Arbutus: Network-Layer Load Balancing for Wireless Sensor Networks

Daniele Puccinelli and Martin Haenggi
Network Communications and Information Processing Laboratory
Department of Electrical Engineering
University of Notre Dame
Notre Dame, IN, USA

***Abstract*—The *hot spot problem* is a typical byproduct of the many-to-one traffic pattern that characterizes most wireless sensor networks: the nodes with the best channel to the sink are overloaded with traffic from the rest of the network and experience a faster energy depletion rate than their peers. Routing protocols for sensor networks typically use a reliability metric to avoid lossy links and thus directly exacerbate the problem. Significant advantages can be obtained by embedding a load balancing scheme at the network layer, as we show with the design and implementation of Arbutus, a novel routing protocol for wireless sensor networks with a built-in load balancing scheme. By imposing a special structure on the collection tree, privileging longer hops, and accounting for network load in the route selection process, Arbutus reduces the impact of hot spots on network lifetime without a deterioration of the end-to-end reliability performance. An implementation of Arbutus on Berkeley motes and the MoteLab testbed shows a 30% reduction in the network traffic load needed to achieve the same packet delivery rate as an existing mote-oriented protocol. This provides key benefits such as a significant lifetime gain and increased fault tolerance.**

I. INTRODUCTION

The standard use of a wireless sensor network (WSN) is single-sink data collection, which naturally creates a many-to-one traffic pattern from the sensing nodes to the sink. Given the limited resources of WSNs, routing protocols normally avoid lossy links at all costs. Nodes with particularly favorable channels are thus likely to have a heavier workload than their peers, as they are chosen to relay traffic that they do not generate. This additional burden curtails the lifetime of these *critical nodes* and leads to network partitioning [1]. This phenomenon is known as the *hot spot problem*; it is the aim of *load balancing* schemes to avoid the formation of hot spots, or at least reduce the gravity of the problem and keep the network layer from ruining the energy conservation efforts of the lower layers.

The availability of multiple routes to the sink depends on the topology of the network and its surroundings and is constrained by the radio hardware and the physical layer. In the best possible load balancing scenario, all nodes can reach the sink in one hop and only send what they generate. At the opposite end of the load balancing spectrum, one particular relay or a small number thereof may be the only way for the sink to reach a whole subsection of the network, thus forming a *topological bottleneck*. An extreme case is a line network where only nearest-neighbor routing is possible. An example of this situation is shown in Fig. 1, which depicts a network where each node can only receive data from its upstream neighbor and can only send it to its downstream neighbor. The closer a node is to the sink, the higher its workload: each relay is a topological bottleneck with respect to the upstream nodes.

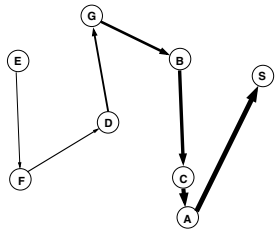

Figure 1. A line network with nearest-neighbor routing. The width of each link is proportional to the corresponding traffic volume.

In the presence of topological bottlenecks, load balancing alone cannot help. For bottlenecks artificially created by a routing strategy, however, we argue that a load balancing scheme directly embedded at the network layer may provide significant lifetime gains through a more efficient redistribution of the workload. To validate our argument we propose Arbutus, a novel cost-based routing protocol for WSNs with a built-in load balancing strategy. The main goal is the avoidance of a premature demise of the network given the constraints of existing lower-end sensing nodes. In this paper we provide a description of the architecture and a first performance evaluation of Arbutus, which we benchmark against an existing routing protocol. All the results presented herein have been obtained on MoteLab [2], a testbed of TMote Sky wireless nodes physically located at Harvard University and available for remote use through a web interface.

II. RELATED WORK

The significant resource constraints of lower-end nodes combined with the peculiarity of a many-to-one or many-to-few traffic pattern have discouraged the use of traditional ad hoc wireless routing protocols [3]. Despite numerous research efforts, WSN routing remains to this day a fairly open issue. Given the experimental nature of our work, in this section we will only focus on mote-oriented routing protocols that have been implemented on actual sensor network platforms.

