Load-Balancing Routing for Wireless Access Networks

Pai-Hsiang Hsiao, Adon Hwang, H. T. Kung, and Dario Vlah

Division of Engineering and Applied Sciences

Harvard University

Cambridge, MA 02138, U.S.A.

{shawn, adon, htk, dario}@eecs.harvard.edu

Abstract -- **Widespread use of wireless devices presents new challenges for network operators, who need to provide service to ever larger numbers of mobile end users, while ensuring Quality-of-Service guarantees. In this paper we describe a new distributed routing algorithm that performs dynamic load-balancing for wireless access networks. The algorithm constructs a *load-balanced backbone tree*, which simplifies routing and avoids per-destination state for routing and per-flow state for QoS reservations. We evaluate the performance of the algorithm using several metrics including adaptation to mobility, degree of load-balance, bandwidth blocking rate, and convergence speed. We find that the algorithm achieves better network utilization by lowering bandwidth blocking rates than other methods.**

I. INTRODUCTION

Internet appliances equipped with low-cost and short-range radios such as Bluetooth [2] and HomeRF [8] are expected to be widespread. Using these devices, mobile wireless users can access application servers from anywhere. Given the quality of service demands from the users all clamoring for access to the wired network infrastructure, a properly designed routing algorithm can maximize network efficiency and improve performance perceived by end users.

For a wireless network, a routing system can be used for any of the following purposes:

1. The wireless network is deployed for temporary use, or acts as an alternative infrastructure to the conventional wired network. Furthermore, the network is stationary in the sense that network nodes have fixed locations. Such a network can be a sensor network [3, 6, 11] or a rooftop network [1, 18]. In this case, the routing system will configure routes automatically and quickly based on current radio links without necessarily requiring a trusted authority.

2. The wireless network is mobile. Because nodes may move, existing links can be broken and new ones can be created dynamically. The routing system will configure routes to reflect changing network topology.

3. The wireless network is required to provide QoS routes. For a given QoS demand, the routing system will attempt to find a satisfying route.

4. Suppose that traffic over the wireless network changes over time. The routing system will configure routes so that links evenly share current loads to minimize the band-width blocking rate resulting in better network utilization.

Past research in routing for wireless networks, such as *ad hoc* networking [9], has mainly been for purposes 1 and 2. Some work has been for purpose 3, such as [4, 19]. In this paper, we address a new routing problem: finding routes for wireless networks that will satisfy all the four purposes: 1, 2, 3 and 4. In particular, our routing algorithm will perform dynamic load-balancing to achieve 4.

We focus on a special class of wireless networks, namely, "wireless access networks." Via such a network, an "end node," such as an information appliance, can send packets to and receive packets from an "egress node" that connects to the external networking infrastructure. (We say two nodes are "connected" when the radio link between them is up.) Thus, a wireless access network is for carrying traffic between end nodes and gateway egress nodes in wireless stub networks.

We describe routing methods for wireless access networks that will route packets over a "backbone tree" rooted at the egress node. As to be shown, this tree-based approach will greatly simplify routing by avoiding per-flow state, and will support dynamic load-balancing.

This paper makes several contributions. We define the load-balancing routing problem for wireless access networks (Section II). We present a distributed algorithm for load-balancing in Section III. To evaluate performance, we show simulation results in Section IV. We find that our algorithm performs well in a variety of network sizes and mobility conditions. In comparison to existing QoS methods, we find that our algorithm achieves better load-balance, and a lower bandwidth blocking rate in Section V. We conclude the paper in Section VI.

II. LOAD-BALANCING ROUTING PROBLEM

We give a formulation for the load-balancing routing problem addressed in this paper.

II. A. Definitions and Assumptions

A "topology graph" as depicted in Fig. 1 is the unit graph

E: egress node
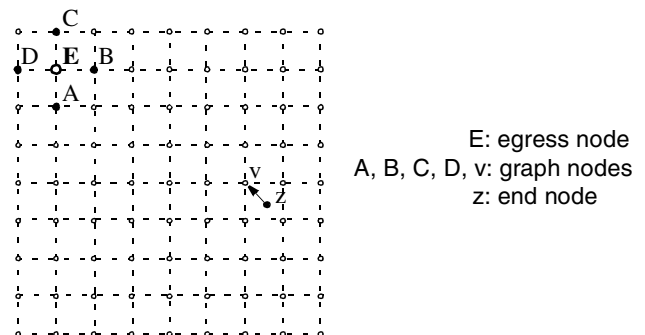A, B, C, D, v: graph nodes
z: end node

Figure 1: A topology graph. Graph nodes are equipped with 4 unit-range radio interfaces. There is no backbone tree, yet.

corresponding to a wireless access network. We assume that the graph has N nodes, which we call "graph nodes." Every graph node has one or more radio interfaces, each capable of connecting to another graph node, an end node (defined below), or an external node outside the wireless access network. "Neighbors" of a node are other nodes that are in radio range.

In this paper we make the following simplifying assumptions. In order to model wireless connections between neighbors as isolated point-to-point links, we assume that graph nodes have separate interfaces connecting to each neighbor. Such assumptions are not unreasonable [12] and may become increasingly popular as low-cost radios such as Bluetooth emerge. We use regular graphs in which each graph node has 4 or 6 interfaces connecting to other graph nodes when the topology graph is a square or hexagonal mesh, respectively. In addition, each graph node has an additional interface for connecting to an end node.

An "egress node" is a graph node that has connections to external networks. We assume that the graph has only a single egress node, which we denote as E in the figures.

An "end node" is an application host or appliance connecting to a graph node. We assume that there are X end nodes where $X \leq N$. Each graph node can connect to at most one end node.

There are three kinds of links. These are graph-node, external-node and end-node links each connecting a graph node to another graph node, an external node, or an end node, respectively. Note that the topology graph (Fig. 1 for example) is composed of graph nodes and graph-node links. External-node and end-node links are outside the topology graph.

We assume that graph-node and end-node links have the same bandwidth capacities GC and EC, respectively. External-node links have sufficient capacity XC such that $XC \geq k \cdot GC$ where k is the maximum degree of a graph node. (For example, $k = 4$ in Fig. 2.) A link is either up achieving its full link bandwidth GC or EC, or down achieving zero bandwidth.

II. B. Load and Load-balancing

A "load" is a flow from an end node to an egress node, or vice versa. We assume that each load has an upper bound bandwidth specification at the time of the request. The size of a load is the bandwidth of the associated request. We denote total bandwidth flowing through node X as f_X.

For a given a set of loads, a "backbone tree" is a tree subgraph rooted at an egress node, in the topology graph, such that for each load, its end node is connected to a tree node. We say that an end node "connects to a tree node" to denote an end node establishing a connection between itself and the tree node.

A "fully load-balanced tree" is a backbone tree for a set of loads such that, for each tree node with multiple branches, all the branches carry the same total amount of loads.

