

# Efficient and Resilient Backbones for Multihop Wireless Networks

Seungjoon Lee Bobby Bhattacharjee Aravind Srinivasan Samir Khuller

Department of Computer Science  
University of Maryland, College Park, MD 20742 USA  
{slee, bobby, srin, samir}@cs.umd.edu

Technical Report CS-TR 4726, University of Maryland, College Park

**Abstract**— We consider the problem of finding “backbones” in mobile ad-hoc networks. The backbone provides end-to-end connectivity, allowing non-backbone nodes to save energy since they do not have to route non-local data or participate in the routing protocol. Ideally, such a backbone would be small, consist primarily of high capacity nodes, and remain connected even when nodes are mobile or fail. Unfortunately, it is often infeasible to construct a backbone that has all of these properties, e.g., a small optimal backbone is often too sparse to handle node failures or high mobility.

We present a parameterized backbone construction algorithm that permits explicit tradeoffs between backbone size, resilience to node movement and failure, energy consumption, and path lengths. We prove that our scheme can construct essentially best possible backbones (with respect to energy consumption and backbone size) when the network is relatively static. We generalize our scheme to build more robust structures better suited to networks with higher mobility. We present a distributed protocol based upon our algorithm and show that this protocol builds and maintains a connected backbone in dynamic networks. Finally, we present detailed packet-level simulations to evaluate our protocol and compare our scheme with existing energy-saving techniques. Our results show that, depending on the network environment, our scheme increases network lifetimes by 20–220% without adversely affecting delivery ratio or end-to-end latency.

## I. INTRODUCTION

Ad-hoc networks are characterized by a lack of fixed routing infrastructure; in these networks, end-nodes are often responsible for relaying traffic. However, in many cases, it is possible to identify a “connected dominating set” of nodes that can form a routing backbone [1], [2]. Nodes not in the backbone have at least one backbone neighbor (hence the backbone is a dominating set). Further, since the non-backbone nodes do not need to route traffic for other nodes, they can save energy by not participating in the routing protocol. If the hardware permits, the non-backbone nodes can further save energy by entering a power-save or sleep mode, only waking up to send messages or check for pending messages. Smaller backbones lead to greater overall energy savings; as such, there have been many algorithms developed for constructing near-optimal connected dominating sets in wireless networks [2], [3], [4], [5].<sup>1</sup> The problem of focusing only on the backbone

size is that when nodes are battery-powered, the use of low-battery nodes in the backbone can shorten the overall network lifetime. Therefore, many schemes have been proposed that consider the residual battery power in selecting backbone nodes [6], [7], [8]. However, along with the battery capacity, these schemes often also use other criteria for including nodes in the backbone (e.g., randomized node selection for arbitration [6]). This leads to the inclusion of low-capacity nodes in the resulting backbone. Clearly, a small backbone composed of high-capacity nodes can significantly increase total network lifetime; the construction of such backbones is the subject of this paper.

The operating environments for ad-hoc networks can vary widely (e.g., minimal node mobility in sensor or rooftop networks [9] vs. higher mobility for rescue operation). Ideally, a backbone construction algorithm should work well in a wide range of network environments. In some existing backbone construction algorithms, nodes use only local information to build and maintain a backbone quickly [6], [7], [8]. Although this class of backbone algorithms can be useful in dynamic networks, they do not provide any guarantee on performance objectives such as backbone size or node capacity. Other backbone construction schemes find a “good” connected backbone, e.g., provable bounds on backbone size or control overhead. However, this second class of algorithms typically have higher control overhead, require longer convergence times, and do not provide efficient mechanisms for backbone maintenance [2], [3]. Therefore, they are most useful in static environments, but in mobile networks with dynamic topology, the overhead of maintaining a “good” backbone can be prohibitive. Due to such inherent heterogeneities in the operating environments for ad-hoc networks, it is unlikely that a single fixed algorithm will work best in all situations. In this work, we develop a general solution that can be tailored to particular network environments.

The contributions of this paper are as follows:

- We present a *parameterized backbone construction algorithm*, which permits explicit tradeoffs between different performance measures including backbone size, resilience to node movement and failure, node capacity, and path lengths. Our scheme has two logical steps. First, each node nominates its highest-capacity neighbor

<sup>1</sup>In general, finding the optimal connected dominating set (CDS) is NP-hard.

as its “leader.” Next, we connect these leaders such that the resulting backbone achieves specific efficiency and resilience properties.

- We prove that our scheme can construct essentially *best possible backbones* with respect to node capacity and backbone size. To the best of our knowledge, this is the first work that achieves both objectives at the same time.
- Based upon our backbone construction algorithm, we present a *distributed protocol* that builds and maintains a connected backbone in dynamic network environments.
- We present *simulation results* that investigate different performance aspects of our proposed algorithm, including backbone size, network lifetime, backbone-node capacities, and path lengths. Compared to previous energy-saving techniques, our scheme increases network lifetimes by 20–220% without adversely affecting data delivery or end-to-end latency.

The rest of this paper is organized as follows. We present the first phase of our algorithm (leader nomination) in Section II and the second phase of the algorithm (leader connection) in Section III. We present our distributed protocol in Section IV and simulation results in Section V. We compare our scheme with related work in Section VI conclude in Section VII.

## II. LEADER NOMINATION

In this section, we describe the initial phase of our algorithm. We first describe how each node nominates a *leader*. Then, we show a number of desirable properties of the resulting set of leaders. We defer the description of connecting the set of leaders to Section III.

We model the network as undirected graph  $G = (V, E)$ , where  $V$  is the set of nodes, and  $E$  is the set of edges between nodes. (We discuss the issue of uni-directional links in Section IV.) We denote the total number of nodes in the network by  $n = |V|$ . We define  $N(v)$  to be the set of neighbors of node  $v$ , and  $N^+(v) = N(v) \cup \{v\}$ . We denote the degree of node  $v$  by  $d_v = |N(v)|$ . In this paper, we assume that the network is connected.  $\Delta$  denotes the maximum degree in the network. A node  $v$  is assigned a unique ID and a capacity value  $c_v$ . Although we can consider various attributes for  $c_v$  (e.g., CPU speed, storage space), we focus on the battery capacity in this paper.<sup>2</sup>

### A. Algorithm Description

We assume that each node knows the capacity value of its neighbors. The algorithm, which is similar to a scheme by Gerla and Tsai [10], proceeds as follows: each node nominates the node with highest capacity value in  $N^+(v)$  as leader. Each node then informs its leader of its decision, and all nominated nodes constitute the set of leaders, which we denote by  $\mathcal{L}$ .

For example, in Figure 1(a), the network has nine nodes. The number in each circle denotes the node capacity (e.g., the capacity of node  $A$  is 6). Thin lines between nodes represent wireless links, while thick lines with arrows represent leader

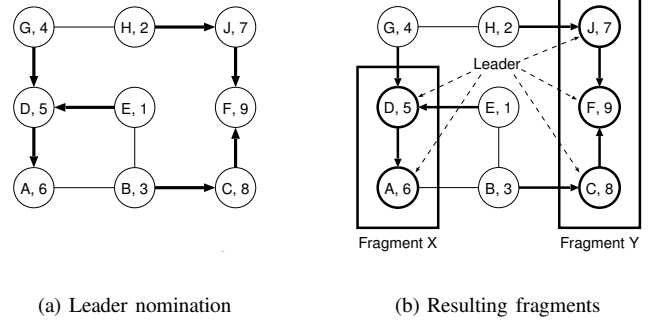


Fig. 1. Leader nomination and resulting fragments.

nomination. In this example,  $G$  nominates  $D$  because  $D$  has a higher capacity (5) than  $H$  (2) and itself (4). As a result, five nodes  $A, C, D, F$ , and  $J$  are nominated as leaders, as shown in Figure 1(b). (Nodes  $A$  and  $F$  nominate themselves as leader, which we do not show in the figure.)

The leader nomination algorithm requires only one-hop neighborhood information and constant time. A similar clustering scheme is proposed in [10]. Gao et al. [11] analyze the size of resulting set using specific geometrical properties. However, their analysis assumes that all nodes have a square-shaped communication region of the same size, which is seldom the case in practice [9]. We next present new analysis results, based on more realistic assumptions.

### B. Properties of the Leader Set

We show that (1)  $\mathcal{L}$  forms a dominating set using high-capacity nodes, and (2) the cardinality of  $\mathcal{L}$  is small under reasonable assumptions. Recall that a *dominating set*  $DS$  of  $G = (V, E)$  is a subset of  $V$ , where each node in  $V$  either is in  $DS$  or has a neighbor in  $DS$  [5]. If all nodes in a dominating set are connected, then it is called a *connected dominating set*. A *minimum* (connected) dominating set is of smallest cardinality among all (connected) dominating set. We define a *maximum-capacity* (connected) dominating set  $DS_M$  to be a (connected) dominating set that maximizes the *bottleneck* node capacity in the set. Formally,  $DS_M$  satisfies:

$$\forall DS, \min_{v \in DS_M} c_v \geq \min_{u \in DS} c_u, \quad (1)$$

where  $DS$  denotes a (connected) dominating set.

**Theorem 2.1:** The set of leaders  $\mathcal{L}$  is a maximum-capacity dominating set.

*Proof:*  $\mathcal{L}$  is a dominating set by construction. We prove that  $\mathcal{L}$  is a maximum-capacity dominating set by contradiction. Assume that  $\mathcal{L}$  is not a maximum-capacity dominating set. Consider a maximum-capacity dominating set  $DS_M$ . Then, the minimum-capacity node  $v \in \mathcal{L}$  satisfies the following:  $\forall u \in DS_M, c_v < c_u$ . By the leader nomination rule, there exists a node  $w$  for which  $v$  is the maximum-capacity node in  $N^+(w)$ . However,  $DS_M$  also has a node in  $N^+(w)$ , which is contradiction. ■

<sup>2</sup>For the ease of exposition, we assume distinct capacity values throughout this paper. In practice, we use unique IDs to break ties.

We now present analysis results showing that the expected size of  $\mathcal{L}$  (denoted by  $E[|\mathcal{L}|]$ ) is small. For the sake of simpler analysis, we first consider the case of  $D$ -regular graphs (i.e.,  $\forall v, d_v = D$ ). Based on the result of this simpler case, we present more generalized analysis results and discuss the analysis results later in this section. In this analysis, we assume  $c_v$  is uniformly distributed between 0 and 1, and  $\log(\cdot)$  denotes the natural logarithm.<sup>3</sup> (The proof of Theorem 2.2 is in Appendix A.)