A number of cost-based routing protocols have been developed for motes using TinyOS [4], an operating system specifically designed for WSNs. MintRoute [5], MultiHopLQI [6], and Collection Tree Protocol (CTP) [7] represent successive evolutions of a common cost-based paradigm defined in [5], which recognizes that the volatility of the wireless channel makes Boolean connectivity models not suitable for use in lower-end sensor networks with low-power radios and limited resources. Link estimation is seen as an essential tool for the computation of reliability-oriented route selection metrics. The sample application scenario is a sensor network with one sink (or potentially multiple sinks in CTP) where each sensing node generates traffic at a constant packet rate. Routing is broken down into three major components: a Link Estimator (LE) that continuously assesses the quality of the wireless links in the network; a Routing Engine (RE) that determines the address of the one-hop neighbor that provides the best progress toward the sink according to a given cost function based on the output of the LE; a Forwarding Engine (FE) that injects its own traffic or relays upstream traffic by unicasting to the address determined by the RE. The rationale behind this partitioning is the separation of the data plane (FE) from the control plane (RE), and the flexibility of choosing different LEs. The FE is a standard block that can be used across different protocols; the LE and the RE completely define a particular protocol. Connectivity discovery and route maintenance are carried out with the help of control beacons that diffuse global state used locally for route selection.

MintRoute (Mint stands for MInimum Number of Transmissions) employs routing tables and works with a LE based on the Expected Number of Transmissions (ETX) metric [8], but can also employ packet delivery rate (PDR) estimates based on sequence numbers. MintRoute adopts a neighborhood management policy based on the FREQUENCY algorithm [9] as a link blacklisting solution, thereby allowing a given node to only keep track of a subset of its neighbors. This way, MintRoute prevents the routing table from growing beyond a given size. In MultiHopLQI, neither routing tables nor blacklisting are used, and a new parent is adopted if it advertises a lower cost than the current parent. The link metric is Link Quality Information (LQI), an 802.15.4-specific form of Channel State Information (CSI) used additively to obtain the cost of a given route. MultiHopLQI avoids routing tables by only keeping state for the best parent at a given time; this measure drastically reduces memory usage and control overhead. The most recent protocol in this family, CTP [7], uses ETX and has a number of special features such as link estimation from both control and data traffic and transmission deferrals in case of parent node congestion. In parallel to the aforementioned protocols, a lightweight data collection protocol, Drain [10], was also developed along with its counterpart for query dissemination, Drip. Drain operates connectivity discovery and route maintenance through a sink-initiated reactive flood and performs route selection based on CSI and link-level acknowledgments (also employed by MultiHopLQI). Drain, like MultiHopLQI, only keeps state for one parent. None of these protocols explicitly pursues load balancing.

III. PROTOCOL DESCRIPTION

Arbutus is a novel routing protocol for WSNs with an embedded load balancing scheme. [1] Depth-limited beaconing is employed for connectivity discovery and route maintenance, and route selection is based on cross-layer bottleneck information. Arbutus tends to minimize the hop count of the routes under the constraints of the wireless channel. Among many other advantages, a smaller hop count directly entails a smaller control overhead [11]. The very structure of the collection tree promotes a low hop count: a beacon is processed only if it comes from a node at a lower depth, and all control traffic from nodes at higher or equal depth is ignored. In other words, a node at a given depth must choose a parent of lower depth, with the added benefit that routing loops are avoided by construction. It is also true, however, that a subset of valid links are ignored, as they would lead to longer routes. Though this greatly reduces the volume of control traffic, in principle shorter and less reliable routes may be chosen over longer and more reliable ones. Arbutus does not employ routing tables: like in MultiHopLQI and Drain, state information is only maintained for the current parent. Rather than the additive approach normally used to spread global state, Arbutus employs bottleneck information [12], [13], [14] which can be obtained with min or max operations over local information.

Link blacklisting is employed to reduce the chances of using asymmetric links, but is used with caution; [15] warns that it may lead to network partitioning. Experiments with MoteLab have shown cases where a node becomes unreachable for a certain amount of time (ranging from minutes to days) due to multipath fading: links that lie in the so-called transitional connectivity region [16] may slip into the disconnected region due to a change in the fading patterns of the environment (a door is closed, a piece of furniture is moved, ...). Motivated by this observation, Arbutus's control plane is augmented with a dynamic blacklisting threshold adaptation scheme.

A key feature of Arbutus is its embedded load balancing scheme. Since a pure reliability metric based on CSI overloads the critical nodes, Arbutus also takes load into account: it employs load estimates to identify overworked nodes and fuses them with CSI to obtain a cross-layer route selection metric.

