformly distributed than in the final state.

The RIH terminates since it basically involves reversal of incoming links at the overloaded nodes and execution of full reversal algorithm, if it is required. The full reversal algorithm is proven to terminate provided the destination is reachable. The disadvantage of this algorithm is that it leads to an increase in the rate of increase of heights of the nodes. Thus, the heights of nodes may become arbitrarily high.

The RIH mechanism may be used in combination with

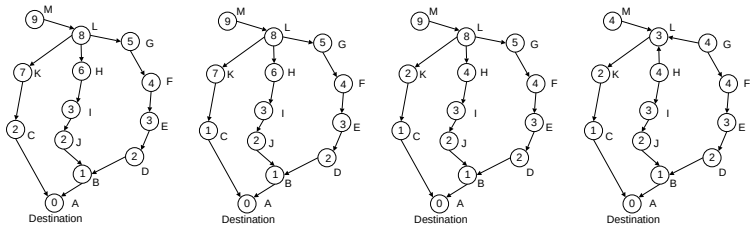


Figure 6. An example showing how proactive decrease in height (PDH) leads to load balancing.

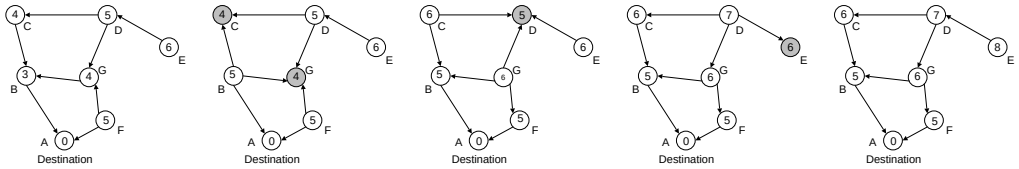


Figure 7. An example showing how reactive increase in heights (RIH) leads to load balancing.

the LRR algorithm in situations, other than load balancing, that require the diversion of traffic from a node. For example if the residual power of a battery powered node falls below a certain threshold then the node may not want to forward the packets of other nodes in order to increase its lifetime. So in order to repel traffic, the node may increase its height according to RIH.

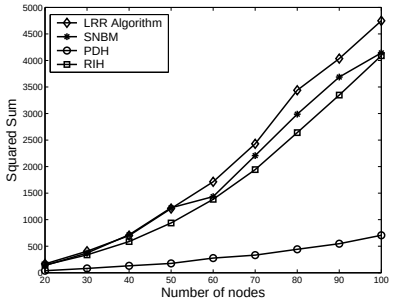


Figure 8. SS for multi-path routing.

6 Simulations

In this section we present simulation results in order to compare the performance of our proposed modifications with the basic LRR algorithm. We consider two extreme forwarding strategies for forwarding traffic over the DAG: (i) Multi-path routing, where each node evenly distributes and forwards traffic over all outgoing links of the DAG, and (ii) Shortest path routing, where each node forwards traffic to the outgoing link which ensures that the packet is routed through the shortest path over the DAG. For the shortest-path simulations we assume that the nodes have sufficient global information in order to identify the outgoing link for shortest path routing, although the LRR algorithms have information only about their immediate neighbors. We consider these extreme forwarding strategies to point out that our modifications for maintaining DAG perform well under a wide range of forwarding strategies. We use the average BF, average SS, network lifetime and the rate of height updates as a measure to assess performance. The network lifetime is defined as the time until some node runs out of power after the network becomes operational.

6.1 Simulation setting

The simulation setting consists of N nodes that are initially distributed randomly over a 1000m $\times$ 1000m square area. The nodes move within the area according to the random way-point mobility model. The velocity of the nodes is uniformly distributed between 2 m/sec and 5 m/sec. The non-zero minimum velocity ensures that the average nodal speed does not decay with time and the network reaches a steady state [19]. The pause time is exponentially distributed with a mean 5 seconds. The communication radius of each node equals $1000\sqrt{\frac{\log N}{N}}$. This radius ensures that the network is connected with high probability [11]. Each node generates traffic at a rate 1 Kbps, which is destined to a stationary sink node located at $(500\text{m}, 500\text{m})$. This simulation scenario models situations where mobile wireless nodes communicate with a central base station, for purposes such as Internet access or for reporting sensed data.

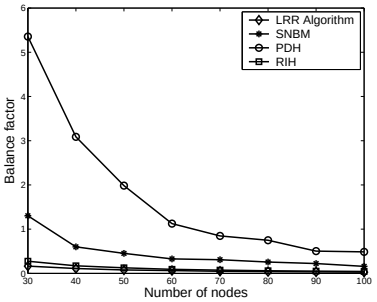


Figure 9. BF for multi-path routing.