*Theorem 2.2:* Suppose  $\forall v, d_v = D$  for a positive integer  $D$ . Then, there exists a constant  $\epsilon > 0$  such that:

$$E[|\mathcal{L}|] \leq (1 + \epsilon) \frac{n}{D} \log(D + 1). \quad (2)$$

Also,  $\epsilon$  approaches 0 as  $D$  increases.

In practice, wireless nodes are likely to have different numbers of neighbors, and Theorem 2.2 does not hold in general. However, due to spatial locality in the distribution of nodes, we expect that neighboring nodes in wireless ad-hoc networks have a similar number of neighbors. Formally, for a constant  $\alpha \geq 1$ , we define  $G = (V, E)$  to be  $\alpha$ -locally-regular if  $\forall (u, v) \in E, d_v \leq \alpha d_u$ . For example, if  $G$  is 3-locally-regular, the degree of any of  $v$ 's neighbors is at least  $d_v/3$  and at most  $3d_v$ .

We now generalize the result in Theorem 2.2 to show that in  $\alpha$ -locally-regular graphs,  $E[|\mathcal{L}|]$  is within an  $O(\log \Delta)$ -factor of the size of a minimum dominating set. (The proofs are in the appendix.)

*Lemma 2.3:* Suppose  $G = (V, E)$  is  $\alpha$ -locally-regular for constant  $\alpha \geq 1$ . Then,

$$E[|\mathcal{L}|] \leq c' \sum_{v \in V} \frac{1}{d_v} \log(d_v + 1),$$

where  $c'$  is a constant that depends on  $\alpha$ .

*Theorem 2.4:* Suppose  $G = (V, E)$  is  $\alpha$ -locally-regular for constant  $\alpha \geq 1$ . Then,

$$E[|\mathcal{L}|] = O(\log \Delta) OPT,$$

where  $OPT$  is the size of a minimum dominating set.

*Discussion:* Note that Theorems 2.2 and 2.4 are essentially best possible. Theorem 2.2 holds for any  $D$ -regular graph, and as shown in [12], there exist  $D$ -regular graphs whose minimum dominating sets are of size at least  $(1 - \epsilon') \frac{n}{D} \log D$  for  $\epsilon' > 0$ . As  $D$  becomes large, this value becomes arbitrarily close to the upper bound in Theorem 2.2. On the other hand, in  $D$ -regular graphs, the size of any dominating set should be at least  $n/(D + 1)$ , and we can easily see that as  $D$  grows large, the approximation ratio of our one-step algorithm to the lower bound becomes close to  $\log D$  (in expectation).

In general, finding a minimum dominating set for a given graph is NP-hard [5]. Furthermore, no polynomial time algorithm can achieve the approximation ratio of  $(1 - \epsilon') \log \Delta$

<sup>3</sup>Our current analysis assumes a uniform distribution of node capacity, and we plan to examine other distributions in the future.

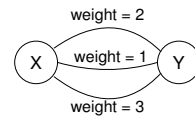


Fig. 2. Multigraph representation of Figure 1(b).

for any  $\epsilon' > 0$  unless NP has  $n^{O(\log \log n)}$ -time deterministic algorithms [13]. Thus, the bound in Theorem 2.4 is within a constant factor of best possible approximation.

We next describe how we connect these leaders to find a connected backbone.

### III. CONNECTING THE LEADERS

In this section, we present the second phase of our algorithm that connects the leaders to construct a connected dominating set. We first describe how to represent the leader set using a multigraph. Then, we describe our proposed algorithm based on this representation.

#### A. Multigraph Representation

The set of leaders form a forest in which edges are leader nomination relations. We refer to each tree in this forest as a *fragment*. For example, in Figure 1(b), there are two fragments: fragment  $X$  (nodes  $A$  and  $D$ ) and fragment  $Y$  (nodes  $C, F$ , and  $J$ ). Since  $\mathcal{L}$  is a dominating set, as shown in [5], chains of up to two non-leader nodes are sufficient to connect all fragments. We define a *virtual edge* to be such a chain of (up to two) non-leader nodes that connects two fragments. We transform the graph into a multigraph, where each fragment corresponds to a vertex with (possibly multiple) virtual edges connecting fragments. For a given virtual edge, we use the minimum node capacity as the weight of the edge.<sup>4</sup> In Figure 1(b), there are three virtual edges between fragments  $X$  and  $Y$ . The first one connects the fragments using nodes  $G$  and  $H$ , the second one uses nodes  $E$  and  $B$ , and the last one uses node  $B$  only. Since we use the minimum node capacity as virtual edge weight, the weights of these three edges are 2, 1, and 3, respectively. Figure 2 shows the corresponding multigraph representation. In the rest of this section, we describe how we use this multigraph to find a connected backbone.

#### B. Spanning Tree-Based Algorithm

We begin with an approach based on the well-studied minimum spanning tree (MST) problem. This MST-based approach is a special case of our parameterized algorithm (Section III-C). Recall that an MST of edge-weighted graph  $G = (V, E)$  connects all nodes in  $V$  using a tree  $T \subseteq E$ , such that the sum of edge weights in  $T$  is minimized [14].

In many algorithms that find MSTs, nodes select a minimum outgoing edge that does not result in a loop [14], [15].

<sup>4</sup>It is possible that no nodes are involved in a virtual edge. In this case, we set the weight of the virtual edge to  $\infty$ .

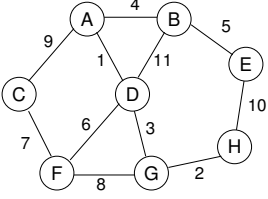


Fig. 3. Example graph.

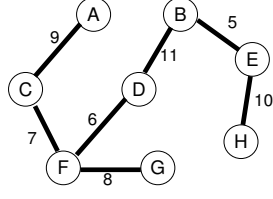
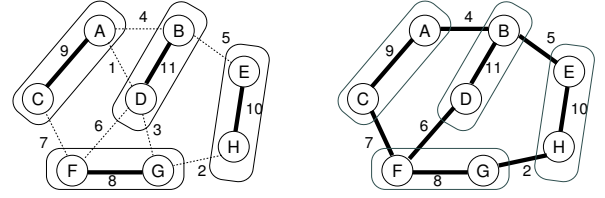


Fig. 4. MST-based connected backbone



(a) Backbone construction after first round

(b) Resulting backbone ( $\mathcal{B}_1$ )

Fig. 5. Illustration of truncated algorithm.

However, since we want to select high-capacity nodes in the backbone, we need to use *maximum-weight* outgoing virtual edges. For example, in Figure 2, when connecting fragments  $X$  and  $Y$  to obtain high-capacity connected backbone, we should use the virtual edge of weight 3. We further illustrate this approach using an example graph in Figure 3. In this figure, each node corresponds to a fragment after the leader nomination phase, and each fragment is connected by virtual edges. (We show only the maximum-weight virtual edges between fragments for clarity.) Figure 4 shows the MST (as we have defined it above).

Let  $\mathcal{B}_{MST}$  denote the connected backbone obtained by using an MST algorithm. We next show that  $\mathcal{B}_{MST}$  produces a *small* connected backbone using *high-capacity* nodes.

**Theorem 3.1:**  $\mathcal{B}_{MST}$  is a maximum-capacity connected dominating set.

*Proof:* Please see Appendix D. ■

**Lemma 3.2:**  $|\mathcal{B}_{MST}| \leq 3|\mathcal{L}|$ , where  $\mathcal{L}$  denotes the leader set.

*Proof:* Suppose that  $\mathcal{L}$  initially consists of  $f$  fragments. We need to use  $(f - 1)$  virtual edges to find a spanning tree. Since there are at most two nodes in each virtual edge and  $f \leq |\mathcal{L}|$ ,  $|\mathcal{B}_{MST}| \leq |\mathcal{L}| + 2(f - 1) \leq 3|\mathcal{L}|$ . ■

**Theorem 3.3:** For  $\alpha$ -locally-regular graphs,

$$E[|\mathcal{B}_{MST}|] = O(\log \Delta) OPT, \quad (3)$$

where  $OPT$  denotes the size of minimum connected dominating set.

*Proof:* This follows from Theorem 2.4 and Lemma 3.2. ■

**Discussion:** Theorem 3.1 states that  $\mathcal{B}_{MST}$  includes a node with low capacity only when it is necessary in maintaining connectivity. We show in Theorem 3.3 that for  $\alpha$ -locally-regular graphs, the size of  $\mathcal{B}_{MST}$  is an  $O(\log \Delta)$ -approximation to a minimum connected dominating set. As discussed in Section II, this is within a constant factor of best possible approximation. However, if a maximum-capacity backbone of even smaller size is desired, we can further reduce the constant factor by using the following optimization. When the  $\mathcal{B}_{MST}$  is found, we get to know the minimum node capacity in the backbone (say,  $c_{th}$ ). We flood this capacity value in the network, and node  $v$  becomes eligible to be in a new backbone

---

**Algorithm 1** Description of Truncated Algorithm (Centralized)

---

- 1: Round  $\leftarrow$  0
  - 2: **while** more than one fragment exists **do**
  - 3:   **if** Round =  $K$  **then**
  - 4:     Merge with all neighboring fragments
  - 5:     Return
  - 6:   **end if**
  - 7:   Each fragment selects the best outgoing edge
  - 8:   Merge fragments using the selected edges
  - 9:   Round  $\leftarrow$  Round + 1
  - 10: **end while**
- 

if and only if  $c_v \geq c_{th}$ . Then, we apply onto the restricted set of eligible nodes any distributed CDS algorithm [2], [3]. For example, by applying the scheme in [2], the approximation ratio of the resulting maximum-capacity backbone becomes at most  $2 \log \Delta + 3$ . (See Appendix E.)

Although an MST-based approach achieves our desired goals (i.e., finding a small backbone using high-capacity nodes), the running time can be long. For example, a distributed MST algorithm by Gallager, Humblet, and Spira (the GHS algorithm) takes  $O(n \log n)$  running time [15]. Other potential problems with tree structures include minimum redundancy and increased path lengths. In Figure 4, the resulting backbone becomes disconnected even when a single link fails. Also, to reach a node in fragment  $G$ , a node in fragment  $H$  needs to use a path consisting of five virtual edges, compared to only one when no backbone is used. There is a clear trade-off between small backbones and shorter path lengths as well as resilience. We address this issue next.