A. Definitions

Bottleneck link quality. Let $(k, p(k)) = (p(k), k)$ represent a link between k and its designated parent $p(k)$, and let $l_{(k,p(k))} \in [0, 1]$ represent a normalized estimate of the quality of this link. We define the bottleneck link quality L_k seen by node k over an $h-$hop route to the sink as

$$L_k = \min_{j=0..h-1} l_{(p^j(k),p^{j+1}(k))} \quad (1)$$

where $p^0(k) = k$, $p^2(k) = p(p(k))$, ...

Bottleneck load. We define the load β_r of a relay r as the ratio of the number of relayed data packets (not locally generated) to the number of locally generated packets over a

[1] The name of an evergreen tree was whimsically chosen to indicate that this routing protocol constructs a tree that does not lose its leaves.

given timeframe, which in this paper coincides with the entire lifespan of r. We define the bottleneck load factor for node k, B_k, as

$$B_k = \max_{j=0..h-1} \beta_{p^j(k)}. \quad (2)$$

Cost. The cost incurred by node i if it awards parent status to node k, *i.e.*, if it routes its traffic over node k, is represented with the notation C^i_k.

B. Basic protocol description

State information. State at node k includes the *parent address* $p(k)$, the *bottleneck link quality* L_k, the *bottleneck load* B_k, the *depth* D_k (hop distance to the sink), a *parent loss counter* H_k, and a *parent use counter* O_k. The state information contained in the outgoing beacon $\mathbf{b}_k$ from node k must include the parent address, the bottleneck link quality L_k, the bottleneck load B_k, and the depth D_k. If node k is the sink, it starts up with the initializations $L_k := 1$, $B_k := 0$, and $D_k := 0$. If node k is not the sink, it runs a *no-route initialization* by setting $p(k) := INVALID\_NODE$ and $D_k := INVALID\_DEPTH$. A beacon and the internal state information of a node without a valid parent address will be referred to as a no-route beacon and no-route state. Beacons are to be sent in compliance with the beaconing management policy. Every node k sets $H_k := 0$ and $O_k := 0$ at startup.

Control beacon filtering. Node i, upon reception of $\mathbf{b}_k$ (used to estimate $l_{(k,i)}$), checks the three *beacon processing conditions*:

1) $p(i) = INVALID\_NODE$ and $D_k \neq INVALID\_DEPTH$, *i.e.*, node i is parentless and node k has a valid route to the sink.
2) $p(i) = k$, *i.e.*, node k is the current parent of node i.
3) Node k is not blacklisted[2] by i and $D_k < D_i$, *i.e.*, its hop distance to the sink is lower than the current depth.

If none of the three conditions above is met, no further action is needed. If any of them is met, the control beacon is accepted.

Parent management. Node i computes the cost of using node k, C^i_k, based on the state information advertised in $\mathbf{b}_k$ and the newly computed $l_{(k,i)}$. If any of the following *cost update conditions* is met:

1) $p(i) = INVALID\_NODE$ (i has no valid parent).
2) $p(i) = k$ (k is i's parent).
3) $C^i_k < C^i_{p(i)}$ (node k offers a lower cost of reaching the sink than the current parent $p(i)$).

then $p(i) = k$ if the following *valid parent conditions* are all met:

1) $p(k) \neq i$ (two-hop loop avoidance, *i.e.*, node i is not the parent of its prospective parent k).
2) $D_k < INVALID\_DEPTH$ (the potential parent has a route to the sink).

If all the valid parent conditions are met, then

- $p(i) := k$, *i.e.*, node k is selected as the new parent of node i.
- $L_i = \min(l_{(k,i)}, L_k)$.
- $B_i = \max(\beta_i, B_k)$.
- $D_i = D_k + 1$.
- $C^i_{p(i)} := C^i_k$.
- $H_i := 0$ and $O_i := 0$.

If any of the valid parent conditions is not met, then $p(i) \neq k$, and, specifically:

- $p(i) := INVALID\_NODE$.
- $D_i := INVALID\_DEPTH$.
- $C^i_{p(i)} := MAXINT$.

The idea behind the cost update conditions is that the routing cost should be updated if no valid parent exists, if the beacon is from the current parent, or if the beacon advertises a lower cost. If any of the cost update conditions is true, then the valid parent conditions kick in; if the prospective parent can be accepted or the current parent can be confirmed, then the internal state of node i is updated accordingly; otherwise, the internal state of i is initialized to no-route state.