A "top load-balanced tree" is a backbone tree for a set of loads such that, for the tree node that has multiple branches and is closest to the root, all the branches carry the same total amount of loads.

A "top subtree" is a branch of the highest-level tree node with multiple branches. We denote a top subtree rooted at node X by t_X.

"Adjacent top subtrees" are pairs of top subtrees that have some nodes that are neighbors of nodes in the other top subtree.

For a given topology graph and set of loads, "load-balancing routing" is to find a fully load-balanced or top load-balanced tree for the set of loads. We focus on top load-balanced trees because for many situations, the egress node is the primary bottleneck. Given a backbone tree and a new set of loads, "rebalancing" is to add or delete connections in the tree to derive a load-balanced or top load-balanced tree for the new load distribution.

This paper addresses top load-balanced trees only, except in Section II. C. 1 which illustrates a fully load-balanced tree.

II. C. Load-Balancing Tree Routing Examples

We illustrate the concept of load-balancing tree routing and the associated terminology.

II. C. 1. Fully load-balanced H-tree for a square mesh

Consider the case when the topology graph is an $N = n \times n$ square mesh, where n is an odd integer. Fig. 1 depicts such a mesh with $n = 7$. Suppose that there are $[(n+1)/2]^2$ loads, where each load is a unit flow with its end node in one of $[(n+1)/2]^2$ 2x2 squares. Thus, for the 7x7 mesh of Fig. 2, there are 16 loads, each with its end node in one of the 16 2x2 squares.

Assume that each end node is connected to the tree leaf at the center of the associated 2x2 square. From Fig. 2, it is easy to see that the H-tree provides fully load-balanced routing for these loads. (An H-tree is a layout known in VLSI chip design [15].)

II. C. 2. Top load-balanced tree for a square mesh

Fig. 3 shows a top load-balanced tree for a square mesh, where each graph node covers a 1x1 square centered at the node, and there is a unit flow originating from each 1x1 square. Using algorithms from Section III, this tree is obtained by a breadth-first backbone tree construction algorithm followed by a best-first rebalancing algorithm. Unlike the H-tree case, this tree construction does not require a priori knowledge about locations of graph nodes. For example, as depicted in Fig. 3, the construction can start from a root node which is not in the
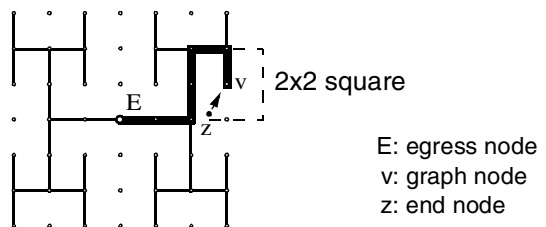

2x2 square

E: egress node
v: graph node
z: end node

Figure 2: Fully load-balanced H-tree routing for a 7x7 mesh. Load is divided equally at each branch point of the tree. Each tree leaf node "covers" a 2x2 square in the grid. The bold line indicates a load from end node Z.
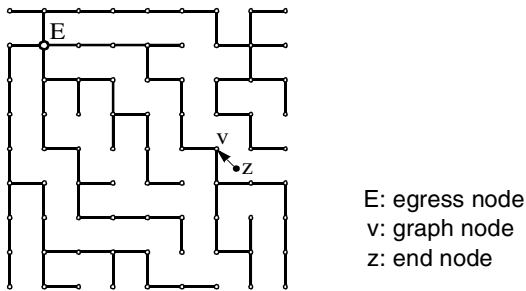
Figure 3: Top load-balanced tree for a square mesh. In this particular example, each top subtree of the egress node covers 20 graph nodes.

E: egress node
v: graph node
z: end node

center of the layout. In addition, unlike the H-tree case, end nodes may connect to any tree node, not just tree leaf nodes.

III. LOAD-BALANCING ALGORITHM

III. A. Use of Backbone Tree

We use a tree topology because it easily allows aggregation and can avoid per-flow state and per-destination state in the routers. The routers do not maintain state information other than about their immediate neighbors. As noted earlier, our traffic model assumes that the primary mode of communication from wireless access networks will be access to the wired Internet.

Conventional per-destination routing information is not necessary. Outgoing packets from the nodes simply follow the tree toward the root, the egress node. Incoming packets follow explicit source-routed paths given by the egress nodes, or they follow hierarchical addresses [16].

The intermediate routers aggregate load amounts toward the root of the tree such that they do not maintain per-flow routing and reservation state. Once flows join, they never fork to different paths. Thus, per-flow state required to maintain such reservations in conventional networks can be discarded in a backbone tree network.

One additional benefit of using a backbone tree is elimination of routing loops.

III. B. Underlying Concepts of the Load-balancing Algorithm

For a given set of loads, our load-balancing routing algorithm will attempt to construct a top load-balanced tree. This is illustrated by the example in Fig. 4. Here, we describe the non-distributed algorithms. In Section III. C, we will describe their distributed implementations.

III. B. 1. Balance Index

For our evaluation of the degree of load balance, we define the "balance index" to be a fairness index among loads in the top subtrees [5]:

$$\beta(E) = \frac{\left(\sum f_{X_i}\right)^2}{k \sum f_{X_i}^2} \qquad (1)$$

where X_i are roots of top subtrees, f_{X_i} is the aggregate load of X_i to the egress E, and k is the number of top subtrees. The balance index has the desired property that it tends to 1 when top subtree loads are more equal while it approaches $1/n$ when the imbalance is large.

The balance index of Eq. (1) satisfies a *monotonic property*, on which our greedy load-balancing heuristics described below depend. That is, for any pair of f_X and f_Y of different values, if another pair with a smaller difference in their values replaces the pair, then the balance index will improve. More precisely, suppose that δ is any value with $0 < \delta < f_X - f_Y$. Then, it follows from the definition of $\beta(E)$ by Eq. (1) that if f_X and f_Y are replaced with $f_X - \delta$ and $f_Y + \delta$ respectively, then $\beta(E)$ will be increased.

III. B. 2. Basic Mechanisms

Suppose the backbone tree starts with similar numbers of nodes in the top subtrees. As soon as flows enter the network, the backbone tree is certain to become unbalanced. During subsequent rebalancing, edges of the backbone tree are altered until the tree is top load-balanced.

More specifically, nodes in the top subtrees with higher load attach to nodes of other top subtrees (they "defect"). We call "relaxation" the process of a node examining its neighbors and evaluating whether or not to defect to an adjacent top subtree. For example, in Fig. 4 (a), node X may defect to either node W or Z.