### C. TRUNC-K: Our Parameterized Algorithm

We now describe our generalized scheme that balances the above-mentioned trade-off when connecting the leader set (Algorithm 1). It is based on a well-known MST algorithm by Boruvka [16]. In Boruvka's algorithm, each fragment finds and marks the best outgoing edge. Then, using such marked edges, fragments are merged into new larger fragments. This step is repeated until there is no outgoing edge (i.e., there is only one fragment). During the first  $K$  rounds, our algorithm runs just as Boruvka's algorithm does, where  $K$  is a parameter to the algorithm. However, in our truncated algorithm, all remaining

	1.4 km × 1.4 km		2.0 km × 2.0 km		2.8 km × 2.8 km		4.0 km × 4.0 km	
	min.	avg.	min.	avg.	min.	avg.	min.	avg.
$\mathcal{B}_0$	0.349	0.913	0.111	0.854	0.021	0.771	0.007	0.666
$\mathcal{B}_1$	<b>0.860</b>	0.967	<b>0.705</b>	0.934	<b>0.307</b>	0.872	<b>0.054</b>	0.760
$\mathcal{B}_2$	0.888	0.970	0.787	0.942	0.573	0.892	0.188	0.796
$\mathcal{B}_{MST}$	<b>0.888</b>	0.970	<b>0.787</b>	0.942	<b>0.574</b>	0.893	<b>0.200</b>	0.802

TABLE I

CAPACITY VALUES OF BACKBONE NODES WITH VARYING NODE DENSITY (1000 NODES).

fragments after  $K$  rounds mark edges to *all* neighboring fragments and are merged into one fragment. One extreme case is  $K = 0$ , where after leader nomination, each pair of neighboring fragments marks one virtual edge (e.g., all edges shown in Figure 3). Another extreme case is when  $K = \infty$ , which results in  $\mathcal{B}_{MST}$ .

Figure 5 shows the operations of the algorithm applied to the graph in Figure 3. Here, we set  $K = 1$ . In the first round, each individual fragment selects the best outgoing edge among neighboring fragments, and fragments are merged using selected edges. Then, as shown in Figure 5(a), there remain four fragments at the end of first round. Since  $K = 1$  in this example, each remaining fragment after the first round connects to all neighboring fragments. For example, fragment FG chooses three edges to fragments AC, BD, and EH. The resulting connected backbone is shown in Figure 5(b).

We call this algorithm TRUNC- $K$  and the resulting backbone  $\mathcal{B}_K$ . In contrast to  $O(\log n)$  rounds in Boruvka’s algorithm, TRUNC- $K$  needs only a constant number ( $K$ ) of rounds to complete, and the resulting backbone has higher redundancy than  $\mathcal{B}_{MST}$ . This eventually leads to both increased resilience against node mobility and decreased average path length. However, these properties can potentially degrade the backbone quality in two aspects. First, in general, the resulting backbone includes more virtual edges than  $\mathcal{B}_{MST}$ , and the result of Theorem 3.3 does not hold. However, we can adjust  $K$  to control the amount of increase, and our simulation results in Section III-D show that even with small values of  $K$ , the increase in backbone size is not significant. Also, the resulting backbone is not a maximum-capacity backbone and may include low-capacity nodes. However, by construction, nodes included in the first  $K$  rounds are part of a maximum-capacity backbone. After the  $K$ -th round, since we choose best virtual edges connecting to remaining neighboring fragments, we are likely to include relatively high-capacity nodes. We further investigate these aspects through simulation experiments next.

#### D. Evaluation of the TRUNC- $K$ Algorithm

In this subsection, we use simulations to understand the TRUNC- $K$  algorithm for different values of the parameter  $K$ . We vary the value of  $K$  and investigate tradeoffs such as the backbone size, capacity distribution of backbone nodes, and average path length. In this simulation, stationary nodes are distributed on a square uniformly at random, and we vary the number of nodes and the size of square to experiment with various settings. Node capacity values are uniformly distributed between 0 and 1. (We also experimented using

	Area	
	2km × 2km	4km × 4km
No backbone	5.04	11.17
$\mathcal{B}_0$	6.14	12.52
$\mathcal{B}_1$	7.85	14.76
$\mathcal{B}_2$	9.36	18.75
$\mathcal{B}_{MST}$	9.36	23.03

TABLE II

AVERAGE PATH LENGTH (1000 NODES).

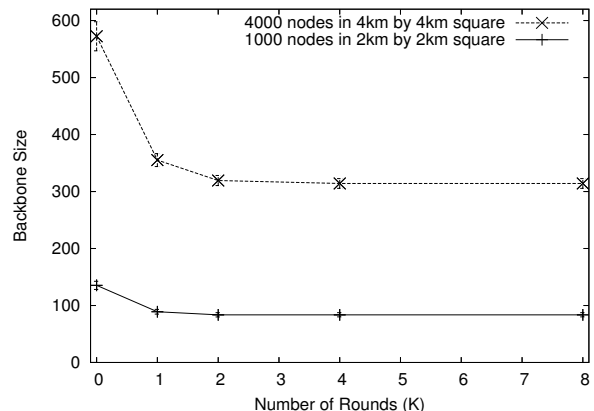


Fig. 6. The size of the backbone with different  $K$  values. The error bars represent standard deviations.

various scenarios with non-uniform node placement and different capacity distribution, and the results were similar.) Nodes within the nominal transmission range (250 meters) become neighbors. For each set of parameters, we use 25 runs with different node placement scenarios and report the average.

*Backbone Size:* In Figure 6, we show the average size of the backbone with varying  $K$ . We use two different network settings—the one with 1000 nodes on a 2km × 2km square, and the other with 4000 nodes on a 4km × 4km square. In Figure 6, the use of extra rounds is most effective when  $K$  is small. Specifically, the first round ( $K = 1$ ) leads to the largest reduction in backbone size. With larger  $K > 2$ , there are a small number of remaining fragments after  $K$  rounds. As a result, when compared to  $\mathcal{B}_{MST}$ , connecting all neighboring fragments does not significantly increase the backbone size.

*Capacity Distribution:* We next investigate another potential problem with TRUNC- $K$  backbone—the resulting backbone may include low-capacity nodes. In these experiments, we use 1000 nodes but vary the size of square, thus varying node density.

In Table I, we list the minimum and average capacity values depending on the  $K$  value with different node density. Even with small  $K$  (1 or 2), the difference in minimum-capacity between  $\mathcal{B}_K$  and  $\mathcal{B}_{MST}$  is small. For example, in the case of 2km × 2km square, the difference in the minimum capacity between  $\mathcal{B}_1$  and  $\mathcal{B}_{MST}$  is around 10% (0.705 vs. 0.787). The difference in average capacity values is even smaller: less than 1% for the same scenario. The reason is as follows: After  $K$  rounds, remaining fragments are relatively large and typically have multiple virtual edges to each neighboring fragment.

Since a fragment always chooses best possible virtual edges to connect neighboring fragments, very low-capacity nodes are not likely to join the backbone. However, in sparser networks, fewer virtual edges are available between neighboring fragments, and the difference in minimum capacity between  $\mathcal{B}_K$  and  $\mathcal{B}_{MST}$  is slightly larger.<sup>5</sup> In the actual deployment of the TRUNC-K algorithm, the value of the parameter  $K$  should be chosen appropriately using the knowledge of expected node density and mobility.

*Average Path Length:* We examine the average path lengths induced by the resulting backbone. We consider all possible node pairs and calculate the shortest path length between each pair. In Table II, we list the average path lengths of all node pairs for experiments with 1000 nodes in different deployment areas. The use of any routing backbone inevitably increases path length since we are forced to find paths using a restricted set of nodes.  $\mathcal{B}_0$  has most redundancy, and the increase in path length is minimum. As  $K$  increases, the path length also increases, and such increase is more severe for networks with larger diameters. Specifically, in the  $4\text{km} \times 4\text{km}$  case, the average expansion in path lengths using the  $\mathcal{B}_{MST}$  is more than twice the underlying shortest paths. Small  $K$  values again offer a good trade-off. For example, compared to  $\mathcal{B}_{MST}$ , the average path length of  $\mathcal{B}_1$  is up to 36% shorter, while the backbone size is only up to 13% larger.

To summarize, backbones obtained using small  $K$  (1 or 2) perform well and provide a reasonable balance among a number of performance measures. We next describe a distributed protocol that implements the TRUNC-K algorithm.

#### IV. DISTRIBUTED PROTOCOL DESCRIPTION

In this section, we present our distributed protocol that implements the TRUNC-K algorithm to construct and maintain a connected backbone in dynamic network environments. Our protocol is based on the GHS algorithm, which is a distributed version of Boruvka’s algorithm [15]. We assume that each node has a unique ID (e.g., IP address). We first describe the leader nomination and explain how to connect the fragments obtained after the nomination phase. We also present a backbone maintenance mechanism later in this section.

##### A. Leader Nomination Protocol

Each node broadcasts a HELLO message periodically that includes information about itself and its neighbors. Table III shows the fields for individual node information in HELLO messages. Using these fields, each node maintains information about two-hop neighbors (e.g., capacity, fragment root IDs).

Before broadcasting a HELLO message, node  $v$  checks which neighbor has the highest capacity. Suppose  $u$  is the highest-capacity neighbor of  $v$ . Then,  $v$  sets its *Leader ID* field to  $u$  in its HELLO message. Upon receiving a HELLO message from  $v$ ,  $u$  becomes a leader and sets its *IsLeader* field to TRUE in subsequent HELLO messages until  $v$  changes leaders and there exist no other neighbors nominating  $u$ .

<sup>5</sup>When  $5\text{km} \times 5\text{km}$  squares are used with 1000 nodes, only four cases out of 25 resulted in connected networks, and the use of  $4\text{km} \times 4\text{km}$  squares with 1000 nodes corresponds to considerably sparse scenarios.

Field Name	Description
<i>Node ID</i>	node identifier
<i>Capacity</i>	capacity of node
<i>IsLeader</i>	whether this node is a leader or not
<i>Leader ID</i>	highest-capacity neighbor
<i>Level-0 fragment root</i>	ID of level-0 fragment root
...	...
<i>Level-K fragment root</i>	ID of level- $K$ fragment root

TABLE III

INFORMATION ABOUT INDIVIDUAL NODES IN A HELLO MESSAGE.

If node  $u$  finds itself as the highest-capacity node in its neighborhood,  $u$  immediately becomes a leader. In this case,  $u$  also becomes a *level-0 fragment root*, where a level-0 fragment is a set of leaders who are themselves connected via the leader-nomination relation. In Figure 1(b),  $A$  and  $F$  are level-0 fragment roots.

##### B. Protocol for Fragment Members

As discussed in Section III, the set of leaders form a forest consisting of multiple fragments, and the protocol described here merges the fragments to form one connected component. In Algorithm 2, we present a high-level protocol description. We begin with the operation of level-0 fragments and later discuss the operation of higher-level fragments.

