ARBUTUS: RELIABLE AND ENERGY-EFFICIENT DATA COLLECTION IN LARGE-SCALE LOW-POWER WIRELESS SENSOR NETWORKS

A Dissertation

Submitted to the Graduate School
of the University of Notre Dame
in Partial Fulfillment of the Requirements
for the Degree of

Doctor of Philosophy

by

Daniele Puccinelli,

Martin Haenggi, Director

Graduate Program in Electrical Engineering
Notre Dame, Indiana
July 2008

ARBUTUS: RELIABLE AND ENERGY-EFFICIENT DATA COLLECTION IN LARGE-SCALE LOW-POWER WIRELESS SENSOR NETWORKS

Abstract

by

Daniele Puccinelli

In data collection applications of low-power wireless sensor networks, a major challenge is ensuring reliability without a significant goodput degradation. Retransmissions over lossy links are often necessary, but energy is at a premium. The use of short, high-quality hops minimizes the number of per-hop retransmissions, but unnecessarily long routes cause congestion and load imbalance. While the former induces packet loss in the form of ingress drops, the latter causes a non-uniform energy depletion pattern in the network that penalizes the nodes with the best channel to their next-hop neighbor. An unbalanced network imposes a particularly heavy burden on the neighbors of the sink that have to relay all the upstream traffic; in a many-to-one collection tree, however, the lifetime of the sink neighbors upper-bounds the lifetime of the network as a whole.

There exists a complex interplay among routing performance (reliability, goodput, energy-efficiency), congestion control, and load balancing. We present an energy-aware data collection architecture for sensor networks, Arbutus, that leverages on this interplay. We thoroughly describe the various aspects of the Arbutus architecture and present the results of the experimental evaluation of its TinyOS implementation.

The evaluation is performed on large-scale remote-access testbeds of Berkeley motes (MoteLab, Twist, Tutornet) with 100-150 nodes. Arbutus is benchmarked against the standard TinyOS 2.x network protocol, the Collection Tree Protocol. An extensive performance analysis is conducted, and the broader implications of our results are shown. Tools from the field of transport geography are employed to interpret the most elusive aspects of our evaluation.

Arbutus is shown to achieve a level of reliability suitable for most applications while ensuring energy-efficiency. While goodput degradation is contained at low levels of offered load, at relatively high offered load points Arbutus typically provides a significant goodput gain, as it exploits the inter-dependencies between reliability, routing and congestion control.

CONTENTS

FIGURES

CHAPTER 1

INTRODUCTION

1.1 Communication challenges of low-power sensor networks

Due to their resource limitations, low-power wireless sensor networks (WSNs) pose considerable communication challenges. One of the most significant is preventing packet loss while maintaining an acceptable goodput. Aside from catastrophic problems such as hardware or software failure and battery depletion, packet loss may occur due to channel errors, congestion-induced buffer overflow, and protocol-level inefficiencies.

Wireless propagation effects such as large-scale path loss, shadowing, and multipath fading contribute to the attenuation of the signal power. Differently from high-end wireless networks such as WLANs, cellular networks, or mobile ad hoc networks (MANETs), low-power WSNs typically employ low-complexity transceivers with a maximum transmit power of 0dBm. If the attenuation brings the signal strength too close to the noise floor, it becomes impossible to correctly receive packets. Interference also causes packet loss due to the collision of packets at the receiver. All these channel-induced losses can be limited through link estimation and the selection of high quality links, and/or by strengthening wireless links through the use of error-control coding or automatic repeat request (ARQ) schemes. Coding has not found much use in WSNs due to the computational complexity of decoding (which has been shown to offset the energy savings made possible by the reduction

of the transmit power [1]). ARQ schemes, however, are typically used; for example, the IEEE 802.15.4 stack employs link-layer acknowledgments. Channel-induced packet loss can be (almost) eliminated by not imposing an a priori constraint on the maximum number of retransmissions, with an inherent tradeoff between reliability, delay, and energy efficiency. Residual channel-related packet loss remains even with an unconstrained number of retransmissions (henceforth referred to as unconstrained retransmissions) due to failures at the Medium Access Control layer (MAC) such as false ACKs.