A weight function w defined for top subtrees assists load balancing. A node will defect from top subtree t_X to t_Y if the defection will reduce the difference between $w(t_X)$ and $w(t_Y)$. The weight function is so defined that reducing the difference between $w(t_X)$ and $w(t_Y)$ will lead to reducing the difference between the load on t_X and that on t_Y. This in turn implies an increase of the balance index of Eq. (1) due to the monotonic
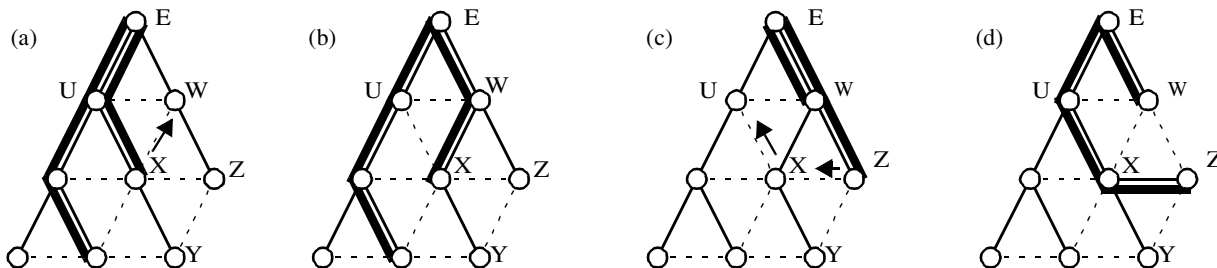

Figure 4: Running example for top load-balancing on a hexagonal mesh topology graph. Edges of the current backbone tree are denoted by solid lines. Flows are indicated by bold solid lines. (a) An unbalanced tree. There are two top subtrees, one carrying two flows and the other zero. The arrow indicates node X's defection to connect to node W. This results in the next diagram (b) showing a top load-balanced tree, where both top subtrees carry the same amount of flow. (c) The loads change. The top subtrees are no longer balanced. (d) After two rebalancing steps indicated by the arrows in (c), top subtrees are balanced again.

property described earlier.

Rebalancing occurs in iterations. Each iteration is an attempt to find a node to defect. A set of iterations continues until a "stopping criterion" where no such node is found. A new set of iterations may begin again after changes in load.

Nodes that do not border other top subtrees (nodes internal to the subtree) do not defect, as their decisions to alter the tree topology do not have any bearing on the load balance of the top subtrees. Thus, only border nodes can defect to adjacent top subtrees.

For example, in Fig. 4 (a), the backbone tree selects node X to relax. Node X examines whether or not to defect to an adjacent top subtree (change edge X-U to X-W or X-Z). Using the total load in the top subtree as the weight function, the current weight for t_U (top subtree rooted at U) is 2, and the weight for t_W is 0. The difference between the two weights is 2. If X connects to W or Z, the weights for both t_U and t_W become 1. The new weight difference between the adjacent top subtrees thus becomes 0. By X defecting, the difference of weights decreases, making the balance index 1. Thus, in Fig. 4 (b), X defects and connects to t_W (shorter the path to the root the better).

Rebalancing adapts to changes in load. Between Fig. 4 (b) and (c), the loads in the tree have changed. How should the tree rebalance? There is not an immediate solution from one node defecting. We use a different weight function $w'(t_A) = m \cdot f_A + \#\ nodes\ in\ t_A$, where f_A is the amount of aggregate flows passing through t_A, and m is a constant larger than the maximum possible number of nodes in the network. Using $w'(t_U)$ and $w'(t_W)$, when node X relaxes, it can compute the outcome of defecting to t_U as seen in Table 1. Thus, X defects to U even though it carries no load (change in difference of w' is -4). Next, Z attaches to X (change in difference of w' is -10), making the backbone tree of Fig. 4 (d) balanced with respect to the balance index of Eq. (1).

In order to minimize the path lengths in the backbone tree, in addition to rebalancing, graph nodes connect to neighbors with shorter hop-distance to the root than their current parent node, only if the neighbor belongs to the same top subtree.

TABLE 1 : Defection outcome evaluation in Fig. 4 (c) and (d)

	Tree (c)	Tree (c) with X defecting to t_U	Tree (d) with Z defecting to t_U
load in t_U	0	0	1
nodes in t_U	4	6	7
$w'(t_U) = 10 \cdot\ load + \# nodes$	4	6	17
load in t_W	2	2	1
nodes in t_W	4	2	1
$w'(t_W) = 10 \cdot\ load + \# nodes$	24	22	11
$\Delta w' = w'(t_W) - w'(t_U)$	20	16	6
$\beta(E)$	1/2	1/2	1
change in $\Delta w'$		**-4**	**-10**

As far as load-balancing is concerned, we expect a top load-balanced tree to be better than conventional QoS mechanisms that first find shortest paths between end points and then perform resource reservation. However, since our algorithm focuses primarily on the load balance of the top subtrees, resulting paths to the egress node may be longer than the shortest paths.

III. B. 3. Heuristics

There are many possibilities for the heuristic in selecting nodes for relaxation. We have considered three heuristics: *best-first*, *random*, and *weighted*.

The *best-first* is a greedy heuristic which selects the node whose defection can improve the balance index the most. This requires global knowledge. In *random*, among the potential nodes, one is selected at random for relaxation. In *weighted*, a candidate node is selected at random and relaxes with probability increasing with its subtree size. This is more likely to relax a node with a larger subtree in hopes that its defection may lead to a larger improvement of the balance index. The *random* and *weighted* heuristics use only local information.

There are times when the algorithm stops in a local maximum of the balance index due to the greedy nature of node relaxation. For example, the balance index for Fig. 5 (a) is clearly not at its maximum. To obtain the top load-balanced tree of Fig. 5 (b), a sequence of events that violate the greedy heuristic must occur, such as X defecting to t_U and Z defecting to t_W. The weight function $w'(t_A) = m \cdot f_A + \#\ nodes\ in\ t_A$ disallows this, since X's defection to t_U will not reduce $\Delta w'$ as shown in Table 2.

To solve this problem, we use principles from simulated annealing [13] to occasionally violate the greedy heuristic and perturb the tree in hopes that a global optimum is reached. With a small probability, nodes that carry a load may defect to adjacent top subtrees even though they deem that the balance index does not improve. Section IV. D contains simulation results for simulated annealing.

III. C. Distributed Implementation

The distributed algorithm is a straightforward implementation of the static algorithm. We cite the differences here.

In a distributed system, each node joins the egress-rooted tree by attaching to a neighbor whose hop-distance to the root is the minimum. Once flows emerge, the rebalancing algorithm will alter this topology as mentioned above.
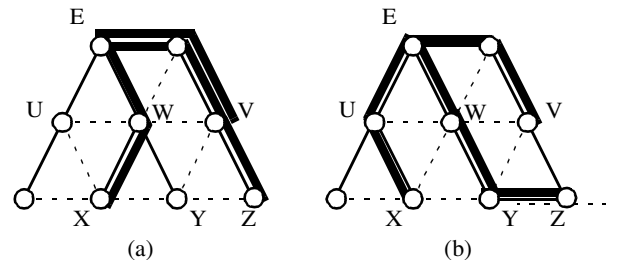

Figure 5: (a) Local optimum of backbone tree. If node X defects to t_U, then Z can defect to t_W resulting in (b) where top load-balance is achieved. However, the weight function w' prevents this from happening, since X's defection to t_U will not reduce $\Delta w'$ as shown in Table 2.