Parent loss counter H_i counts the beacons sent by node i since a parent was (re)established. It is incremented every time a beacon is sent, and it is periodically checked against the route maintenance threshold T_H. Given the beaconing frequency, the number of sent beacons since the current parent was accepted or confirmed indicates the time since a beacon was received from the parent; in this sense, T_H works as a timeout to detect parent loss and the condition $H_i > T_H$, if met, triggers route maintenance at i. Parent use counter O_i counts the total number of data packets sent (relayed and generated) by i to the current parent $p(i)$ since the latter was (re)awarded parent status. It is incremented every time the data plane generates and sends a packet, which it may only do if $p(i) \neq INVALID\_NODE$ and $D_{p(i)} < INVALID\_DEPTH$. O_i is periodically checked against the parent use threshold Ω; the condition $O_i > \Omega$ is used by i to monitor its own use of $p(i)$.

Routing cost function. Upon reception of control beacon $\mathbf{b}_k$ node i computes the cost C^i_k of granting parent status to node k. Arbutus fuses physical layer and network layer information into a cross-layer cost function $C^i_k = f(l_{(k,i)}, L_i, B_k)$; specifically, we use

$$C^i_k = 2 - l_{(k,i)} - L_i + 2B_k. \quad (3)$$

Note that B_i is advertised in the outgoing beacon, but B_k is used in the cost function. In [17], it is stated that a lifetime-maximizing protocol should be adaptive to the network age by favoring the nodes with the best channels in the initial stages of operation, and by shifting to the nodes with the most residual energy later on. This is what Arbutus does: initially $B_k \ll 1$, and a parent is chosen if it offers a reliable route, but later on the traffic load kicks in and residual energy is also accounted for.

Beaconing management and route maintenance. Control beacons are sent by any node k at the rate f_s (fast beaconing frequency) starting with a uniformly random delay after the microcontroller boots. Afterwards, the beaconing frequency at node k is managed as follows.

- Slow beaconing: upon reception of a data packet addressed to node k or if $O_k > \Omega$, $f_b \ll f_s$ (slow beaconing

[2]Blacklisting is performed based on an adaptive policy described at the end of the present section.

frequency) is (re)enforced. Fast beaconing is used until node k successfully advertises its route, *i.e.*, until it is the intended receiver of a data packet, which means that it has been granted parent status. If node k does not have child nodes, it will never be the intended receiver of a data packet; for this reason, fast beaconing is also discontinued when $O_k > \Omega$, *i.e.*, once k has sent at least Ω (parent use threshold) data packets to the same parent.

- Fast beaconing: node k, if the route maintenance condition $H_k > T_H$ is met (in case of parent loss or if k remains parentless at startup for longer than the duration of the route maintenance threshold T_H), performs a no-route initialization and employs f_s (fast beaconing) to rapidly inform the child nodes that a route to the sink no longer exists. Dynamic blacklisting threshold adaptation is initiated.

Adaptive link blacklisting policy. Let $\Theta(t)$ be a time-varying link blacklisting threshold with $\Theta(0) = \Theta_0$, let b_n indicate the time when the $n-$th beacon is sent (assume $b_0 = 0$), and let $g(t)$ be a time-varying threshold adjustment function. The link blacklisting policy at node i dictates that if the link estimate $l_{(k,i)}$ based on $\mathbf{b}_k$ received at time b_n lies above a dynamically adapted threshold $\Theta(b_n) = \Theta(b_{n-1}) + g(b_n)$, then node i does not blacklist link (k, i) and processes $\mathbf{b}_k$. The threshold is decreased (by a constant value d) when a route is needed and cannot be found, presumably due to the absence of sufficiently good links. More precisely, the threshold is adjusted as follows: $g(b_n) = -d$ if the $n-$th outgoing beacon is a no-route beacon and $H_i > T_H$ (route maintenance condition), $g(b_n) = d$ if the $n-$th outgoing beacon is not a no-route beacon but the $n-1-$th is, and $g(b_n) = 0$ otherwise. The idea is that, at time b_n, node i refuses to use a downstream neighbor k such that $l_{(k,i)} < \Theta(b_n)$ unless there are no alternatives, in which case the threshold is forced down and k eventually gets used.