We illustrate the protocol operations of level-0 fragments using Figure 7. Each level-0 fragment forms a tree rooted at its fragment root. To discover neighboring level-0 fragments, level-0 fragment roots periodically send  $\text{REQ}_0$  messages, which are forwarded down this tree to the leaves (who cannot forward the REQ message any further). The leaves then generate a  $\text{REPLY}_0$  message that contains information about other fragments (if any) that they are connected to. The  $\text{REPLY}_0$  messages are forwarded back towards the fragment root. For example, in Figure 7(b), node  $D$  generates a  $\text{REPLY}_0$  message, which contains the ID of other level-0 fragments that  $D$  knows of ( $F_y$  in this example) along with the cost of the virtual edge to connect to  $F_y$ . (Recall that  $D$  keeps the information about fragment roots of two-hop neighbors.) At each hop, before forwarding the REPLY message towards the leader, nodes update the message if they know of a better virtual edge than the one carried in the message. Also, nodes add information about any new neighboring fragments that are not in the REPLY message. In our example, node  $B$  does not modify the REPLY message from  $D$ , since its path to  $F_y$  is worse than the one that  $D$  found (Figure 7(b)). ( $A$  and  $C$  also send  $\text{REPLY}_0$  messages, which we have not shown in the figure.)

Once the fragment root has accumulated all REPLY messages (or has timed out on some), it sends a CONNECT message using the best outgoing virtual edge. This is shown in Figure 7(c), where  $X$  connects to  $Y$  using the weight 7 edge through  $D$ . This virtual edge has two non-backbone nodes  $P$  and  $Q$ , and upon receiving the  $\text{CONNECT}_0$  message, they become *bridges* and join the backbone. Using these bridge nodes,  $F_x$  and  $F_y$  are merged to form a new level-1 fragment.

Level-1 fragments also need to find neighboring level-1

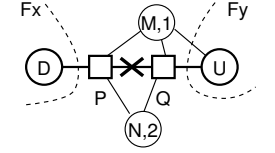
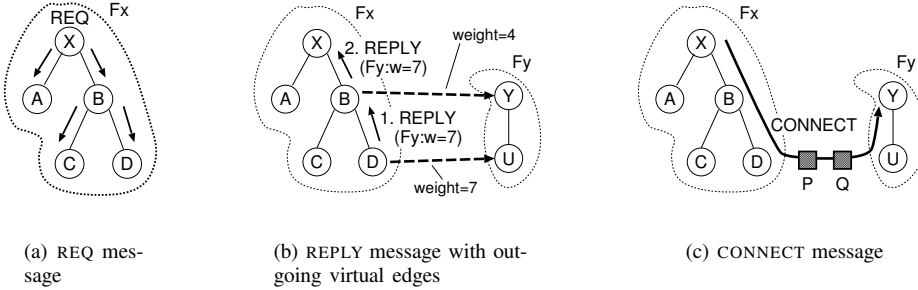


Fig. 8. Local maintenance.

Fig. 7. Overview of protocol operations.

---

**Algorithm 2** Distributed operation of level- $i$  fragments
 

---

- 1: Level- $i$  fragment root periodically sends  $REQ_i$  message
  - 2: Fragment members send  $REPLY_i$  messages with neighboring level- $i$  fragment information
  - 3: **if** level- $i$  fragment root receives all  $REPLY_i$  messages, or a timeout occurs **then**
  - 4:   **if**  $i = K$  **then** {highest level}
  - 5:     Fragment root sends  $CONNECT_i$  messages to all neighboring level- $i$  fragments
  - 6:   **else** {  $i < K$  }
  - 7:     Fragment root sends  $CONNECT_i$  message only to best neighboring level- $i$  fragment
  - 8:   **end if**
  - 9: **end if**
- 

fragments to form next-level fragments. We need to elect level-1 fragment roots to send  $REQ_1$  messages. We use the following rule similar to the GHS algorithm [15]: If two fragments choose each other as their best neighboring fragment, then two fragment roots become candidates for the next-level fragment root. We choose the node with lower ID as the level-1 fragment root.

Level- $i$  fragments ( $0 < i < K$ ) operate similarly to above procedures until their level reaches  $K$ . At the highest level- $K$ , instead of connecting using only the best virtual edge, the level- $K$  fragment roots send  $CONNECT$  messages to *all* neighboring level- $K$  fragments, thus assuring a connected backbone.

In Figure 7,  $X$  is a both level-0 and level-1 fragment root, and it periodically sends both  $REQ_0$  and  $REQ_1$  messages. In general, a node can be a fragment root of up to  $(K+1)$  levels at the same time. Even if higher-level fragments are already found, lower-level fragment roots (e.g.,  $Y$  in Figure 7) still send  $REQ_i$  messages periodically. This allows lower-level fragments to find new or better virtual edges to neighboring fragments in dynamic networks.

### C. Backbone Maintenance

All of the protocol specific states (e.g., leaders, bridges, fragment roots) are “soft.” A node removes a neighbor if it does not receive a HELLO message from the neighbor for a certain duration (e.g., four HELLO-PERIODS). If a leader finds that no neighbors nominate it as leader for some time (e.g.,

due to mobility or capacity change), it stops being a leader. When a bridge does not receive a  $CONNECT$  message for a certain period of time, it stops being a bridge.

In a dynamic network, however, the basic protocol mechanisms described above may not be sufficient for the timely maintenance of the connected backbone. We efficiently reconstruct the backbone using a simple local search protocol that exploits spatial locality. We illustrate its operation via an example. In Figure 8, node  $P$  detects a link failure to backbone neighbor  $Q$ .  $P$  looks up its neighbor table to find other nodes that also had  $Q$  as neighbor. (Note that these nodes need not currently be part of the backbone). In this example,  $P$  finds two such neighbors,  $M$  and  $N$ , and sends a  $RECOVER$  message to  $N$ , which has higher capacity. Upon receiving this message,  $N$  temporarily joins the backbone and forms a bridge to  $Q$ . In the next  $REQ$ - $REPLY$  phase,  $X$  might choose a different virtual edge (of higher weight) to connect to  $F_y$ . If that happens,  $N$  will leave the backbone after a timeout.

There are potential problems with the local recovery scheme. First, the repaired backbone may include lower-capacity nodes than necessary. However, as mentioned above, in the next  $REQ$ - $REPLY$  phase, the fragment root will discover the best virtual edge and send the appropriate  $CONNECT$  message. Also, a node may not be able to find a common neighbor for recovery. However, in networks with reasonable node density, such events will likely be infrequent. Finally, the recovery scheme does not help when nodes fail. However, the  $TRUNC$ - $K$  backbone should have sufficient resilience to maintain connectivity against infrequent recovery failures. We examine the effectiveness of this recovery scheme using simulations in Section V.

### D. Discussion

*Control Overhead:* As shown in Table III, HELLO messages contain  $(K+3)$  node IDs per neighbor: (*Node ID, Leader ID, Level-0 fragment root, ..., Level- $K$  fragment root*). For example, suppose that node  $U$  in Figure 7(c) is about to broadcast a HELLO message. When  $K=1$ , the information for neighbor  $Q$  should include  $(Q, U, Y, X)$ . Including more information in HELLO messages increases the control overhead, and to minimize the increase, we use a simple indexing scheme as follows.  $U$  arranges all neighbors into an array and uses the index values for leader IDs and fragment roots of its neighbors. In the above example,  $U$  includes the

following information for its neighbor  $Q$ :  $(Q, I_U, I_Y, I_X)$ . Since an index field is typically shorter (e.g., 1 byte) than an actual ID (e.g., 4 bytes), the amount of length increase becomes smaller. Note that  $U$  still needs to include some non-neighbor IDs (e.g.,  $X$ ) in its HELLO messages. However, since many of its neighbors share leaders and fragment roots, the number of such non-neighbors nodes are likely to be small. TRUNC-K also uses a few additional control messages (e.g., REQ, REPLY), which can potentially increase the overall control overhead. Our simulation results in Section V-C show that the overall control overhead of TRUNC-K is minimal.

*Link Quality:* In practice, wireless links show a wide range of difference in their quality [9]. Since links between backbone nodes are likely to be used frequently, it is beneficial to build a backbone connected through high-quality links. In the future, we plan to incorporate the link quality aspect into backbone construction and maintenance mechanisms. Another related issue is uni-directional links (e.g., due to the difference in antenna capability or transmit power). Although our protocol description assumes that wireless links are bidirectional, we can easily detect uni-directional links using the neighbor information in periodic HELLO messages. Existing protocols such as the IEEE 802.11 MAC protocol [17] assume bi-directional links, and if TRUNC-K is used with such protocols, we can exclude uni-directional links when building backbones.

## V. SIMULATION STUDY

In this section, we compare TRUNC-K with prior approaches using simulation experiments. Based on the results in Section III-D, we consider only the case of  $K = 1$ . Although TRUNC-K backbones can be used to save different types of resources (e.g., to avoid cryptographic operations by non-backbone nodes), we focus on saving energy and extending network lifetime. We first describe prior approaches, and then we compare the performance of our scheme against those schemes.

### A. Brief Description of Existing Schemes

In this section, we compare the performance of our algorithm with that of SPAN [6], GAF [7], and the scheme proposed by Wu et al. [8].

In SPAN [6], a node becomes a *coordinator* and joins the backbone when any two neighbors are not connected using up to two current coordinators. To minimize contention and give priority to high-energy nodes, SPAN uses a randomized backoff, in which the energy level, number of neighbors, and a random number determine the backoff time. A coordinator withdraws after some period of time to give other neighbors a chance to become coordinators.

In GAF [7], the area is divided into square-shaped virtual grids. GAF assumes the availability of location information (e.g., from GPS), and each node can know its virtual grid from the location information. Then, through control messages, GAF elects the highest-energy node in each grid, such that such elected nodes stay awake on behalf of other nodes. In GAF, grids are constructed such that a node can reach any

	Number of nodes (square size in km $\times$ km)			
	500 (1.4 $\times$ 1.4)	1000 (2.0 $\times$ 2.0)	2000 (2.8 $\times$ 2.8)	4000 (4.0 $\times$ 4.0)
SPAN	54.5	113.9	227.7	467.4
Wu et al.	67.4	150.3	308.9	652.5
GAF	158.0	308.6	605.6	1236.9
TRUNC-1	<b>44.7</b>	<b>89.0</b>	<b>174.6</b>	<b>355.1</b>

TABLE IV

BACKBONE SIZE CONSTRUCTED BY DIFFERENT SCHEMES.

node in any neighboring grid, and therefore these elected nodes form a connected backbone.