Link-layer retransmissions have been shown to be highly beneficial [2]. The main problem with ARQ in WSNs is that sensing nodes are typically static: if the channel between two nodes is subject to a deep fade, no improvement can be expected until the fading patterns of the deployment area change, which may take an indefinite amount of time. In other words, the channel coherence time tends to be very large, unlike in MANETs where mobility makes ARQ rather effective. Lossy links that would require a large number of retransmissions should therefore be avoided, hence the importance of link estimation in WSN routing.

Congestion is particularly severe in WSNs due to their typical many-to-one traffic pattern. Depending on the offered load, the data buffers of the nodes may overflow. The only way to truly eliminate ingress drops (at the cost of a large control overhead[1]) is by way of a pull-based paradigm [3] whereby nodes pull data from their upstream descendants. With the more common push-based paradigm, nodes push their data downstream to their parents, and post-facto congestion control is adopted. This approach has a much lower control overhead, but cannot be expected to completely prevent packet loss.

Further reasons for packet loss are found in network protocols. A common form

[1]The control overhead is the ratio of the control traffic (received and sent) over all traffic.

of protocol-level inefficiency is the formation of routing loops, which typically occur when a node loses its parent (for instance, a congested parent may declare itself unavailable to avoid ingress drops) and adopts a descendant as its new parent. Egress drops, *i.e.*, the elimination of perfectly good packets erroneously believed to be flawed (*e.g.*, false duplicates) are another typical example. Incompatibility between protocols pertaining to different layers is yet another potential reason for delivery performance degradation; a well-known example is the use of a network protocol that requires promiscuous mode operation for link estimation along with a MAC protocol that avoids snooping to save energy [4]. Despite many research efforts, these issues largely remain unsolved [3].

1.2 Main contributions of this dissertation

This dissertation makes the following contributions to sensor network routing for data collection:

Arbutus architecture. In the vast design space for WSN routing, we focus on distributed schemes for tree-based data collection in homogeneous networks. We propose Arbutus[2], a novel routing architecture for low-power WSNs that leverages on several recent developments in the sensor network community. The main principle behind the Arbutus architecture is that routing over a few long hops can be much more efficient than routing over many short hops [5, 6]. The Arbutus architecture contains three main contributions: a novel *tree-based routing* and *load-balancing* protocol, a novel *hybrid link estimator*, and a novel *congestion control* scheme. Arbutus is implemented in TinyOS [7], the de facto standard operating system for WSNs, on top of the standard CSMA-based MAC layer.

[2]The name *Arbutus* whimsically refers to an evergreen tree, suggesting that Arbutus builds a spanning tree that does not lose its leaves.

Testbed-based experimental approach. The vast majority of WSN protocols studies mostly employ simulation. The main reason is that it is extremely difficult to perform an extensive evaluation on resource-constrained motes [8]. The Arbutus architecture, however, is designed for mote networks. We purposefully avoid simulation as an evaluation tool and adopt an experimental approach. In order to evaluate Arbutus on large-scale networks of 100-150 nodes, we use remote-access testbeds at Harvard University (MoteLab [9]), Technische Universitaet Berlin (Twist [10]), and the University of Southern California (Tutornet[3]).

Benchmarking against existing protocols. For a meaningful comparison, a benchmark is chosen among schemes that fall in the same subset of the WSN routing design space as Arbutus. The Collection Tree Protocol (CTP)[4], the latest evolution of the MintRoute [11] architecture, is also a distributed scheme for tree-based data collection in homogeneous WSNs; it is the reference routing protocol for TinyOS 2.x, has been strenuously tested and shown to work well in mote networks, and has already been used in several studies [3, 12–17]. At the time of writing, however, a specific study on CTP's performance had not been published; since we employ it as a benchmark, a side contribution of this dissertation is to provide a thorough evaluation of CTP that will benefit its many users.

For the analysis of the lifetime benefits, we employ a widely used table-free WSN routing protocol, MultiHopLQI[5], as a benchmark.

Transport-geographical interpretation and impact of the topology. We employ tools from the field of transport geography [18] to interpret some of our results. In par-

[3]http://enl.usc.edu/projects/tutornet

[4]At the time of writing, the most comprehensive description of CTP is contained in a TinyOS Enhancement Proposal, available at `http://www.tinyos.net/tinyos-2.x/doc/html/tep123.html`.

[5]Available at `http://www.tinyos.net/tinyos-1.x/tos/lib/MultiHopLQI`.

ticular, we use transport geography to study the impact of network topology on the routing and load balancing performance of Arbutus. Our emphasis on the impact of network topology is another key contribution, since it is fairly common to fix the topology and vary other parameters (such as the offered load) while studying routing schemes. Our analysis also provides network design guidelines for node placement in sensor networks.