IV. PERFORMANCE EVALUATION

A. Baseline protocol

We choose MultiHopLQI as a baseline (as in [18]). Key differences between Arbutus and MultiHopLQI lie in the tree construction process, link estimation, and route selection. For tree construction, a parent is chosen if it offers a better route to the sink; CSI-based link costs are added up to obtain route costs. MultiHopLQI always picks the most reliable links and does not explicitly pursue load balancing. A certain amount of load balancing, however, is obtained in the presence of fading, which modifies link quality values and occasionally influences parent selection. We intend to benchmark Arbutus against MultiHopLQI to ascertain that the use of suboptimal links does not significantly deteriorate the overall performance of the protocol. We compare the two protocols with respect to standard routing performance metrics such as average PDR at the sink (computed as the ratio of the sum of the packets received from all nodes to the number of nodes) and hop count; to gauge the load balancing performance, we introduce a novel figure of merit, which we denote as *relaying cost-to-benefit ratio*. In a many-to-one traffic scenario where all nodes transmit at the same rate, the only factor that introduces a spread in the energy consumption of individual nodes is the relayed load. Therefore, we can estimate the lifetime gain based on ratio of the relayed load to the delivered load. Let $\mathcal{N}$ be the set of all nodes in a network, let $\mathcal{R}$ be the subset of all relays, and let us indicate the PDR for a node i at the sink as Π_i. In the economy of routing, the *benefit* is the total number of data packets received by the sink, whereas the *cost* is the total of all relayed data packets. The cost-to-benefit ratio η is thus defined as

$$\eta \triangleq \frac{\sum_{i\in\mathcal{N}} \beta_i}{\sum_{i\in\mathcal{R}} \Pi_i}. \tag{4}$$

The advantage of using the cost-to-benefit ratio is that it allows an estimate of the benefits of load balancing without the need to wait for energy depletion.

B. Implementation details

MoteLab's TMote Sky nodes are equipped with the CC2420 radio from Chipcon, which makes CSI available in two different forms: it provides a measure of the Received Signal Strength (RSS) with a 1dB resolution and the value of the correlation between the received codeword and the codebook entry associated to it by the decoder, made available in the aforementioned LQI field in compliance with the 802.15.4 protocol. LQI provides soft information about bad links (it correlates well with the PDR of a link), while RSS provides soft information about good links (it ranges from about -50dBm to about -90dBm, but typically anything above -80dBm corresponds to a unitary PDR in the absence of interference). RSS is supplied by most radios, whereas LQI is specific to 802.15.4-compliant devices. In implementing Arbutus, we leverage on both: we employ two blacklisting sub-strategies (and thus need two thresholds, Θ_0^{RSS} and Θ_0^{LQI}), and the end decision is the logic AND of their outcomes. LQI alone is used for the routing cost function.

In the experimental evaluation, all nodes in the network generate traffic of fixed size (30 bytes for both protocols) at a constant rate. Specifically, node k generates and sends a data packet to parent $p(k)$ (as determined by the control plane) with traffic generation rate f_{gen}. In Arbutus, upon reception of a data packet from node i ($p(i) = k$), node k enters it into a queue of length Q and forwards it to $p(k)$ (queueing is necessary as k may have several child nodes). Data packet generation and forwarding are halted when $p(k) = INVALID\_NODE$. Arbutus comes with several tunable parameters; in particular, we set f_b =1 pkt/min, f_s =6 pkts/min, $\Theta_0^{\text{RSS}} = -88$dBm, and Θ_0^{LQI}=50. Link-level acknowledgments and retransmissions are not used for either protocol (we operate with a heavily congested network, and retransmissions would not help [19]). Both protocols are built on top of a B-MAC medium access layer [20]. It is worth pointing out that the beaconing frequency normally used by this implementation of Arbutus (f_b) is roughly half of the beaconing frequency used in MultiHopLQI, which guarantees Arbutus to incur a significantly lower control overhead.

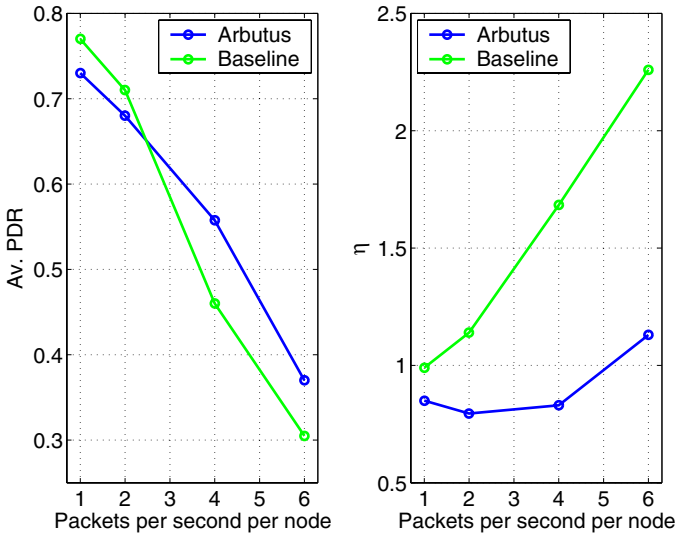

Figure 2. Average PDR and η for Arbutus and MultiHopLQI. At low loads, the use of suboptimal links comes at the cost of a slightly lower PDR.