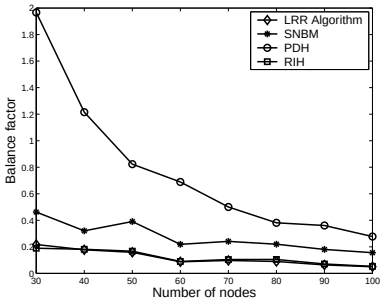


Figure 11. BF for shortest path routing.

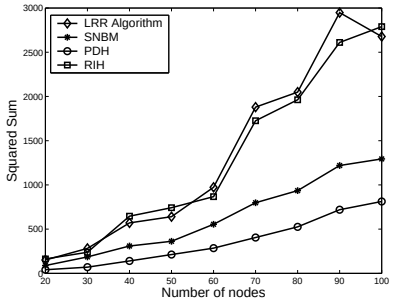


Figure 10. SS for shortest path routing.

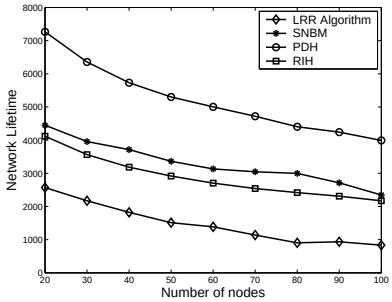


Figure 12. Network lifetime for multipath routing.

For SNBM, the height of selfish node is updated to be a random integer between $h_{\min}$ and $h_{\max}$. For RIH simulations, a node is considered overloaded if the instantaneous traffic exceeds $\sqrt{N}$ Kbps. For RIH simulation $T_{\min}$, the minimum duration between two successive initiations of RIH, is randomly chosen to be between $T_1 = 2$ and $T_2 = 5$ seconds. For the network lifetime simulations, each node has enough battery power to forward and transmit in total 10 Megabits before it runs out of power.

Network partitioning into two or more disconnected components is unlikely to occur in the simulations due to the appropriate choice of communication radius of the nodes. However, we handle the rare occurrences of partitioning so that in components that do not contain the destination the nodes seize to perform link reversals. The nodes in the component that contains the destination continue to perform reversals normally.

6.2 Simulation results

Figures 8 and 9 show the plots of SS and BF, respectively, against the number of nodes in the network. In the figures, the basic full LRR algorithm is compared with the proposed load balancing mechanisms (SNBM, PDH, RIH) when using multi-path routing. The SS is the least for PDH and highest for the basic LRR algorithm. This implies that all the modifications lead to an reduction in the amount of load that is served by the nodes. The BF is highest for PDH and least for basic LRR algorithm. This shows that the modifications to the link reversal algorithm lead to a more uniform distribution of load than the basic LRR algorithm. The SS and BF for the shortest path routing is shown in Figures 10 and 11. The SS and BF for the shortest path routing show the same trend, namely, PDH has the best performance followed by SNBM, while the performance of RIH and basic LRR algorithm is almost the same.

Figure 12 shows the plots of network lifetime in seconds against the number of nodes for the LRR algorithm, SNBM, PDH and RIH. The lifetime result is presented only for multi-path routing scenario. It is observed that the mechanisms proposed in this paper provide significant improvement in the lifetime of the network. The network lifetime is maximum for PDH, followed by SNBM, RIH and the LRR algorithm.

Figure 13 shows the rate of height updates against the pause time of the mobility model. Since in the LRR algorithm, a node generates routing control message only when the height of the node is updated and the routing update is propagated over a single hop only, the overhead is proportional to the rate of height update. It is observed that the basic LRR algorithm has the smallest update rate (since it doesn't do any extra height updates due to load balancing) followed by SNBM, PDH and RIH (which not only have to perform the height updates due to topological changes, but also perform height updates due to the load balancing).

These results indicate that although PDH provides the best load distribution, it leads to a much higher overhead. The SNBM has smaller overhead, but the load distribution achieved by SNBM is also inferior to PDH. RIH is the worst of the proposed mechanisms, it yields the worst performance and causes the highest overhead. The reason for the large overhead of RIH is that the reacting node adjusts in height

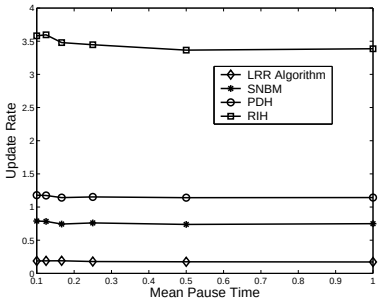


Figure 13. Comparison of number of updates for various versions of link reversal algorithm.

such that it has only has outgoing nodes. This might make some other nodes sinks, leading to a chain reaction of height updates. The other mechanisms, SNBM and PDH, ensure that no sinks are created in the process of height update.