	Tree (a)	Tree (a) with X defecting to t_U
load in t_U	0	1
nodes in t_U	2	3
$w'(t_U) = 10 \cdot load + \# nodes$	2	13
load in t_W	1	0
nodes in t_W	3	2
w'(t_W)	13	2
$\Delta w' = w'(t_W) - w'(t_U)$	11	11
$\beta(E)$	9/15	9/15
change in $\Delta w'$		**0**

Each node X in the distributed computation collects information necessary for making defecting decisions. This information is the number of nodes and total load of all top subtrees, and the number and total load of X's descendants. Information about the top subtrees is flooded to the nodes by the egress node. Information about X's descendants is reported by X's children. X sums these reports and sends the totals to its parent.

Each node could relax and defect independently of other nodes in the network. However, since information that relaxation is based on can be stale, we use a synchronization mechanism to avoid loop formation and backbone tree destruction.

The egress node floods a "relaxation notification" to a single top subtree at a time. Upon receiving the notification each node relaxes and sends a reply if by defecting, it can improve the load balance. Finally, the egress node selects one of the replying nodes and signals it to defect. This mechanism ensures that at most one node defects per rebalancing iteration.

IV. SIMULATION RESULTS

We have built a simulator incorporating our topology graph and load models, the distributed and non-distributed load-balancing algorithms as well as end node mobility.

IV. A. Performance Evaluation Criteria

Several criteria are used to evaluate the performance of the load-balancing routing algorithm. First, the rebalancing algorithm should converge such that a stopping criterion will be reached. Second, convergence should occur quickly. That is, the number of rebalancing iterations should be small. Third, the algorithm should be able to adapt to load migration, even when the load change is large. Here we evaluate our load-balancing routing algorithm against these criteria, based on simulation results.

IV. B. Rebalancing with Load Mobility

With mobility in the end nodes, loads will migrate. To evaluate the success of adaptation, we focus only on mobility not on load arrivals and departures.

We use the balance index and bandwidth blocking rate to evaluate the performance of our algorithm. Balance index is defined in Section III. B. 1., and bandwidth blocking rate is the fraction of bandwidth demand rejected.

IV. B. 1. Mobility Model

Our mobility model has two parameters: speed and instability. *Speed* is how far an end node can move in a unit time, whereas *instability* is how frequently an end node can move to different graph nodes. In our simulation, we start one iteration every time unit (e.g., 1 second). We assume an iteration completes within this time unit. We define speed n such that in one time unit, a load's end node can move from its current position to another position at most n graph node-hops away, similar to Brownian motion of particles. The loads in the network move with end nodes, independent of graph nodes. We define instability x, or probability of movement, such that a load's end node will move with its pre-determined speed to another position with probability x between iterations. The instability is analogous to the pause time parameter of the *random waypoint* model [10]. Instead of end nodes pausing for a fixed duration, they are stationary with a certain probability, namely *1-x*.

In our simulations, the topology graph has 81 nodes in a square 9x9 grid where each graph node has 4 radios, with one for each of their neighbors. The egress node is at the center of the topology graph. The initial tree graph is balanced (Section III. B. 2). We place 40 end nodes with unit loads randomly in the graph.

The end nodes move according to the mobility model above. Rebalancing uses $w'(t)$ as defined in Section III. B. 2. for the weight function and the *weighted* heuristic of Section III. B. 3 for node relaxation.

IV. B. 2. Rebalancing Keeps the Balance Index High

The first set of experiments in Fig. 6 shows that rebalancing can keep the balance index as defined in Section III. B. 1. at 1 under moderate mobility. The figure shows two experiments with the same mobility pattern. To remove the effects of bandwidth blocking, the capacity of each radio is 40, each enough to accommodate all the 40 loads in the network. Each run lasts for 500 iterations. The runs use mobility parameters of speed 1 and instability 0.1. If an iteration lasts one second, and the
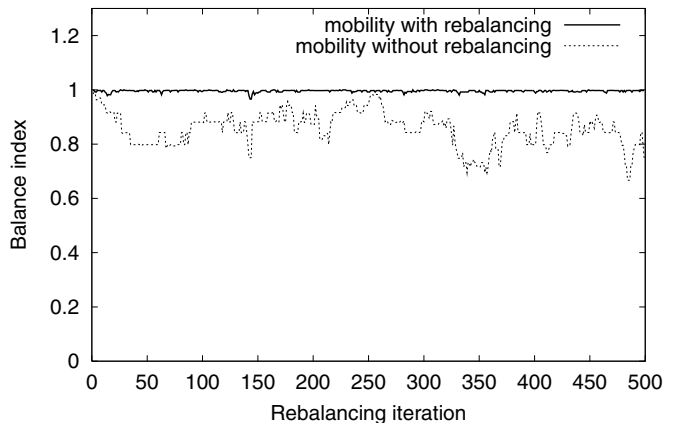

Figure 6: Change history of the balance index as rebalancing iterations progress. End nodes move every rebalance iteration with speed 1 and instability 0.1. Rebalancing improves the balance index.

radio range is 250 meters, then the average velocity of an end node is 25 meters per second. An average of 4 end nodes move every second (instability 0.1).

Mobility alone changes the balance index as indicated in the run without rebalancing of Fig. 6. The index does not decrease without bounds because movement of nodes shifts the loads rather randomly according to the mobility model. However, with rebalancing at each iteration, the balance index remains within a much smaller range.

IV. B. 3. Rebalancing Keeps the Bandwidth Blocking Rate Low

In the second set of mobility experiments the bandwidth blocking rate is lowered with rebalancing. This is the motivation of our work. Here, the capacity of each radio is 10 load units. There is no bandwidth blocking except at the egress node links.

Not all of the 40 end nodes can have their required bandwidth satisfied in the beginning because some top subtrees have more than 10 end nodes—more load than the radio capacity. We show the bandwidth blocking rate of this initial placement as the line "initial" in Fig. 7. End nodes that do not have their load accepted will retry after each iteration.

Rebalancing significantly improves the bandwidth blocking rate. When the speed parameter is 1, the difference in bandwidth blocking rate to no rebalancing is large even under relatively low instability (Fig. 7). Higher speed parameters will lead to larger bandwidth blocking rates. When end nodes migrate to different top subtrees due to high speeds, it is not likely that the top subtrees can accommodate the loads without rebalancing. The maximum speed we show in Fig. 7 is 5. Because the network diameter is 9, a larger speed will make the placement of end nodes between iterations look completely random, and the bandwidth blocking rate will have similar results to the "initial" placement in the figure. As indicated in the left side of the figure, if the end node instability is lower than 0.06, rebalancing can maintain a lower bandwidth blocking rate than the initial placement, even with a speed of 5.