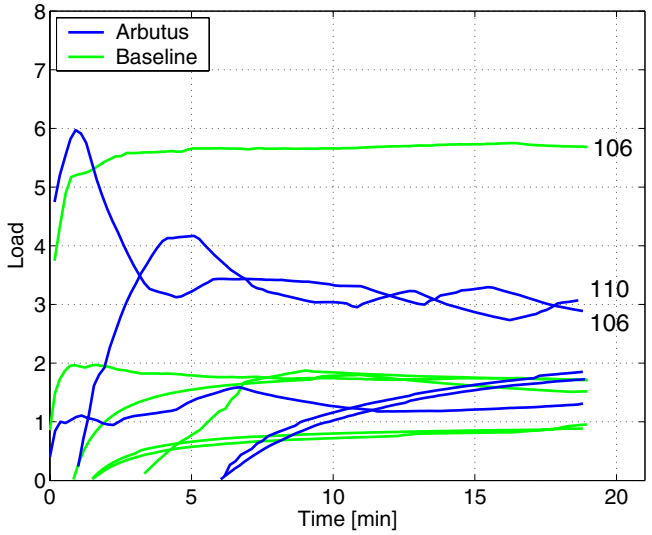

Figure 3. Time evolution of the load of all the relays in our sample experiment. The considerable difference in the workload of node 106 is due to the load balancing action of Arbutus reflected by the value of η (0.8 for Arbutus, 1.2 for the baseline protocol MultiHopLQI).

C. Experimental results on the MoteLab testbed

We first focus on a subset of the MoteLab testbed and use a 23-mote network whose nodes are distributed on two different floors. We use four different values of f_{gen} (1, 2, 4, and 6 pkts/sec per node) and let every node generate traffic addressed to the sink. Arbutus constructs a slightly shallower collection tree, with an average depth of 1.9 (2.1 for MultiHopLQI) and an average maximum depth of 3 (3.2 for MultiHopLQI) over all our experiments with this subnetwork. Fig. 2 shows the average PDR at the sink, computed as the average of the PDR for each node over the whole experiment, and the cost-to-benefit ratio η as were obtained by repeating a data collection experiment (of the duration of at least 15 minutes) on the 23-mote subnetwork 4 times for each value of f_{gen}. Despite not always favoring the most reliable links, Arbutus

	Arbutus	Baseline	Ratio
Average fraction of nodes reached	0.88	0.80	1.10
Average PDR	0.33	0.27	1.21
Average same-floor PDR	0.62	0.57	1.09
Average hop count	4.1	4.6	0.91
Maximum hop count	8.7	8.9	0.98
Average η	5.00	7.13	0.70

Table I
BENCHMARKING RESULTS ON THE COMPLETE MOTELAB TESTBED (f_{gen} = 1 PKT/SEC).

almost performs as well as MultiHopLQI in terms of end-to-end reliability at lower traffic loads, and performs considerably better as the traffic generation rate is increased. By actively balancing the traffic load, Arbutus constructs a more efficient tree and needs a smaller per-relay load to achieve a better average PDR at the sink. In order to appreciate the meaning of η and its impact on performance, Fig. 3 shows one instance of the time evolution of the load of each relay with both protocols. In this particular experiment (f_{gen} = 2 pkts/sec), η = 0.8 for Arbutus and η = 1.2 for MultiHopLQI. While MultiHopLQI peruses node 106 to take advantage of its high-quality connectivity to the sink, Arbutus redistributes the load over 106 and 110 and reduces the overall amount of traffic load to be relayed by employing shorter routes; the load balancing benefits also show in terms of PDR (in this experiment it is 0.59 for Arbutus and 0.48 for MultiHopLQI).