7 Conclusion and future work

In this paper we studied the problem of load balancing in LRR algorithm with the goal of maximizing the lifetime of the network. We proposed three heuristic mechanisms for evenly distributing the load in the full LRR algorithm and used simulations in order to evaluate the performance of the proposed mechanisms in mobile ad hoc wireless networks. For a wide range of network configurations, the PDH mechanism achieves the best performance among the proposed mechanisms.

In this paper we do not consider the effect of multi-path coupling which may lead to a larger energy consumption and packet loss. The basic cause of the multi-path coupling effect in wireless networks is the interference correlation, caused by random access MAC protocols like IEEE 802.11. However, our results would be valid for practical networks with small traffic rates (typical of many sensor networks) and networks with contention-free MAC, as proposed in [1].

In the future, we would like to develop centralized and distributed algorithms for constructing optimal DAG and forwarding strategies, according to the metric defined in Equation (3). We would also include the effects of random access MAC protocols in the performance evaluation of the load balancing mechanisms.

References

[1] C. Busch, M. Magdon-Ismail, F. Sivrikaya, and B. Yener. Contention-free mac protocols for wireless sensor networks. In *DISC*, pages 245–259, 2004.

[2] C. Busch, S. Surapaneni, and S. Tirthapura. Analysis of link reversal routing algorithms for mobile ad hoc networks. In *Proceedings of the SPAA 2003*, pages 210–219, June 2003.

[3] A. Chaudhury. Power aware link reversal routing. Master's thesis, Rensselaer Polytechnic Institute, 2004.

[4] S. Corson and A. Ephremides. A distributed routing algorithm for mobile wireless networks. *ACM/Baltzer Wireless Networks Journal*, 1(1):61–82, February 1995.

[5] E. M. Gafni and D. P. Bertsekas. Distributed algorithm for generating loop-free routes in networks with frequently changing topology. *IEEE Transaction on Communications*, 29(1):11–18, 1981.

[6] Y. Ganjali and A. Keshavarzian. Load balancing in ad hoc networks: Single-path routing vs. multi-path routing. In *INFOCOM*, 2004.

[7] H. Hassanein and A. Zhou. Routing with load balancing in wireless ad hoc networks. In *MSWIM 2001*, pages 89–96, New York, NY, USA, 2001. ACM Press.

[8] P.-H. Hsiao, A. Hwang, H. T. Kung, and D. Vlah. Load balancing routing for wireless access networks. In *INFOCOM*, pages 986–995, 2001.

[9] D. B. Johnson and D. A. Maltz. Dynamic source routing in ad hoc wireless networks. In Imielinski and Korth, editors, *Mobile Computing*, volume 353. Kluwer Academic Publishers, 1996.

[10] E. P. C. Jones, M. Karsten, and P. A. S. Ward. Multipath load balancing in multi-hop wireless networks. In *To appear in Proc. of WiMob 2005*, August.

[11] A. Kumar, D. Manjunath, and J. Kuri. *Communication Networking An Analytical Approach*, chapter 8, pages 456–476. Morgan Kaufman Publishers, 2004.

[12] V. D. Park and M. S. Corson. A highly adaptive distributed routing algorithm for mobile wireless networks. In *Proceedings of the IEEE INFOCOM '97*, 1997.

[13] M. R. Pearlman, Z. J. Haas, P. Sholander, and S. S. Tabrizi. On the impact of alternate path routing for load balancing in mobile ad hoc networks. In *MobiHoc 2000*, pages 3–10, Piscataway, NJ, USA, 2000. IEEE Press.

[14] C. Perkins, E. M. Belding-Royer, and S. R. Das. Ad hoc on-demand distance vector (aodv) routing. Internet Draft, IETF, February 2003.

[15] C. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In *ACM SIGCOMM'94 Conference on Communications Architectures, Protocols and Applications*, pages 234–244, 1994.

[16] C. E. Perkins. *Ad Hoc Networking*. Addison Wesley, 2000.

[17] D. Turgut, B. Turgut, S. Das, and R. Elmasri. Balancing loads in mobile ad hoc networks. In *Proceedings of ICT 2003*.

[18] M. Vainio. Link reversal routing. `citeseer.ist.psu.edu/534655.html`.

[19] J. Yoon, M. Liu, and B. Noble. Random waypoint considered harmful. In *INFOCOM*, 2003.

[20] L. Zhang and J. Gao. Load balanced short path routing in wireless networks. In *INFOCOM*, 2004.