IV. C. Local Convergence Speed for Rebalancing Heuristics

In this section we compare the speed of convergence for the three rebalancing heuristics we mentioned in Section III. B. 3. Suppose that we are given a square mesh and a set of loads. Starting from an initial backbone tree, we measure how quickly these heuristics will settle down on a new backbone tree which will achieve some local maximum for the balance index.

Fig. 8 shows the number of iterations it takes to reach the stopping criterion for the three rebalancing heuristics. Suppose that randomly sized loads are randomly placed on the square mesh. The capacity of each radio is scaled to remove the effect of bandwidth blocking as in Section IV. B. 2. The simulation starts from a backbone tree with its top subtrees having approximately the same numbers of nodes. There is exactly one defection per iteration till the stopping criterion is met.

In Fig. 8 (a), the topology graph is a square grid with increasing number of graph nodes (5x5, 10x10, 15x15, etc.), and each end node connects to one graph node. The egress is in the center. The initial configuration is nearly load-balanced. All three heuristics reach balance index 1 relatively quickly.

Best-first converges at a near-constant number of iterations regardless of the size of the topology graph. However, the *best-first* heuristic is impractical in a distributed implementation

(a)

(b)

Figure 8: Speed of convergence. (a) The number of iterations for convergence as the number of nodes in the topology graph increases. Each graph node connects to an end node. (b) The number of iterations for convergence as the number of loads increases, for a 30x30 graph.
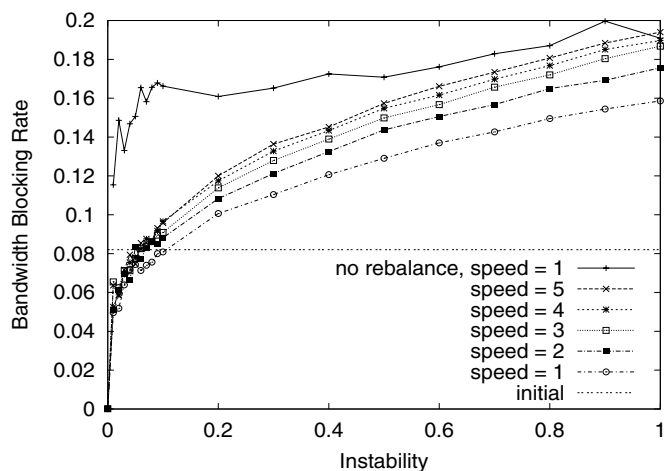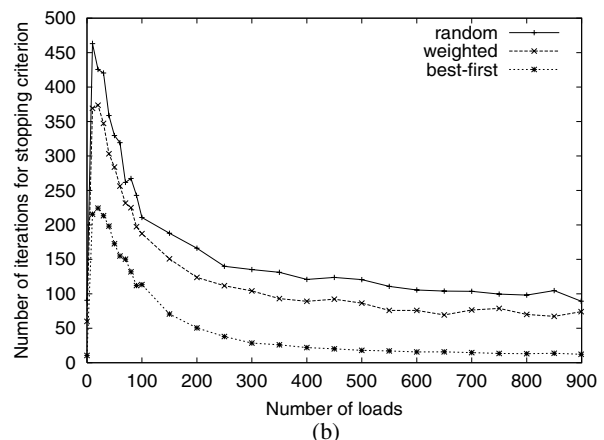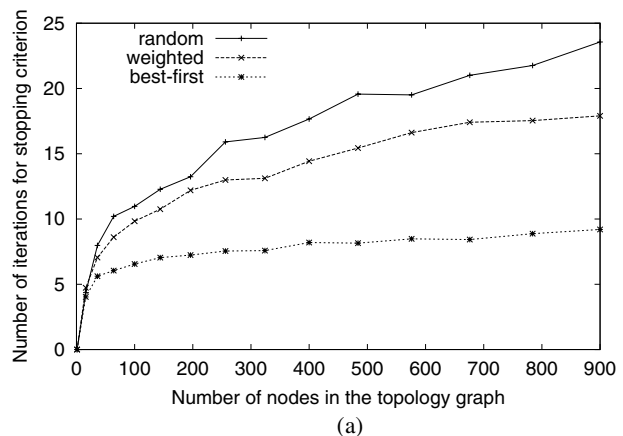

Figure 7: Bandwidth blocking rate as the end nodes increase their mobility with different speed parameters. With instability less than 0.06, rebalancing lowers the bandwidth blocking rate below that of the initial placement.
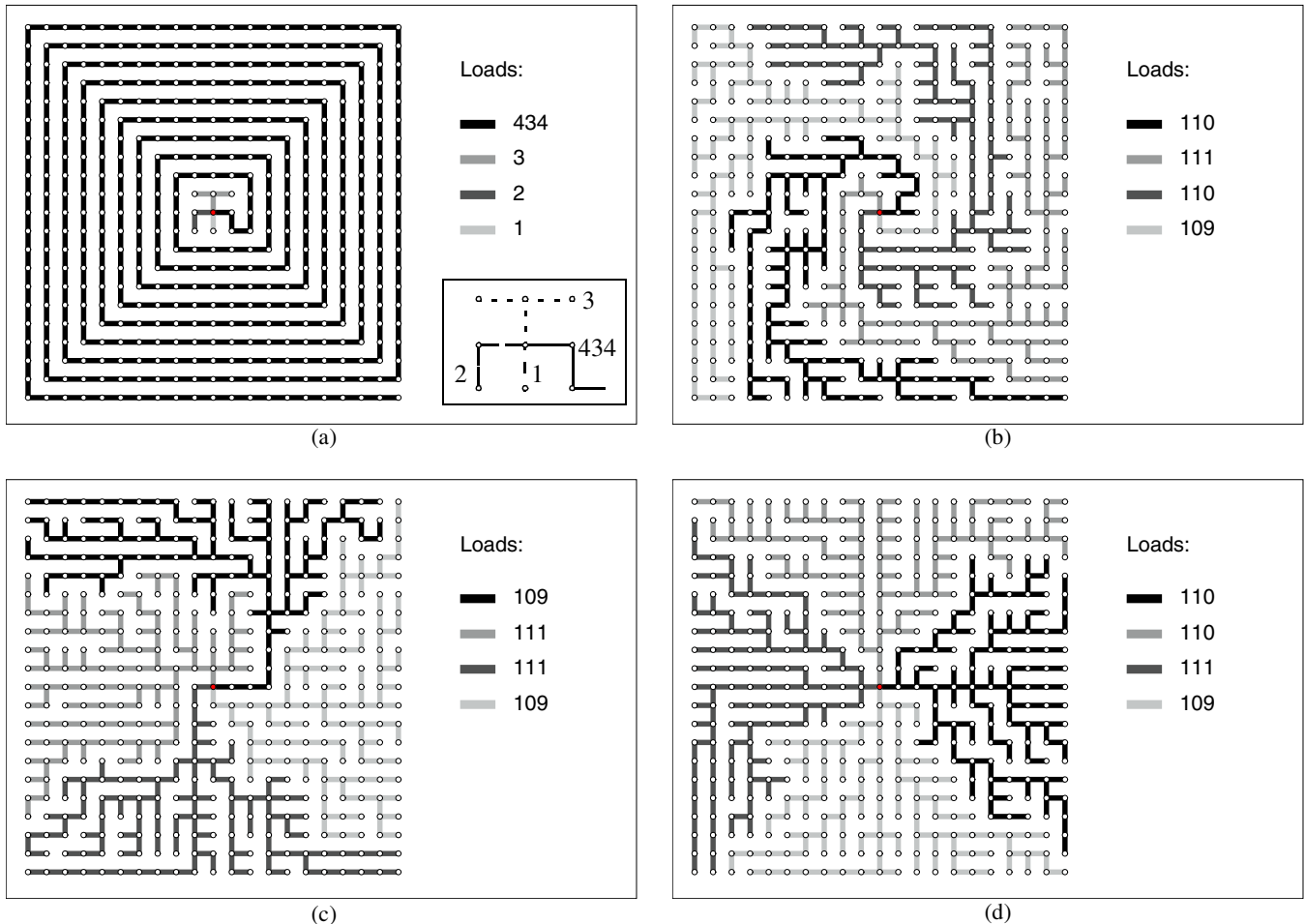
Figure 9: Spiral backbone tree example. (a) the initial spiral backbone tree with 4 top subtrees. (b), (c), and (d) show the backbone tree after 100, 500, and 2000 rebalancing iterations using *random* relaxation and simulated annealing.