CHAPTER 2

BACKGROUND AND RELATED WORK

2.1 Generalities

Network and nodes. We denote the set of all nodes in a given network by $\mathcal{N} \subset \mathbb{N}$; for simplicity, we will refer to $\mathcal{N}$ as the network, but we intend it to only denote the set of nodes (and not the set of edges). We assume the presence of only one sink in the network, denoted as $s \in \mathcal{N}$. The application model assumed in this dissertation is many-to-one data collection: every node generates data packets with target rate f_{gen} and routes them to s using a distributed routing protocol. The parent of a node i, denoted as $p(i)$, is the destination of the data packets unicast by i. We use the symbol z for an invalid node. We define the *relayed load* β_i of node i as the number of relayed packets per time unit (measured in pkts/sec). The *hop count* of node i from the sink depends on the routing protocol and is denoted by H_i.

Links. We denote a directed wireless link from node i to node j as (i, j). This notation indicates that the link is *physical*: if i transmits packets using a given physical layer and a set transmit power, j receives at least one of the packets over a given time window T (or else, (i, j) is said not to exist within T). Node j is said to be i's *neighbor* if (i, j) and (j, i) exist. We further define the concept of *generalized link*, which we indicate as $[i, j]$, to represent a route between i and j whose intermediate relays may be unknown. We denote as $[i, s]_k$ the set of all the relays used by the

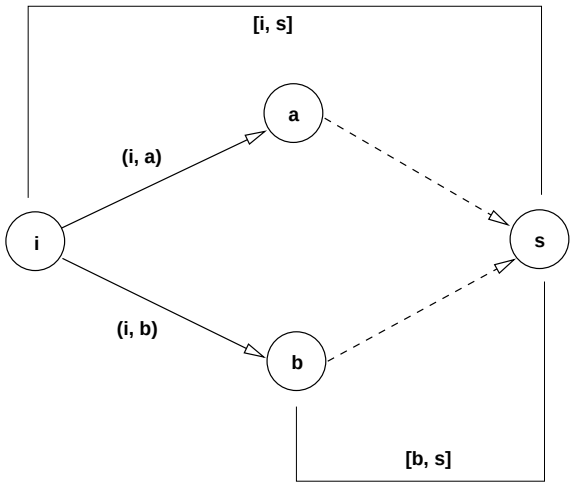


Figure 2.1. The arrows show the direction of unicast data traffic. Solid arrows represent physical links, while dashed ones represent generalized links.

routing protocol (at a given time) to get i's packets to s using k as the first relay. Figure 2.1 clarifies our notation using a distributed routing example viewed from the perspective of a given node i. Node i wants to send a packet to the sink s. A distributed routing protocol runs at every node, thereby allowing each node to know the address of the next hop toward the sink. Let us visualize the network from i's point of view: i has physical links to its neighbors a and b (they can all hear each other directly). Node i, however, does not know what lies beyond its neighbors, other than the fact that the sink s is somewhere downstream from them. Therefore, we say that i and s are connected by a generalized link, $[i, s]$. If $p(i) = a$, then $[i, s]$ corresponds to $[i, s]_a$. Likewise, if $p(i) = b$, then $[i, s]$ corresponds to $[i, s]_b$. It is up to the distributed routing protocol to determine whether a or b should be $p(i)$.

Link performance measures. We define the packet delivery rate (PDR) over (i, j), $\pi_{i,j}$, as the ratio between the number of packets received by j over the number of packets sent by i to j over a given time window. The PDR is therefore an empirical

probability of reception over (i, j). We define a soft connectivity matrix based on PDR as $\mathbf{\Pi} = \{\pi_{i,j}\}$. Also based on PDR, we can define the concept of *asymmetric link*. In this dissertation, a link (i, j) is said to be asymmetric if $|\pi_{i,j} - \pi_{j,i}| > 0.5$. The delivery ratio at the sink for node i, defined as the number of packets generated by node i and delivered to the sink (either directly or through multiple hops), is denoted as e_i.