In the scheme by Wu et al. [8], a node initially joins the backbone if its two neighbors are not connected. Then, to reduce the size of this initial backbone, node  $v$  searches for a neighbor  $u$ , or two neighbors  $u$  and  $w$ , such that the (union of) neighbor set(s) includes the neighbor set of  $v$ . If we use only the above rule, due to symmetry among nodes, backbone nodes may lose connectivity. To break symmetry and ensure connectivity, the authors of [8] use the power level and degree of node.

### B. Comparison Study in Large Networks

In this set of experiments, we use the same settings as in Section III-D. We consider capacity values in  $[0, 1]$ ; in GAF, the side length of the square grid is set to 100 meters (which is the value the authors of GAF use when the transmission range is 250 meters [7]). We measure the performance when the initial backbone stabilizes, and report the average of 25 runs each.

We first examine the size of backbones constructed by different schemes. In this set of experiments, we vary the number of nodes and the size of square, but maintain the average node degree constant. We also experimented with different settings such as varying node density and non-uniform node placement, and the results were similar. In Table IV, we present the average backbone sizes for various scenarios. The standard deviations are small (less than 6% of the average in all cases), and we do not present them here. We observe that TRUNC-1 leads to the smallest backbones in all the experiments. Specifically, when the network has 4000 nodes, the TRUNC-1 backbone has 355 nodes on average. This is 24% smaller than the SPAN backbone, which is the second smallest in all these experiments.

Besides the small size, our proposed scheme also builds a backbone consisting of higher-capacity nodes. In Table V, we tabulate the minimum and average capacity values of backbone nodes. In all cases, the backbone by TRUNC-1 achieves the highest values for both minimum and average node capacity. For example, in 4000-node networks, the TRUNC-1 backbone does not include any of bottom 30% nodes, while the GAF backbone includes some of bottom 0.5% nodes. In the same scenario, the average capacity of TRUNC-1 backbone is also 30% higher than those of SPAN and GAF. When the routing backbone is used to reduce power consumption and increase the network lifetime, the use of low-capacity nodes can drain their energy unnecessarily. We later investigate this aspect using packet-level simulations.

	No. of Nodes (square size in km $\times$ km)							
	500 (1.4 $\times$ 1.4)		1000 (2.0 $\times$ 2.0)		2000 (2.8 $\times$ 2.8)		4000 (4.0 $\times$ 4.0)	
	min.	avg.	min.	avg.	min.	avg.	min.	avg.
SPAN	0.056	0.700	0.046	0.686	0.025	0.704	0.011	0.708
Wu et al.	0.005	0.504	0.002	0.480	0.002	0.495	0.002	0.504
GAF	0.032	0.714	0.014	0.707	0.007	0.720	0.003	0.723
TRUNC-1	<b>0.752</b>	<b>0.937</b>	<b>0.705</b>	<b>0.934</b>	<b>0.502</b>	<b>0.933</b>	<b>0.335</b>	<b>0.933</b>

TABLE V

CAPACITY VALUE OF BACKBONE NODES BY EACH SCHEME

	No. of nodes (square size in km $\times$ km)			
	500 (1.4 $\times$ 1.4)	1000 (2.0 $\times$ 2.0)	2000 (2.8 $\times$ 2.8)	4000 (4.0 $\times$ 4.0)
No backbone	3.66	5.04	6.86	9.60
SPAN	4.39	6.17	8.44	11.89
Wu et al.	4.14	5.75	7.86	11.01
GAF	3.93	5.45	7.45	10.46
TRUNC-1	5.66	7.84	10.27	14.35

TABLE VI

AVERAGE PATH LENGTH BY DIFFERENT SCHEMES.

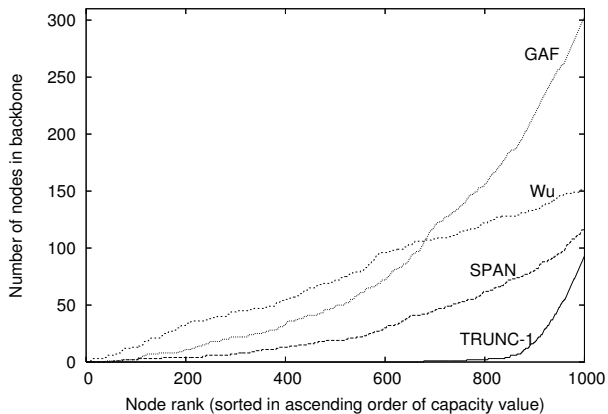


Fig. 9. The capacity distribution of backbone nodes in different schemes.

In Figure 9, we present a detailed snapshot of a representative run with 1000 nodes. In this figure, we sort all nodes in an ascending order of capacity value and cumulatively plot the number of backbone nodes whose capacity is less than or equal to that of a given node. For example, the GAF backbone includes 49 nodes out of 500 lowest-capacity nodes, while SPAN chooses 19 nodes from the lowest 500 nodes. In contrast, the TRUNC-1 backbone does not include any of the lowest-capacity nodes, but selects only 93 nodes among the top 330 nodes.

Finally, in Table VI, we report the average path lengths by different schemes as well as the case using no backbones. Not surprisingly, since TRUNC-1 backbones are smaller in size than any other scheme (Table IV), its average path lengths are the longest. However, the amount of reduction in backbone size is more than the increase in the path length, especially in larger networks. Specifically, in the case of 4000-node networks, the difference in the average path length between SPAN and TRUNC-1 is around 20%, while the difference in the backbone size is more than 30%.<sup>6</sup> The other two schemes (GAF and Wu et al.) have shorter path lengths on average, but their backbones are substantially larger in size (Table IV). This result illustrates that TRUNC-1 backbones provide relatively good paths considering the small size.

<sup>6</sup>In contrast, TRUNC-0 backbones are 22% larger than SPAN backbones, but decrease the average path lengths by 3%.

### C. Packet-level Simulations

In this subsection, we report results from ns-2 simulations [18]. In Section V-B, SPAN and TRUNC-1 performed best, and we compare only these two schemes here. We use the SPAN simulation code provided by the authors of SPAN.<sup>7</sup> We also use the code by Naoumov and Gross [19], which enables faster simulation with large networks. Due to high resource requirements, we have been able to perform simulations only with relatively small topologies (with 150 nodes). We first describe our simulation environment before presenting the results.

1) *Simulation Environment*: In our simulation experiments, both TRUNC-K and SPAN run on top of the IEEE 802.11 MAC layer [17], and non-backbone nodes stay in the power saving mode. In the IEEE 802.11 power saving mode, time is partitioned into *beacon periods*. Each beacon period starts with an ATIM (Ad-hoc Traffic Indication Message) window, during which all nodes should stay awake. If a node has a packet to send, it buffers the packet until it can advertise the packet using ATIM frames in the next ATIM window. When a node finds that there are buffered incoming packets, it sends an ATIM-ACK and stays awake during the beacon period to receive actual messages. Otherwise, it goes back to sleep until the start of the next beacon period. Since nodes cannot send actual data packets during the ATIM window, power saving mode usually leads to increased delay and reduced throughput. Chen et al. [6] slightly modified the power saving mode in the 802.11 MAC to improve performance, which we use in our simulation experiments.

We assume there are three classes for the node energy level. A low-energy node has 300J of energy, which is used in the experiments in [6]. A medium-energy node has 600J, and a high-energy node has 2500J. (2500J is usually sufficient to last 3000 seconds of simulation time.) We vary the node percentage of each class to examine the performance in different settings. We use the following power consumption values reported in [6]: 1.4W for transmission, 1.0W for receiving, 0.83W for idling, and 0.13W for sleeping. We place 150 nodes uniformly at random on a 1000 meter by 1000 meter square area. The transmission range of each mobile node is 250 meters.

We choose 10 pairs of source and destination nodes uniformly at random among high-energy nodes; each source generates CBR (Constant Bit Rate) traffic 50 seconds into the simulation at the rate of one 128-byte packet per second. The MAC-level data transmission rate is 2 Mbps. As we discuss

<sup>7</sup>Available at <http://www.pdos.lcs.mit.edu/span/>. We plan to make our simulation code public as well.

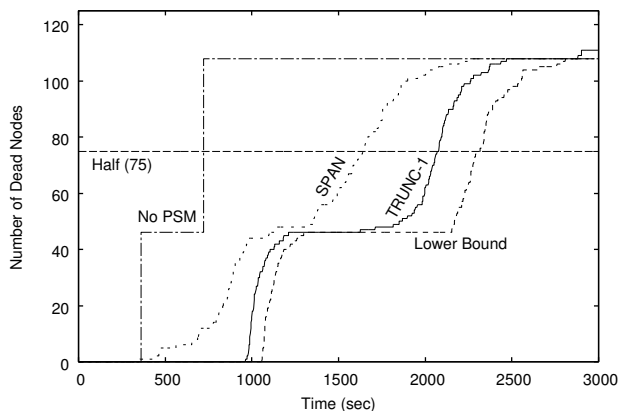


Fig. 10. Number of dead nodes over time. L:M:H=3:4:3.

later, SPAN rotates backbone nodes frequently (e.g., 2 changes per second), which shortens path lifetimes. When we used on-demand routing protocols over SPAN, the path maintenance overhead was high. Instead, we use an idealized scheme for packet routing, where a path is found on top of the connected backbone using the centralized Floyd-Warshall algorithm [14] implemented in ns-2. This corresponds to a best case scenario for SPAN. Nodes move according to the Random Waypoint mobility model (pause time=400s, and maximum speed is 1–16m/s) [18]. We also set the minimum speed to be 0.1m/s to avoid the problem of speed decay [20]. Unless otherwise stated, we use mobile scenarios with the maximum speed of 1m/s.

In the simulations of both TRUNC-1 and SPAN, each node sends a HELLO message every two seconds. For TRUNC-1, we set the period of REQ messages to 14 seconds, which leads to reasonable performance. In each case, we report the average of 5 runs.

2) *Simulation Results:* For the first set of results, we examine two types of network lifetimes [21]. Network *1-life* is the time when the first node dies, and *half-life* is the time when the half of initial nodes die.<sup>8</sup> In addition to TRUNC-1 and SPAN, we use two other schemes for reference. The first one is to identify a lower bound, in which all nodes always stay in sleep mode except when they wake up for ATIM windows. Each node also sends a 128-byte HELLO message every two seconds. In the second scheme (No-PSM), no power saving mode operation is used, and all nodes always stay awake without sending any control messages. We send no data traffic in either of the two reference cases.

In Figure 10 we present a snapshot for the number of dead nodes over time. In this setting, approximately 30% of nodes are low-energy (L), 40% of them are medium-energy (M), and the rest 30% are high-energy (H) nodes. (To denote this ratio, we use an abbreviated notation L:M:H=3:4:3.) In Figure 10, the network 1-life of SPAN is similar to that of “No-PSM.” This is expected from Figure 9 to some extent; SPAN

<sup>8</sup>We assume that the network needs external support after this time (e.g., addition of fresh nodes in the case of sensor networks).