because it requires global knowledge of the complete topology graph. It is shown here as a best case, for the purpose of comparison. Fig. 8 (a) shows that the *weighted* heuristic performs somewhat better than the *random* heuristic because its probabilistic defection makes it behave more similar to *best-first*.

In Fig. 8 (b), the topology graph has 900 nodes (30x30) with random placement of random load sizes. The figure shows the convergence speed as the number of loads in the network increases. In this scenario, the capacity of the egress is set to the total load in the network. The capacity of each radio interface of the egress node is one quarter of this aggregate load. Thus, as the number of loads increases, the size of each load decreases in comparison to the egress capacity.

When the number of flows is small, chances are that the paths of loads do not lie on borders between top subtrees. Many graph nodes that do not carry loads have to defect to adjacent top subtrees before a graph node that does carry a load may get a chance to defect. This explains the peak in the number of convergence iterations when the number of loads is low in Fig. 8 (b). As the number of loads increases, more and more graph nodes carry a load, and more lie on the borders, so convergence occurs more quickly than before.

IV. D. Stress Test: Rebalancing a Worst-case Tree

As shown in Fig. 9 (a), a spiral backbone tree is a particularly unbalanced tree. Suppose the initial backbone tree starts out as this 21x21 node tree with every graph node supporting a load. The tree initially has 434, 3, 2 and 1 nodes in its four subtrees as indicated by the boxed magnification. The rest of the figure shows snapshots of the rebalancing algorithm in effect. Nodes relax using the *random* heuristic with simulated annealing. After 100 iterations, the tree is load-balanced but not particularly efficient in path lengths (Fig. 9 (b)). After 2000 iterations the spiral is unwound (Fig. 9 (d)). Fig. 10 shows the balance index as the rebalancing iterations progress on the initial spiral backbone tree. The downward spikes in the balance index indicate the simulated annealing perturbations. This demonstrates that the load-balancing algorithm can cope with such a worst-case tree.

V. COMPARISON WITH OTHER RESEARCH

In this section we compare our load balancing algorithm to two common QoS path-finding algorithms: *shortest-widest*, and *widest-shortest* path.

The shortest-widest path algorithm [21] is used in wireless QoS routing networks such as Cedar [20] and ticket-based

probing [4]. It finds a set of paths with most available bandwidth (the widest paths), and then selects the shortest one among them. The widest-shortest path algorithm proceeds in opposite order: it finds a set of shortest paths first, of which it selects the widest one. A more complete discussion of the two methods appears in [15].

We present two sets of simulation results in which we compare our load balancing algorithm with the QoS path-finding algorithms. In the first set, we study bandwidth blocking rates. In the second set, we show load balancing performance by comparing balance indices.

In both cases, we generate loads in a 9x9 grid for a desired number of loads N and offered load ρ. First, sizes of the N loads are chosen randomly from the range [0,1], and the loads are placed at N random graph nodes. The capacity of the egress node is set to the sum of load sizes divided by ρ. The capacity of graph-node links is one quarter of the egress capacity, which is the egress capacity divided by the degree of the egress node. For $\rho = 1$, the total amount of load equals the total capacity of wireless links at the egress.

V. A. Comparison of Bandwidth Blocking Rates

Similar to [15], we define the bandwidth blocking rate as the fraction of load rejected by a path-finding algorithm. In Fig. 11 we plot the bandwidth blocking rate against the number of loads N, for offered load $\rho = 1$.

If bandwidth blocking at intermediate links is neglected, the problem reduces to bin-packing, where the egress links can be viewed as bins and the loads as weights. The solution always rejects fewer loads than other algorithms which take into account intermediate links. Thus, we provide bin-packing results (using the *first-fit-decreasing (FFD)* heuristic [7]) as a lower bound on the blocking rate. It is not practical to use this heuristic for path finding.

The top load-balancing algorithm achieves the next lowest blocking rate, followed by the widest-shortest and the shortest-widest path selection algorithms. This result is expected; the load-balancing algorithm adjusts paths dynamically, while the other two algorithms select the paths and terminate. To illumi-

Figure 11: After bin-packing, the bandwidth blocking rate is lowest for the load-balancing algorithm, followed by widest-shortest and shortest-widest path selection. The bin-packing solution is presented as a lower bound only. When the number of loads is small, their demands are coarse-grained. This leads to an increased blocking rate, because any rejected load comprises a large fraction of the total demand.
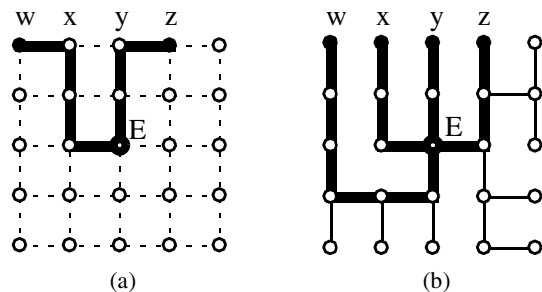
Figure 12: Unit loads are placed at end nodes *w*, x, y and *z*, in a grid with unit capacities. (a) *shortest-widest path:* suppose paths for *w* and *z* are reserved first, indicated by emphasized lines. The loads at *x* and *y* cannot be satisfied, resulting in a 50% blocking rate. (b) *load-balancing:* each load is satisfied, without any blocking.