Another link performance measure is the Required Number of Packets (RNP) [19], which indicates the total number of transmissions that are needed by node i to get a packet across $(i, p(i))$ (averaged over a given time window). Differently from the PDR, the RNP is not tied to a specific link, as $p(i)$ may change while the same packet is being retransmitted. Indeed, if a given parent requires too many retransmissions, parent rotation is normally encouraged by the network layer (parent changes dictated by a routing protocol are whimsically known as *churn*). Therefore, we employ the notation M_i to indicate the RNP over $(i, p(i))$, deliberately leaving out $p(i)$ from the notation.

Link performance estimates. Link performance estimates rely on Channel State Information (CSI); common forms thereof are the Received Signal Strength (RSS) and the Link Quality Indicator (LQI). The RSS used to be considered a poor predictor of link quality, mostly because of the limitations of early platforms [20]. The community has recently focused on the 802.15.4 stack, and, in particular, on motes built around the CC2420 transceiver, which provides a much more reliable RSS indication. The RSS has recently been recognized as a good predictor of link quality; specifically, it has been shown that the RSS, if higher than about -87dBm, correlates very well with the PDR [21]. While most radios provide a RSS indicator, the LQI is specific to 802.15.4, and its implementation is left to the radio chip designer. In the

CC2420, the LQI is implemented as the sum of the first 8 correlation values after the Start of Frame Delimiter represented as a 7 bit unsigned integer value.

Critical set. The neighborhood of the sink, $\mathcal{N}_1 = \{i : 1 \leq H_i \leq 1+\epsilon\}$ (with $\epsilon \geq 0$), can be referred to as the *critical set*, as it contains the nodes that must relay all upstream traffic to the sink. The extra workload makes these nodes critical: their batteries get drained at a faster rate than their upstream peers, and their demise causes the disconnection of the sink from the rest of the network. We further define the *critical fraction* as the ratio of the cardinality of the critical set to the total number of nodes in the network. We will employ $\epsilon = 1.5$ to account for parent rotation.

Node degree. It is normally defined in the presence of Boolean connectivity as the number of links that a node has to other nodes. In our case, we must take both soft connectivity and link directionality into account, so we define the incoming soft degree of node i as $\sum_{k\in\mathcal{N}} \pi_{k,i}$ and the outgoing soft degree as $\sum_{k\in\mathcal{N}} \pi_{i,k}$. We define the average node degree as the average of the incoming and the outgoing soft degree. The soft degree is an empirical estimate of the expected node degree in the connectivity graph.

2.2 Properties of low-power wireless links

Several studies have focused on the properties of low-power wireless links; early efforts have observed the non-monotonic decay of received power with distance [22]. In [11,20,23], it is shown that a *transitional reception region* separates a region with high PDR from a disconnected region. The extent of the transitional region varies dramatically depending on the environment, and it is particularly wide indoors, due to the larger impact of multipath fading. In [20], nodes in the transitional

area are shown to experience significant PDR fluctuations, and it is claimed that spread-spectrum usage may help with fading, but experimental data [24] with the CC2420 have shown that this is not the case in indoor environments. The average link quality in the transitional region decays smoothly, but its variance is huge, and distance is a poor predictor of connectivity (due to multipath fading and shadowing). Asymmetric links are shown to be common in the transitional region, where RSS fluctuations have a large impact on the delivery ratio.

Using a controlled experiment with motes (built around the CC2420) wired together through an attenuator, it is observed in [25] that no packets can be received below an SNR of 4dB, the noise floor is -96dBm, and there exists a 1.5dB transitional region (between -92 and -90.5 dBm) where links exhibit a PDR between 0 and 1; all links above -90dBm are shown to have a PDR of 1. Different nodes, however, have different noise floors, and an example is shown where the noise floors of a number of motes are scattered over 6dB (which corresponds to the RSS accuracy rating reported in the datasheet of the CC2420[1]). In non-controlled experiments, however, the width of the transitional region is shown to be 11dB; the reason for the extra 5dB is partly ascribed to an inherent bias due to the fact that RSS samples for lost packets cannot be measured. RSS is shown to be very stable over short intervals, though it experiences significant fluctuations over longer intervals. Such long-term fluctuations are due to changes in the topology of the deployment area that modify multipath patterns and induce fading [26].