Ratio of L:M:H	1-Life (sec)		Half-Life (sec)	
	TRUNC-1	SPAN	TRUNC-1	SPAN
4:4:2	892.1 (101.0)	365.5 (28.3)	1911.0 (139.6)	1506.3 (54.5)
3:4:3	946.6 (22.0)	375.0 (29.1)	2106.2 (33.7)	1689.8 (40.5)
2:4:4	962.0 (13.7)	412.3 (54.7)	2208.3 (41.1)	1842.3 (43.9)

TABLE VII

NETWORK LIFETIMES WHEN THE PROPORTION OF NODES AT DIFFERENT ENERGY LEVELS IS VARIED. THE VALUES IN PARENTHESES ARE STANDARD DEVIATIONS.

includes low-energy nodes in the backbone, and their lifetimes decrease significantly. Although SPAN rotates the backbone node responsibility, there exists an unfortunate low-energy node in most of our experiments that stays in the backbone during the first 350 seconds. In contrast, with TRUNC-1, the network 1-life is close to 960 seconds, which is 2.7 times longer than that of SPAN. This is because the TRUNC-1 backbone consists mostly of high-energy nodes plus a few medium-energy nodes, and low-energy nodes can stay in sleep mode and save energy. In the case of TRUNC-1, we observe a sharp increase in the number of dead nodes as the first node dies. This is the time (960 seconds) when all low-energy nodes in TRUNC-1 run out of power. Note that this is earlier than the case of lower-bound (around 1050 seconds). This is because with TRUNC-1, nodes consume more energy to exchange larger HELLO messages (in this experiment around 211 bytes on average) than the lower-bound case (128 bytes). Compared to SPAN, TRUNC-1 also increases the average lifetime of low-energy nodes by 28% (1038.1 seconds vs. 811.3 seconds), while decreasing the standard deviation by 68%.

We now consider the lifetime of medium-energy nodes in Figure 10. The use of low-energy backbone nodes in SPAN allows more medium-energy nodes to be in sleep mode and increase their lifetime. Still, compared to SPAN, the TRUNC-1 backbone increases the network half-life by around 26%. We explain this as follows. In this network setting, a connected backbone needs to use several medium-energy nodes to maintain connectivity. Ideally, as the initial medium-energy backbone nodes expend their energy, they should be replaced with different medium-energy nodes, such that their lifetimes do not decrease significantly. From Figure 10, we infer that TRUNC-1 evenly distributes the backbone responsibility among all medium-energy nodes, and no medium-energy nodes die until 1600 seconds. (In Figure 10, after all low-energy nodes die in the case of TRUNC-1 backbone, we observe a relatively stable period during which no node dies.) In contrast, in SPAN, medium-energy nodes start to die before 1200 seconds, and the network half-life of SPAN decreased.

In Table VII, we tabulate the average network lifetimes and standard deviations while varying the proportion of nodes at different energy levels. We observe that in all scenarios, TRUNC-1 achieves longer network lifetimes than SPAN (133–152% longer for 1-life and 20–26% longer for half-life). We also observe that the network 1-life by SPAN is longest when L:M:H=2:4:4. When there are more low-energy nodes, it is

	Energy consumption ratio (Idle:Sleep)		
	6.3:1 [6]	14.0:1 [22]	40.0:1 [7]
1-life	152	189	220
half-life	24	34	43

TABLE VIII

LIFETIME EXTENSION OVER SPAN (IN %) WITH DIFFERENT RATIOS BETWEEN IDLE AND SLEEP MODE ENERGY CONSUMPTION.

(L:M:H=3:4:3)

more likely in SPAN that at least one low-energy node stays in the backbone for a prolonged period, which makes the network 1-life shorter. We also experimented using different parameters (e.g., different L:M:H ratios and initial battery capacity values), and the results were similar. We note that in all these experiments, the average backbone sizes of TRUNC-1 and SPAN are very similar (between 21 and 23 nodes depending on the scenarios).

In the previous experiments, we have used the energy consumption values reported in [6]. In Table VIII, we briefly compare the results when we use different sets of values reported in [22] and [7]. We only show the ratio between idle and sleep energy consumption, which is the most dominant difference between these sets. Compared to the results in Table VII, as nodes in sleep mode consume less energy, TRUNC-1 achieves larger network lifetime extension over SPAN (e.g., up to 220% increase in 1-lifetime, compared to previous 152%).

*Backbone Maintenance:* We now examine the backbone resilience as well as the effectiveness of our proposed maintenance mechanisms against backbone partition. Let us define the *coverage* of a connected backbone to be the number of nodes that are in the backbone or have a neighbor in the backbone. For an ideal connected backbone, its coverage would always be equal to the number of nodes alive in the network. In Figure 11, we show the largest coverage of the TRUNC-1 backbone over time, while we change the maximum speed (1m/s, 8m/s, and 16m/s). Due to node mobility or energy-level change, the backbone may get disconnected, and we see occasional drops in the coverage of TRUNC-1 backbone. However, our protocol detects such disconnections quickly, and the local maintenance scheme helps to regain the perfect coverage in a short period of time. As node mobility becomes higher, we observe modest increase in the number of partitions in the TRUNC-1 backbone. Compared to the TRUNC-1 backbone, the SPAN backbone results in more frequent coverage loss (Figure 12). In SPAN, nodes periodically leave the backbone only after ensuring that the departure does not cause backbone disconnection. However, it is possible that a node makes such a decision based on outdated information (e.g., due to node mobility), which occurs frequently, for example, once every 30 seconds on average in Figure 12(a).

Another aspect of backbone maintenance is the frequency with which nodes in the backbone change. In the case of Figure 12(a), the SPAN backbone undergoes 674 membership changes between 100 and 400 seconds. This is because backbone nodes in SPAN periodically leave the backbone. In the same scenario, TRUNC-1 causes 49 changes in the backbone

	1 packet/sec	2 packets/sec	4 packets/sec
SPAN	0.96 (0.04)	0.88 (0.05)	0.62 (0.02)
TRUNC-1	0.98 (0.02)	0.92 (0.05)	0.58 (0.04)

TABLE IX

AVERAGE DELIVERY RATIO WITH VARYING TRAFFIC LOAD. THE VALUES IN PARENTHESES ARE STANDARD DEVIATIONS.

Max. speed	REQ	REPLY	CONNECT	RECOVER	HELLO
1m/s	1.80	2.15	1.64	0.12	52.73
8m/s	2.02	2.29	1.94	0.34	53.43
16m/s	1.83	2.04	2.14	0.49	53.65

TABLE X

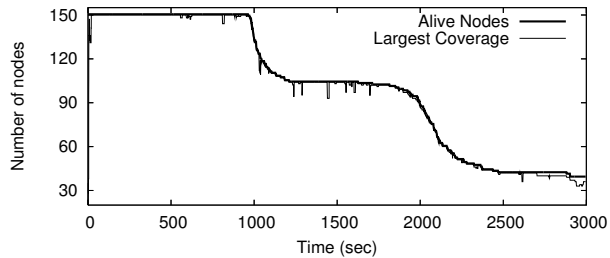
AVERAGE NUMBER OF CONTROL PACKETS PER SECOND IN THE ENTIRE 150-NODE NETWORK.

membership. Suppose that an on-demand routing protocol such as DSR [1] found a path using backbone nodes. In the case of SPAN, nodes on such a path are likely to leave the backbone about 12 times more frequently than TRUNC-1, and the source may need to find a new path consisting of backbone nodes frequently. (Recall that this is why we use the idealized routing in our experiments.)

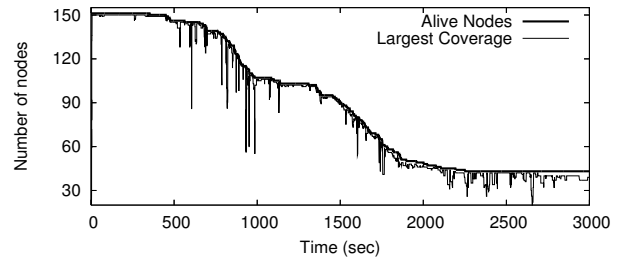
*Data Delivery:* We briefly report the results about data delivery performance of TRUNC-1 backbone. In the previous light-load experiments, both TRUNC-1 and SPAN achieve near-perfect data delivery ratios. In this set of experiments we experiment with high-load scenarios using 1024-byte data packets with varying packet rates. We also use static networks and ensure the distance between source and destination is more than 500 meters such that all data packets go through at least two intermediate hops. In Table IX, we tabulate the average data delivery ratios and standard deviations with different sending rates. We observe that as the amount of data traffic increases, the average delivery ratio decreases in both schemes, and the difference between TRUNC-1 and SPAN is marginal. In these experiments, TRUNC-1 leads to shorter average end-to-end delays than SPAN, but the difference is not significant.

*Control Overhead:* In both TRUNC-1 and SPAN, each node sends a HELLO message every two seconds. HELLO messages in TRUNC-1 contain more information, and the average message is longer than that of SPAN. Specifically, in TRUNC-1, the average length of HELLO messages is around 192 bytes, and in SPAN it is around 131 bytes. Note that the difference is due in part to more dead nodes in SPAN, which lead to fewer neighbors in HELLO messages.