Figure 13: The best balance is achieved by the load-balancing and the shortest-widest algorithms, followed by the widest-shortest path algorithm.

nate this difference, in Fig. 12 we show a case where the shortest-widest algorithm rejects a large fraction of loads, while load-balancing satisfies all demands.

V. B. Comparison of Load-Balancing Performance

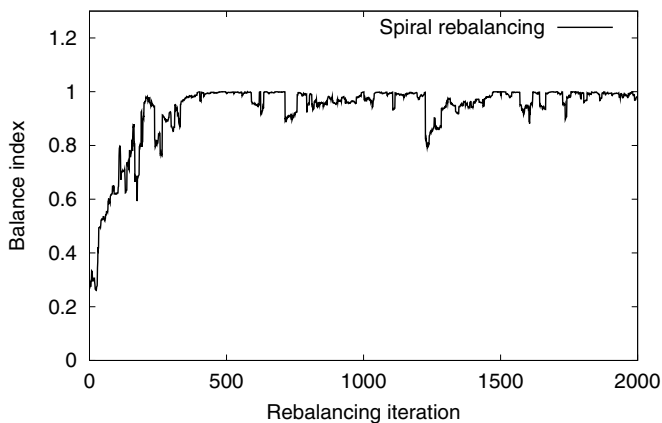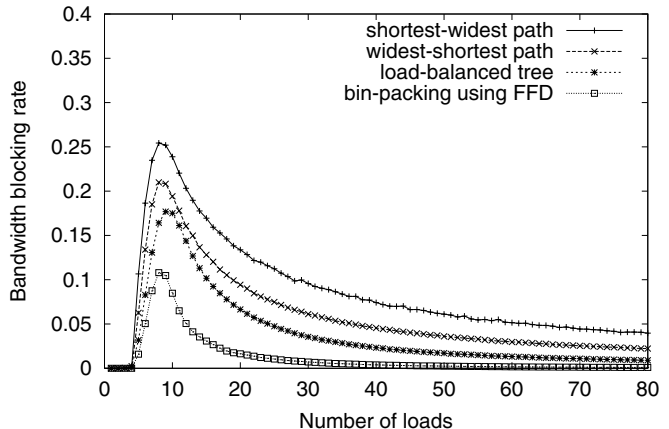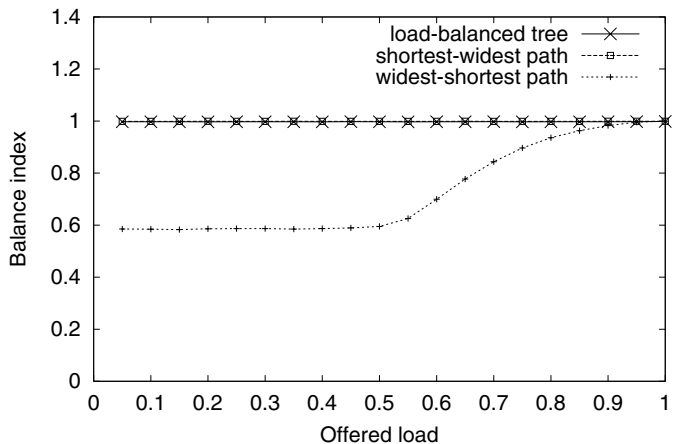In Fig. 13, we evaluate load-balancing properties of shortest-

Figure 10: Change history of the balance index for a spiral backbone tree as rebalancing iterations progress. The downward spikes are due to simulated annealing perturbations. Note that the balance index converges quickly considering the number of edges that need adjustment.

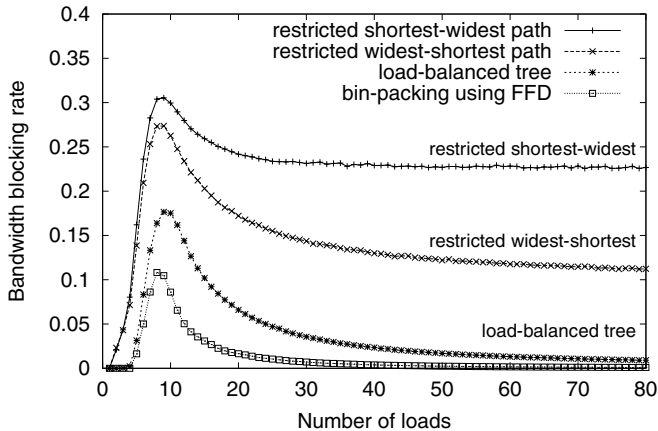Figure 14: The bandwidth blocking rate of the tree-restricted QoS routing algorithms vs. load-balancing algorithm.

Figure 15: The balance index of the tree-restricted QoS routing algorithms vs. load-balancing algorithm.

widest path, widest-shortest path, and our load-balancing algorithm. In order to create an imbalance, the egress node is placed in the top left region of the grid like in Fig. 1. The offered load ρ varies from 0 to 1, with the number of loads fixed at $N = 80$.

In widest-shortest, the majority of the loads such as load f_v from node v in Fig. 1 first follow shortest paths to E through nodes A or B. Once A and B fill up, the loads go through C or D. However, if the offered load ρ is less than 0.5, A and B together never become fully utilized. Thus, those loads do not detour to C or D making the balance index less than 1.

If the offered load ρ is greater than 0.5, A and B do become fully utilized. Thus, those loads that would go through A or B now detour through C or D. As the offered load increases, there tend to be more such loads making the egress node more balanced.

In shortest-widest, as the loads select paths to E, they fill up A, B, C, and D evenly. For example, suppose that node A lies on the "widest" paths to E at a given point in time. Once a load goes through node A, the capacity of A decreases, and A will likely no longer be along the "widest" paths. Instead, another node, C for instance, will now lie along the "widest" paths to E. Thus, in Fig. 13, the balance index of shortest-widest path algorithm tends to be 1. Our load-balanced tree algorithm achieves the same result.

V. C. Tree-restricted Comparison

The previous two subsections compared performance of the top load-balancing algorithm to two general QoS path finding algorithms. The former produces a set of paths which form a tree, while the latter return a graph. In this section, we restrict the two QoS algorithms to trees as well. We justify this with our earlier reasoning about simplicity of routing in tree topologies.

We make a simple change to shortest-widest and widest-shortest path algorithms. Instead of finding a path to the egress node, we alter the algorithms to consider paths to the set of nodes carrying non-zero flow. This prevents paths from crossing, resulting in a tree topology.