These observations have raised the awareness that standard routing solutions are destined to fail in low-power WSNs. In [27], the ETX metric is proposed; the idea is to estimate the total number of transmissions needed to get a packet across a link, and use the route with the minimum ETX. For (i, j), the ETX is estimated

[1]The datasheet is available at `http://focus.ti.com/docs/prod/folders/print/cc2420.html`.

as $E_{(i,j)} \approx 1/(\pi_{i,j}\pi_{j,i})$. MintRoute, a routing protocol specifically designed for low-power WSNs, leverages on link estimation to find reliable routes by setting up a cost field based on ETX. ETX has been shown to be a robust metric, especially on top of an ARQ scheme [28]. The standard way of estimating ETX of (i, j) as $E_{(i,j)} \approx 1/(\pi_{i,j}\pi_{j,i})$ relies on the assumption that the reception over (i, j) is independent from the reception over (j, i). While this may be reasonable in MANETs due to mobility, it is not true in a typical WSN deployment where nodes are static: losses on the direct and reverse channel are correlated. In [19], where it is shown that RNP is a better metric than PDR for the definition of link quality (if the underlying routing protocols uses unconstrained retransmissions, RNP is the same as ETX), it is also observed that the inverse relationship between PDR and RNP that is commonly assumed does not hold in practice due to temporal correlations. It is therefore all the more crucial to use layer 2 ACKs to measure ETX, as done in [14], which proposes a hybrid link estimator that fuses the estimated broadcast beacon delivery rate with the Measured Number of Transmissions (MTX) of data traffic, measured by counting layer 2 ACKs.

Temporal variations have been the focus of several studies. It is shown in [29] that temporal variations of wireless links are not correlated with internode distance, but with link quality. Links in the transitional region, in particular, exhibit the most significant variations. It is shown in [25] that links in the transitional region tend to be bimodal, oscillating between high and low PDR; this implies that routing protocols should not just decide *where*, but also *when* to send: in [25] it is shown that delaying retransmissions by a fixed amount of time reduces the transmissions-to-deliveries ratio by 15%. Link bimodality also means that care must be exercised when making routing decisions based on periodic broadcast beacons: transitional links exhibit significant quality fluctuations.

2.3 A network layer for sensor networks

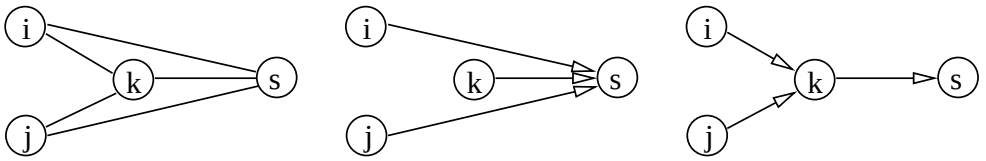
2.3.1 Long-hop and short-hop routing

For multihop wireless networks, a fundamental question is whether it is better to route over many short hops (short-hop routing) or over a smaller number of longer hops (long-hop routing). Short-hop routing has gained a lot of support, and its proponents mainly produce two arguments: reduced energy consumption and higher signal-to-noise-plus-interference ratios. Both arguments stem from a simplified analysis that is mainly based on the disc model [30]; in [5], we have shown 18 reasons why long-hop routing is, in most cases, a very competitive strategy. Given our focus on resource-constrained devices, the most interesting reasons why we advocate a small number of long hops are the reduction of energy consumption at the sensing nodes, a better load balancing, a more aggressive exploitation of radio sleep modes, and a reduced route maintenance overhead.

In any WSN, network lifetime (no matter how it is defined) is upper-bounded by the lifetime of the critical nodes. While radio activity is not the only factor that affects a node's lifetime, a notion of workload based on the total number of transmissions per delivered packet can be used as a lower bound of a node's energy consumption. The average workload W of the critical set $\mathcal{N}_1$ of a given sink assignment is defined as

$$W = \frac{1}{|\mathcal{N}_1|} \sum_{i \in \mathcal{N}_1} M_i(1 + \beta_i). \tag{2.1}$$

Figure 2.2 shows a simple example comparing the long-hop and the short-hop approach. In Figure 2.2(a), the connectivity graph of a sample network shows which node pairs can communicate (under the constraints of a given physical layer) at the maximum admissible transmit power of a particular transceiver. In Figures 2.2(b)-



(a) Connectivity graph, showing which node pairs can communicate given the hardware and the physical layer.

(b) Long-hop routing. The critical set average workload is given by $W_l = (M_i + M_j + M_k)/3$.

(c) Short-hop routing. The critical set average workload is given by $W_s = 3M_k$.