To properly benchmark Arbutus against MultiHopLQI, we now present the results of our experiments with the complete MoteLab testbed, which is extremely challenging in terms of load balancing. Its nodes are distributed across three floors with limited inter-floor connectivity, resulting in the presence of topological bottlenecks. We choose 5 different sink assignments, perform at least 3 experiments for each assignment at various times of the day and night (we use f_{gen}=1 pkt/sec), and then average all values; the results of our benchmarking are shown in Table I. The average PDR is computed as the average of the PDR for each working node[3] in the testbed measured by the sink over the whole experiment. Similarly, the average same-floor PDR is computed over all working nodes located on the same floor as the sink. On average, the performance of Arbutus is substantially better; considering that the working nodes in the testbed during our experiments ranged from 83 to 96 (the average number of working nodes over all experiments is 89), this benchmarking confirms Arbutus's scalability properties. Arbutus's tree is slightly shallower, as its average depth is 91% of the depth of MultiHopLQI's.

The 30% average reduction in the relaying cost-to-benefit ratio η shows the added value of Arbutus's built-in load balancing scheme. MultiHopLQI always picks the best channel but introduces imbalance in the process, overburdening the critical nodes with extra load; in the short term, this leads to packet loss (interference and congestion) that offsets the benefits of using the best links, while in the long run it curtails network lifetime. Fading induced by activity in the environment [21] is the main reason for the significant differences between day and night experiments; results also vary considerably depending

[3]A MoteLab node is a working node if it can be programmed to send packets over the wired backchannel. At any given time, not all nodes in the testbed are working as some are disabled for reasons beyond our control.

on the particular sink assignment. Over our experiments, the standard deviation σ of η is 2.5 for Arbutus and 2.8 for MultiHopLQI. The reduced amount of control traffic employed by Arbutus comes at a delay price: the average time of first contact (defined as the time the first data packet is received by the sink) is about 20s higher; this is only an issue at startup, and could be improved with a more refined adaptive blacklisting scheme. On the contrary, the average latency of Arbutus is considerably lower, as it is proportional to the hop count (no retransmissions).

V. CONCLUSIONS AND RESEARCH DIRECTIONS

Arbutus, our lightweight routing protocol for WSNs, effectively embeds a load balancing scheme at the network layer. Benchmarking experiments on the MoteLab testbed show that the end-to-end reliability performance of Arbutus compares favorably with the baseline protocol, MultiHopLQI. In addition to that, the relaying cost-to-benefit ratio of the baseline protocol is reduced by 30%, *i.e.*, Arbutus can achieve the same data delivery performance with a reduced network load. This can translate into a significant lifetime gain: a reduced load implies energy savings for the network as a whole, but in particular for the critical nodes, whose workload is reduced insofar as allowed by the topology of the network. At the same time, if the per-node load is decreased, node failures are, on average, not as critical, and the network benefits from an improved fault tolerance.

In this paper, we employ Arbutus in static energy conditions: the nodes are wall-powered, and we estimate energy consumption based on load. The main problem with this approach is that we do not consider the potentially disruptive effects of node depletion. Our idea, which we are implementing in our current work, is to emulate battery discharge at the nodes based on a simple estimate of the radio energy consumption at the individual nodes.

Several aspects of the protocol can be improved. The main issue with the basic version of Arbutus presented herein is the large number of parameters, which require careful calibrations. Calibrations are particularly critical for the blacklisting strategy, which can cause partitioning if the thresholds are not set properly.

The work in this paper also provides the experimental motivation for a study of topological bottlenecks in sensor networks and the constraints imposed by network topology on load balancing. If a node is a topological bottleneck, its neighbors will send their traffic through it, as no other node provides a route to the sink. In a similar case, network-layer load balancing cannot be enough, and additional measures are necessary to address the hot spot problem. One direction worth pursuing is the augmentation of Arbutus with sink mobility or sink repositioning support.

ACKNOWLEDGMENTS

The authors would like to thank Geoff Werner-Allen and Matt Welsh for making their MoteLab testbed available to the community. The support of NSF (CNS 04-47869) and CRANE/DTRA (N00164-07-8510) is gratefully acknowledged.

REFERENCES

[1] M. Haenggi. Energy-Balancing Strategies for Wireless Sensor Networks. In *IEEE International Symposium on Circuits and Systems (ISCAS'03)*, Bangkok, Thailand, May 2003.

[2] G. Werner-Allen, P. Swieskowski, and M. Welsh. MoteLab: a wireless sensor network testbed. In *Proceedings of the 4th international symposium on Information Processing in Sensor Networks (IPSN'05)*, Los Angeles, CA, USA, April 2005.