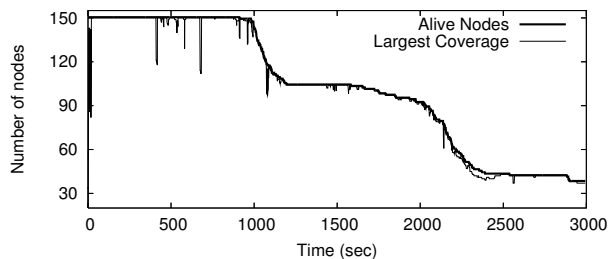
TRUNC-1 uses additional control messages (e.g., REQ and CONNECT), and in Table X we tabulate the average numbers of those control packets per second used in the *entire* network. As shown in the table, the total number of non-HELLO control packets is only around 6 packets per second in the 150-node network, and their average sizes are 20 to 70 bytes. When each of 150 nodes sends a HELLO message every two seconds, the expected number is 75 per second. In Table X, however, due to dead nodes, the number of HELLO messages is around 30% smaller. We also observe that the overall increase in control overhead due to higher mobility is marginal. We believe that



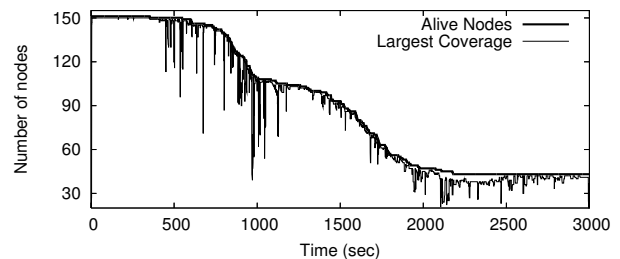
(a) Maximum speed=1m/s. (TRUNC-1)



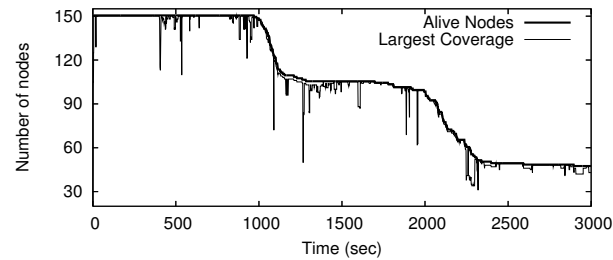
(a) Maximum speed=1m/s. (SPAN)



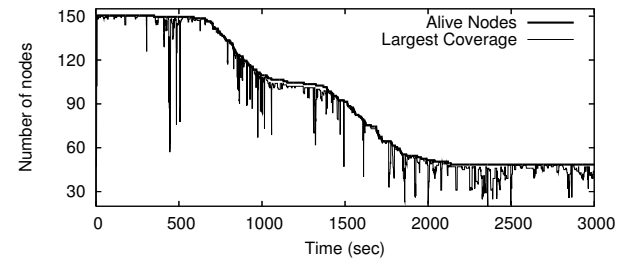
(b) Maximum speed=8m/s. (TRUNC-1)



(b) Maximum speed=8m/s. (SPAN)



(c) Maximum speed=16m/s. (TRUNC-1)



(c) Maximum speed=16m/s. (SPAN)

Fig. 11. TRUNC-1 backbone coverage.

Fig. 12. SPAN backbone coverage.

the advantages of TRUNC-1 (e.g., longer network lifetime, better backbone coverage) outweigh the modest increase in control overhead.

## VI. RELATED WORK

Many distributed algorithms are proposed to find a connected dominating set. Das and Bharghavan [2] directly apply well-known centralized algorithms [5]. Using the unit-disk graph model, Wan et al. [3] propose a message-optimal algorithm that achieves a constant approximation ratio. Dubhashi et al. [4] propose a distributed algorithm that finds an  $O(\log \Delta)$  approximation to the minimum connected dominating set in  $O(\log n \log \Delta)$  running time. None of them consider backbone maintenance or node capacity. As described in Section V, SPAN [6], GAF [7], and the scheme by Wu et al. [8] all consider remaining energy level when finding a connected

backbone. However, none achieves a maximum-capacity backbone, nor provides a bound on the backbone size. Also related are clustering algorithms for sensor networks [23], [24]. Kuhn et al. [23] propose a clustering algorithm that finds a dominating set in the initial deployment phase. HEED [24] selects cluster-heads based on the residual energy and other parameters such as node degree, but it assumes that the network is quasi-stationary. In contrast, TRUNC-K provides backbone maintenance mechanisms for dynamic networks.

TRUNC-K exploits a power saving mode to save energy. There are schemes that exploit the sleep mode operation, but are not based on the connected backbone approach. Zheng and Kravets [25] propose an on-demand power saving scheme, where nodes stay awake according to traffic load and their soft-state timers. Assuming dense sensor networks, ASCENT [26] uses a simple mechanism to determine “active” and “passive” nodes. However, ASCENT does not guarantee network con-

nectivity. We also can achieve energy saving through transmission power control at each node. Rodoplu et al. [27] present a localized algorithm that preserves network connectivity and achieves the globally minimum-energy topology. Wattenhofer et al. [28] propose a cone-based distributed topology control algorithm that guarantees network connectivity. Li et al. propose an MST-based local topology control algorithm, which they later generalize to provide  $k$ -vertex fault tolerance [29]. Such topology control schemes are complementary to our work.

## VII. CONCLUSIONS AND FUTURE WORK

We have presented a parameterized scheme TRUNC-K that builds a connected backbone in wireless ad-hoc networks. We have also proved that our scheme can construct essentially best possible backbones with respect to backbone size and node capacity. We generalized our scheme to construct and maintain a resilient backbone in dynamic networks. Through detailed simulations, we demonstrated that our proposed scheme outperformed existing energy-saving techniques in many aspects.

In the future, we plan to investigate how to adjust the  $K$  value according to network environments (e.g., node mobility or density). Then, we will be able to include adaptive protocol mechanisms that can automatically change the  $K$  value when network parameters change over time (e.g., increased mobility, or new node deployment). As discussed earlier, another obvious extension to the current scheme is to consider the difference in link quality [9]. In the future work, we will examine how to combine node capacity and link quality in nominating leaders and connecting fragments, and further investigate the combined effect of the two aspects.

## REFERENCES

- [1] D. Johnson and D. Maltz. *Dynamic Source Routing in Ad Hoc Wireless Networks*. Kluwer Academic Publishers, 2001.
- [2] Bevan Das and Vaduvur Bharghavan. Routing in ad-hoc networks using minimum connected dominating sets. In *International Conference on Communications*, June 1997.
- [3] P. Wan, K. M. Alzoubi, and O. Frieder. Distributed construction of connected dominating set in wireless ad hoc networks. In *Proceedings of IEEE INFOCOM*, 2002.
- [4] D. Dubhashi, A. Mei, A. Panconesi, J. Radhakrishnan, and A. Srinivasan. Fast distributed algorithms for (weakly) connected dominating sets and linear-size skeletons. In *Proc. of ACM-SIAM symposium on discrete algorithms*, 2003.
- [5] S. Guha and S. Khuller. Approximation algorithms for connected dominating sets. In *Proc. of the Fourth Annual European Symposium on Algorithms*. Springer-Verlag, 1996.
- [6] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris. Span: an energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. *Wireless Networks*, 8(5), 2002.
- [7] Y. Xu, J. Heidemann, and D. Estrin. Geography-informed energy conservation for ad hoc routing. In *Proceedings of the 7th MobiCom*, pages 70–84. ACM Press, 2001.
- [8] J. Wu, M. Gao, and I. Stojmenovic. On calculating power-aware connected dominating sets for efficient routing in ad hoc wireless networks. In *Proc. of IEEE ICPP*, 2001.
- [9] Daniel Aguayo, John Bicket, Sanjit Biswas, Glenn Judd, and Robert Morris. Link-level measurements from an 802.11b mesh network. In *ACM SIGCOMM*, September 2004.
- [10] M. Gerla and J. T. Tsai. Multicenter, mobile, multimedia radio network. *Wireless Networks*, 1(3):255–265, 1995.
- [11] J. Gao, L. Guibas, J. Hershberger, L. Zhang, and A. Zhu. Discrete mobile centers. In *Proc. of symposium on computational geometry*. ACM Press, 2001.
- [12] B. Bollobás. *Random Graphs*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 2001.
- [13] Uriel Feige. A threshold of  $\ln n$  for approximating set cover. *J. ACM*, 45(4):634–652, 1998.
- [14] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, 2nd Edition*. MIT press, 2001.
- [15] R. G. Gallager, P. A. Humblet, and P. M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Trans. on Programming Languages and Systems*, 1983.
- [16] R. E. Tarjan. *Data structures and network algorithms*. CBMS-NSF Conference Series in Applied Mathematics. SIAM, 1983.
- [17] IEEE. Wireless LAN medium access control (MAC) and physical layer (PHY) specifications. IEEE 802.11 Standard, 1999.
- [18] The VINT Project. The Network Simulator–ns-2. Available at <http://www.isi.edu/nsnam/ns>.
- [19] Valeri Naoumov and Thomas Gross. Simulation of large ad hoc networks. In *Proceedings of the 6th international workshop on Modeling analysis and simulation of wireless and mobile systems*, pages 50–57. ACM Press, 2003.
- [20] J. Yoon, M. Liu, and B. D. Noble. Random waypoint considered harmful. In *Proceedings of Infocom*, April 2003.
- [21] D. M. Blough and P. Santi. Investigating upper bounds on network lifetime extension for cell-based energy conservation techniques in stationary ad hoc networks. In *Proc. of Mobicom*, pages 183–192. ACM Press, 2002.
- [22] L. M. Feeney and M. Nilsson. Investigating the energy consumption of a wireless network interface in an ad hoc networking environment. In *Proc. of IEEE INFOCOM*, 2001.
- [23] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. Initializing newly deployed ad hoc and sensor networks. In *MobiCom '04: Proceedings of the 10th annual international conference on Mobile computing and networking*, pages 260–274. ACM Press, 2004.
- [24] Ossama Younis and Sonia Fahmy. Distributed clustering in ad-hoc sensor networks: A hybrid, energy-efficient approach. In *Proceedings of IEEE INFOCOM*, March 2004.
- [25] R. Zheng and R. Kravets. On-demand power management for ad hoc networks. In *Proc. of IEEE Infocom*, April 2003.
- [26] Alberto Cerpa and Deborah Estrin. ASCENT: Adaptive Self-Configuring sEnsor Networks Topologies. *IEEE Transactions on Mobile Computing*, 3(3), 2004.
- [27] V. Rodoplu and T. H. Meng. Minimum energy mobile wireless networks. *IEEE JSAC*, 17(8), August 1999.
- [28] L. Li, J. Y. Halpern, P. Bahl, Y. Wang, and R. Wattenhofer. Analysis of a cone-based distributed topology control algorithm for wireless multi-hop networks. In *Proceedings of ACM symposium PODC*. ACM Press, 2001.
- [29] Ning Li and Jennifer C. Hou. FLSS: a fault-tolerant topology control algorithm for wireless networks. In *Proceedings of Mobicom*, pages 275–286. ACM Press, 2004.
- [30] C. M. Fortuin, F. Ginibre, and P. N. Kasteleyn. Correlational inequalities for partially ordered sets. *Communications of Mathematical Physics*, pages 89–103, 1971.
- [31] M. Hofri. *Analysis of Algorithms*. Oxford University Press, 1995.
- [32] H. Shachnai and A. Srinivasan. Finding large independent sets in graphs and hypergraphs. In *ACM Symposium on Parallel Algorithms and Architectures*, 2001.