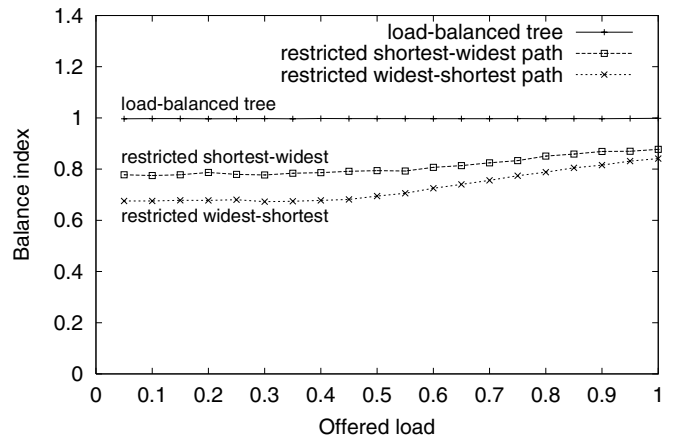In Fig. 14, we show the bandwidth blocking rates of the tree-restricted QoS algorithms in comparison to load balancing. Their performance is worse than the unrestricted versions. Similarly, in Fig. 15 we compare the balance indices of the three algorithms, showing that the topology restriction causes the QoS algorithms to perform worse.

VI. Conclusion

Wireless data networks have been an active area of research. With the expected high growth rate of wireless users, managing the load and utilization for egress links of wireless access networks will be of great concern. In a multi-hop wireless access network, major aggregation points such as the egress gateways to the wired Internet are naturally the chief points of bandwidth contention.

We define a model for a multi-hop wireless access network and propose load-balancing routing for this network to alleviate the egress node bottleneck. We show that it can lower the bandwidth blocking rate compared to widest-shortest and shortest-widest QoS routing algorithms. Our load-balancing routing algorithm lowers the bandwidth blocking rate to maximize network utilization. We define the balance index metric to analyze and assess the state of load balance in the network.

Using a backbone tree for an access network, the graph nodes, or routers, do not maintain per-destination state such as routing tables or per-flow state to support quality of service. The egress node maintains much of the global state for load balancing. It is more likely for a single robust egress node to perform this duty instead of having powerful graph nodes deployed everywhere.

We propose a simple method of balancing loads using a process of node defection. The rebalancing algorithm shifts the path of loads in the tree relatively quickly. The algorithm converges given that the state of the network does not change too rapidly. With mobility in the end nodes, rebalancing reduces the bandwidth blocking rate for the loads. The algorithm can even load-balance a poorly constructed backbone tree such as a spiral.

In comparison to shortest-widest and widest-shortest QoS routing algorithms, our load-balancing routing algorithm has lower bandwidth blocking rates and can keep the egress node's

radio links load-balanced. Unlike the others, our algorithm does not have per-destination or per-flow state. Improving existing QoS routing algorithms to reduce the bandwidth blocking rates entails modification of the online algorithms that place loads on paths. Our solution utilizes a load-balanced tree topology where it is easy to shift loads.

Hand-off is necessary to maintain load paths from the end nodes. However, hand-off issues are not central to this paper; thus, results from existing work (such as [17, 22]) easily apply to this environment.

For future work, we would like to extend the algorithm to work with multiple egress nodes. When many egress nodes exist, multiple backbone trees will span from them. How will the end nodes connect to the appropriate trees to balance the loads in the overall network? Though we have focused on top load-balanced trees, we would like to find and evaluate algorithms for fully load-balanced trees as well. Also, we hope to investigate the impact of stale information on the performance of the routing algorithm. Currently, the distributed algorithm relies on periodic flood updates from the egress node. There has been research on the impact of triggered updates on the correctness and overhead of QoS routing protocols [17]. Our load-balancing routing algorithm may benefit from such extensions as well.

REFERENCES

[1] Beyer, D., Vestrich, M., and Garcia-Luna-Aceves, J. J. "The Rooftop community network: free, high-speed network access for communities," http://ksgwww.harvard.edu/iip/doeconf/beyer.html.

[2] The Bluetooth Special Interest Group. "Specification of the Bluetooth system," December 1999.

[3] Bult, K., Burstein, A., Chang, D., Dong, M., Kaiser, W. J., et al. "Wireless integrated microsensors," in *Proc. of Conference on Sensors and Systems* (Sensors Expo), Apr. 1996.

[4] Chen, S. and Nahrstedt, K. "Distributed quality-of-service routing in ad hoc networks," *IEEE JSAC*, 17(8), Aug. 1999.

[5] Chiu, D., and Jain, R., "Analysis of the increase and decrease algorithm for congestion avoidance in computer networks," Journal of Computer Networks and ISDN, 17(1), June 1989, p. 1-14.

[6] Estrin, D., Govindan, R., Heidemann, J., and Kumar, S. "Scalable coordination in sensor networks," in *Proc. Mobicom '99*, Seattle, WA, pp. 263-270, Aug. 1999.

[7] Garey, M. R. and Johnson, D. S. *Computers and intractability*, San Francisco: W. H. Freeman, 1979.

[8] Home RF. "Technical summary of the SWAP specification," http://www.homerf.org/tech/index.html.

[9] Johnson, D. "Routing in ad hoc networks of mobile hosts," in *Proc. of IEEE workshop on Mobile Computing Systems and Applications*, Dec. 1994.

[10] Johnson, D. and Maltz, D. A. "Dynamic source routing in ad hoc wireless networks," in *Mobile Computing*, ed. T. Imielinski and H. Korth, Ch. 5, pp. 153-181, Kluwer Academic Publishers, 1996.

[11] Kahn, J., Katz, R., and Pister, K. "Mobile networking for smart dust," in *Proc. Mobicom '99*, Aug. 1999, pp. 271-278.

[12] Katz, R. and Brewer, E. "The Case for Wireless Overlay Networks," in Proc. of *SPIE Multimedia and Networking Conference*, San Jose, CA, January 1996.

[13] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. "Optimization by simulated annealing," *Science*, 220(4598):671-680, May 1983.

[14] Ma, Q. and Steenkiste, P. "On path selection for traffic with bandwidth guarantees," *Fifth IEEE International Conference on Network Protocols*, Atlanta, Oct. 1997.

[15] Mead, C. and Conway, L. *Introduction to VLSI systems*, Reading, MA: Addison-Wesley, 1980.

[16] Ramanathan, R. and Steenstrup, M. "Hierarchically-organized, multihop mobile wireless networks for quality-of-service support," Mobile Networks and Applications 3 (1998) 101-119.

[17] Shaikh, A., Rexford, J., and Shin, K. G. "Evaluating the overheads of source-directed quality-of-service routing," in *Proc. ICNP '98*, Oct. 1998.

[18] Shepard, T. J. "A channel access scheme for large dense packet radio networks," in *Proc. of the SIGCOMM '96 Conference on Communications Architectures, Protocols and Applications*, Aug. 1996.

[19] Sinha, P., Sivakumar, R., and Bharghavan, V. "CEDAR: a core-extraction distributed ad hoc routing algorithm," *IEEE JSAC*, 17(8), Aug. 1999.

[20] Wang, Z. and Crowcroft, J. "Quality of service routing for supporting multimedia applications," *IEEE JSAC*, 14(7), Sept. 1996.

[21] Zonoozi, M., Dassanayake, P., and Faulkner, M. "Optimum Hysteresis, Signal Averaging Time and Handover Delay," IEEE Vehicular Technology Conference, March 1997, pp 310-313.