[3] T. Stathopoulos. *Exploiting Heterogeneity for Routing in Wireless Sensor Networks*. PhD thesis, University of California at Los Angeles, October 2006.

[4] TinyOS. http://www.tinyos.net.

[5] A. Woo, T. Tong, and D. Culler. Taming the Underlying Challenges of Reliable Multihop Routing in Sensor Networks. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys'03)*, Los Angeles, CA, USA, November 2003.

[6] MultiHopLQI. http://www.tinyos.net/tinyos-1.x/tos/lib/MultiHopLQI.

[7] Collection Tree Protocol (CTP). http://www.tinyos.net/tinyos-2.x/doc/html/tep123.html.

[8] D. De Couto, D. Aguayo, J. Bicket, and R. Morris. A High-Throughput Path Metric for Multi-Hop Wireless Routing. In *Proceedings of the 9th Annual International Conference on Mobile Computing and Networking (MobiCom'03)*, San Diego, CA, USA, 2003.

[9] E. Demaine, A. Lopez-Ortiz, and J. Munro. Frequency Estimation of Internet Packet Streams with Limited Space. In *Proceedings of the 10th Annual European Symposium on Algorithms (ESA'02)*, Rome, Italy, September 2002.

[10] G. Tolle and D. Culler. Design of an Application-Cooperative Management System for Wireless Sensor Networks. In *Second European Workshop on Wireless Sensor Networks (EWSN'05)*, Istanbul, Turkey, January 2005.

[11] M. Haenggi and D. Puccinelli. Routing in Ad Hoc Networks: A Case for Long Hops. *IEEE Communications Magazine*, 43, October 2005.

[12] D. Puccinelli, E. Sifakis, and M. Haenggi. A Cross-Layer Approach to Energy Balancing in Wireless Sensor Networks. In *Workshop on Networked Embedded Sensing and Control (NESC'05)*, Notre Dame, IN, USA, September 2005.

[13] D. Puccinelli, M. Brennan, and M. Haenggi. Reactive sink mobility in wireless sensor networks. In *Proceedings of the 1st International MobiSys Workshop on Mobile Opportunistic Networking (MobiOpp'07)*, San Juan, Puerto Rico, June 2007.

[14] B. Chen, K. Muniswamy-Reddy, and M. Welsh. Ad-hoc multicast routing on resource-limited sensor nodes. In *Proceedings of the second international workshop on Multi-hop ad hoc networks: from theory to reality (REALMAN'06)*, Florence, Italy, May 2006.

[15] O. Gnawali, M. Yarvis, J. Heidemann, and R. Govindan. Interaction of retransmission, blacklisting, and routing metrics for reliability in sensor network routing. In *Proceedings of the First IEEE Conference on Sensor and Adhoc Communication and Networks (SECON'04)*, pages 34–43, Santa Clara, CA, USA, October 2004. IEEE.

[16] M. Zuniga and B. Krishnamachari. Analyzing the Transitional Region in Low Power Wireless Links. In *Proceedings of the First IEEE International Conference on Sensor and Ad hoc Communications and Networks (SECON'04)*, Santa Clara, CA, USA, October 2004.

[17] Q. Zhao, A. Swami, and L. Tong. The Interplay Between Signal Processing and Networking in Sensor Networks. *IEEE Signal Processing Magazine*, 23(4):84–93, July 2006.

[18] M. Wachs, J. Choi, J. Lee, K. Srinivasan, Z. Chen, M. Jain, and P. Levis. Visibility: A New Metric for Protocol Design. In *Proceedings of the Fifth ACM Conference on Embedded Networked Sensor Systems (SenSys'07)*, Sidney, Australia, November 2007.

[19] U. Malesci and S. Madden. A Measurement-based Analysis of the Interaction Between Network Layers in TinyOS. In *European Workshop on Wireless Sensor Networks (EWSN'06)*, Zurich, Switzerland, February 2006.

[20] J. Polastre, J. Hill, and D. Culler. Versatile Low Power Media Access for Wireless Sensor Networks. In *Proceedings of the 2nd ACM Conference on Embedded Networked Sensor Systems (ACM SenSys'04)*, Baltimore, MD, November 2004.

[21] D. Puccinelli and M. Haenggi. Spatial Diversity Benefits by Means of Induced Fading. In *Third IEEE International Conference on Sensor and Ad Hoc Communications and Networks (SECON'06)*, Reston, VA, USA, September 2006.