## APPENDIX

We describe a special case of the FKG inequality [30]. Consider an event  $F$  that is determined by a vector  $\vec{Y} = (Y_1, Y_2, \dots, Y_m)$  of independent random variables  $Y_i \in \{0, 1\}$ . Suppose that whenever  $F$  holds for  $\vec{a}$ ,  $F$  also hold for any  $\vec{b}$  that coordinate-wise dominates  $\vec{a}$  (i.e.,  $\forall i, a_i \leq b_i$ ). Then, we call  $F$  an *increasing* event of  $\vec{Y}$ . For increasing events  $F_1, F_2, \dots, F_l$ , the following holds:

$$\Pr\left(\bigwedge_{i=1}^l F_i\right) \geq \prod_{i=1}^l \Pr(F_i) \quad (4)$$

We also use the following identity from [31], which holds for any non-negative integer  $d$  and any real  $x \notin \{-d, -d +$

$1, \dots, 0\}$ :

$$\sum_{l=0}^d \binom{d}{l} \frac{(-1)^l}{x+l} = \frac{1}{x \binom{d+x}{d}} \quad (5)$$

### A. Proof of Theorem 2.2

We prove that in any  $D$ -regular graph with  $n$  nodes,  $E[|\mathcal{L}|] \leq c \frac{n}{D} \log(D+1)$  for a constant  $c$  that approaches 1 as  $D$  becomes large. Consider an indicator variable  $X_v$ , where  $X_v = 1$  iff  $v \in \mathcal{L}$ . Then,  $|\mathcal{L}| = \sum_v X_v$ . Let us denote by  $P_v$  the probability that node  $v$  is nominated as leader. Then, from the linearity of expectation:

$$E[|\mathcal{L}|] = E[\sum_v X_v] = \sum_v E[X_v] = \sum_v P_v \quad (6)$$

Then, to prove the theorem, it is sufficient to show:

$$\forall v, P_v \leq \frac{c}{D} \log(D+1) \quad (7)$$

Since all nodes have exactly  $D$  neighbors,  $P_v$  is same for all  $v$ 's. To show Inequality 7, we use the following:

$$\begin{aligned} P_v &= 1 - Pr[\text{no nodes nominate } v] \\ &= 1 - \int_0^1 Pr[\text{no nodes nominate } v | c_v = t] dt \quad (8) \end{aligned}$$

We define  $\mathcal{E}_i = Pr[i\text{-th neighbor of } v \text{ does not nominate } v]$ . We also denote  $\mathcal{E}_0 = Pr[v \text{ itself nominates other node}]$ . For each node  $u$ , consider the following binary random variable  $Y_u$ :  $Y_u=1$  iff  $c_u > c_v$ ;  $Y_u=0$  otherwise. Then,  $\mathcal{E}_i$  ( $0 \leq i \leq D$ ) is an increasing event of  $n$  random variable  $Y_u$ 's, and we can apply the FKG inequality to Equation 8, similarly to Shachnai and Srinivasan [32].

$$\begin{aligned} P_v &= 1 - \int_0^1 Pr[\mathcal{E}_0 \wedge \mathcal{E}_1 \wedge \dots \wedge \mathcal{E}_D | c_v = t] dt \\ &\leq 1 - \int_0^1 \prod_{i=0}^D Pr[\mathcal{E}_i | c_v = t] dt \\ &= 1 - \int_0^1 (1 - t^D)^{D+1} dt \quad (9) \end{aligned}$$

Let us define  $A = \int_0^1 (1 - t^D)^{D+1} dt$ . We want to find a constant value  $c \geq 1$  that satisfies the following:

$$A \geq \left(\frac{1}{D+1}\right)^{\frac{c}{D}} \quad (10)$$

Then, since  $x \geq 1 + \log x$  for all  $x > 0$ ,  $A \geq 1 - c \log(D+1)/D$ , and consequently,  $P_v \leq c \log(D+1)/D$ , which is what we want to show. Now, it remains to determine  $c$ . By taking the natural logarithm of Equation 10,  $c$  should satisfy:

$$c \geq -\frac{D}{\log(D+1)} \log A = \frac{D}{\log(D+1)} \log(A^{-1}) \quad (11)$$

We further simplify  $A$  by using Inequality 5:

$$\begin{aligned} A &= \sum_{l=0}^{D+1} (-1)^l \binom{D+1}{l} \int_0^1 t^{Dl} dt \\ &= \sum_{l=0}^{D+1} \frac{(-1)^l}{Dl+1} \binom{D+1}{l} \\ &= \left( \frac{D+1+1/D}{D+1} \right)^{-1} \\ &= \left( \prod_{i=1}^{D+1} \left(1 + \frac{1}{Di}\right) \right)^{-1} \quad (12) \end{aligned}$$

We note the following facts:  $H(n) = \sum_{i=1}^n 1/i \leq \log n + 1$ , and  $1+x \leq e^x$  for all  $x \geq 0$ . We denote  $\exp(x) = e^x$ . Then, we can find an upper bound of the righthand side of Inequality 11,

$$\begin{aligned} \frac{D}{\log(D+1)} \log(A^{-1}) &= \frac{D}{\log(D+1)} \log\left(\prod_{i=1}^{D+1} \left(1 + \frac{1}{Di}\right)\right) \\ &\leq \frac{D}{\log(D+1)} \log\left(\prod_{i=1}^{D+1} \exp\left(\frac{1}{Di}\right)\right) \\ &= \frac{1}{\log D} H(D+1) \\ &\leq \frac{\log(D+1) + 1}{\log(D+1)} \end{aligned}$$

Consequently, using the following  $c$  clearly satisfies Inequality 11:

$$c = 1 + \frac{1}{\log(D+1)} \quad (13)$$

It is easy to see that as  $D$  grows large,  $c$  approaches 1.

### B. Proof of Lemma 2.3

We denote the righthand side of Equation 13 as a function of degree  $d$ . Note that  $c(d)$  is decreasing where  $c(1) = 1 + 1/\log 2 \approx 2.44$ . Then, using similar steps in Appendix A, we can show:

$$E[|\mathcal{L}|] \leq \sum_{v \in V} \alpha c(d_v) \frac{\log(d_v + 1)}{d_v} \leq c' \sum_{v \in V} \frac{\log(d_v + 1)}{d_v}, \quad (14)$$

where  $c' = \alpha c(\delta)$  and  $\delta$  is the minimum degree in the network.

### C. Proof of Theorem 2.4

We first identify a lower bound on the size of an optimal dominating set ( $OPT$ ) in  $\alpha$ -locally-regular graphs. Consider the following Integer Program (IP) for a minimum dominating set:

$$\begin{aligned} &\text{minimize } \sum_{v \in V} x_v, \\ &\text{subject to } \forall v, x_v + \sum_{u: (u,v) \in E} x_u \geq 1, \quad x_v \in \{0, 1\} \end{aligned}$$

Relaxing the integrality constraints, we get a Linear Program (LP) where  $\forall v, x_v \geq 0$ . The optimal solution of LP is a lower

bound of the optimal solution of IP. Let us consider the dual of LP:

$$\begin{aligned} & \text{maximize } \sum_{v \in V} x_v, \\ & \text{subject to } \forall v, x_v + \sum_{u: (u,v) \in E} x_u \leq 1, x_v \geq 0 \end{aligned}$$

We show that for  $c'' = \frac{1}{\alpha+1}$ , using  $x_v = \frac{c''}{d_v}$  is a feasible solution of the dual. Clearly, this choice of  $x_v$  is positive. We now show that this choice of  $x_v$  satisfies the first set of constraints as well. Note that in  $\alpha$ -locally-regular graphs, for any neighbor  $u$  of node  $v$ ,  $d_u \geq d_v/\alpha$ .

$$\begin{aligned} x_v + \sum_{u: (u,v) \in E} x_u &= \frac{c''}{d_v} + \sum_{u: (u,v) \in E} \frac{c''}{d_u} \\ &\leq \frac{c''}{d_v} + \sum_{u: (u,v) \in E} \frac{c''}{d_v/\alpha} \\ &= \frac{c''}{d_v} + \frac{c''\alpha}{d_v} \\ &= c''(1 + \alpha) = 1 \end{aligned} \quad (15)$$

Since any feasible solution of the dual is a lower bound of any primal solution,  $OPT$  is lower-bounded by:  $OPT \geq c'' \sum_{v \in V} \frac{1}{d_v}$ . From Lemma 2.3,

$$\begin{aligned} E[|\mathcal{L}|] &\leq c' \sum_{v \in V} \frac{1}{d_v} \log(\Delta + 1) \\ &\leq c'(1 + \alpha) \log(\Delta + 1) OPT, \end{aligned} \quad (16)$$

which completes the proof of  $E[|\mathcal{L}|] = O(\log \Delta) OPT$ .

#### D. Proof of Theorem 3.1

We denote by  $v$  the minimum-capacity node of the resulting backbone. If  $v \in \mathcal{L}$ , by Theorem 2.1, the backbone is a maximum-capacity connected dominating set. We show the case where  $v$  is a part of a virtual edge. Consider the virtual edge that included  $v$  while connecting fragments  $F_1$  and  $F_2$ . Let us pick any two leaders each from  $F_1$  and  $F_2$  and call them  $L_1$  and  $L_2$ , respectively.

We first prove by contradiction that the following set is not connected:  $S = \{u \mid c_u > c_v\}$ . Note that  $L_1 \in S$  and  $L_2 \in S$ . Let us assume  $S$  is connected. Then,  $L_1$  and  $L_2$  have at least one path  $P$  consisting only of nodes in  $S$ . In this case,  $F_1$  and  $F_2$  can get merged using possibly multiple virtual edges using the nodes on  $P$ . As a result,  $F_1$  and  $F_2$  would not have chosen the virtual edge with  $v$ . This contradiction proves that  $S$  is not connected. Since  $S$  is not connected, no subset of  $S$  can be a connected dominating set.

#### E. Proof of $(2H(\Delta) + 1)$ -bound in [2]

Das and Bharghavan [2] have proposed a distributed connected dominating set algorithm that achieves the  $(2H(\Delta) + 1)$  bound (Lemma 2 in [2]). However, the corresponding proof is not available in the literature, and we present the proof here. Their algorithm first finds a dominating set  $DS$  using the greedy heuristic [14], which guarantees  $s \doteq |DS| \leq$

$H(\Delta)OPT$ . Then, after identifying connected components, they apply an MST-based algorithm using chains of up to two nodes. To get the desired bound, we should first connect two components using one node whenever available. If no pairs of components have common neighbors, then we connect components using chains of two nodes. Suppose that we have added  $x$  nodes before using chains of two nodes. Let us denote by  $y$  the number of remaining components at this point. Clearly,  $y \leq s - x$ . Since these components have no common neighbors, each component requires at least one distinct node in  $OPT$ , and  $y \leq OPT$ . We now use 2 nodes for each connection of remaining components, giving a total cost of:  $s + x + 2(y - 1) \leq 2s + y \leq (2H(\Delta) + 1)OPT$ .