Figure 2.2. In this example, if $M_i + M_j < 8M_k$, long-hop routing is advantageous in terms of average critical set workload, which has a direct implication on network lifetime.

2.2(c) the arrows indicate child-parent relations as established by a routing protocol: all data packets are unicast from child to parent following the arrows. We assume a target offered load of 1 pkt/sec/node, and we assume that the children of the critical nodes are able to meet their target offered load, so that β_h can be replaced by the number of children of h for all $h \in \mathcal{N}_1$. With the long-hop approach shown in Figure 2.2(b), all nodes have zero relayed load ($\beta_h = 0$ for all $h \in \mathcal{N}$); therefore, the total work performed by the critical nodes (all nodes but s, in this case) is equal to $W_l = \sum_{h \in \mathcal{N}} M_h$. With the short-hop approach, a sample scenario is shown in Figure 2.2(c): $\beta_k = 2$ pkts/sec, $|\mathcal{N}_1| = 1$, and $W_s = 3M_k$. The long-hop approach is more energy-efficient if $W_l < W_s$, which is true if $M_i + M_j < 8M_k$. We are assuming that no duplicates from i and j are received by k; in the case of duplicates, the short-hop approach would be even less energy-efficient.

The rationale of a short-hop approach is to minimize the RNP network-wide, which often corresponds to a small $|\mathcal{N}_1|$: if a given node k chooses its best neighbor (the one with the best channel) as $p(k)$, then a subset of the sink neighbors may choose a parent other than the sink. As a result, a larger amount of load is injected into the network, and the relayed load in the critical area is necessarily larger.

In Figure 2.2, nodes i, j, and k are all sink neighbors; with the short-hop approach of Figure 2.2(c), only k communicates directly to s, and i and j leverage on k. With a long-hop approach, on the other hand, more sink neighbors communicate directly to the sink: in Figure 2.2(b), i, j, and k all do so. Long-hop routing yields a larger $|\mathcal{N}_1|$, but also a potentially larger M_h for all nodes $h \in \mathcal{N}_1$. If load balancing is effective, all β_h converge to a relatively narrow range of values, and no individual β_h stands out. Fewer nodes network-wide act as relays, and more nodes are in the critical set, which becomes less critical: nodes are never unnecessarily overloaded, and more nodes with sink connectivity share the burden of relaying data from upstream, thus allowing for a redistribution of the load. Since it is up to the link estimator to select links with a low RNP, long-hop routing is all the more effective if the routing protocol is complemented by a good link estimator.

2.3.2 Architectural considerations

Due to the countless applications of sensor networks [31], the design space for routing is large and diverse [32]. The significant resource constraints of low-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 [33]. Despite numerous research efforts, WSN routing remains to this day a fairly open issue. According to [34], the main obstacle to progress in the field of WSNs is primarily the lack of an overall architecture; a unifying abstraction, the Sensornet Protocol (SP), is advocated. The Internet's *narrow waist* is IP, which provides an abstraction of point-to-(multi)point best effort packet delivery. In WSNs, however, processing potentially occurs at every hop, which means that an overall architecture should be built around best-effort single-hop communication rather than an end-to-end route abstraction. The basic idea is to *lower the waistline*: the *narrow waist* of

the architecture should not be at the network layer, but between the network and the data link layer. The challenge of SP is that the network layer, albeit agnostic to the link layer, must cooperate with it to select the best possible neighbor for the next hop.

2.3.3 Control plane and data plane

The functions provided by a generic WSN network layer can be classified into *control plane* and *data plane* functions [35].

In the control plane, a *routing engine* is responsible for the computation of the next hop address in nodes other than the sink. The operation of the routing engine is enabled by a *routing topology component* responsible for the discovery and maintenance of the topology.

Data collection trees can be constructed in different ways by different topology modules, and over a given tree structure different next hop selection mechanisms, or routing engines, can be used.

The data plane can also be decomposed into a *forwarding engine* and an *output queue*: the former is responsible for unicasting a packet to the address provided by the routing engine, while the latter provides buffering and, possibly, scheduling functions.

2.4 A taxonomy of routing protocols for sensor networks

In this dissertation, we focus on solutions that have been shown to work under the constraints of the existing sensor network hardware. In particular, we consider protocols that have been implemented in TinyOS and shown to work on Berkeley motes.

Several classifications of WSN routing schemes are possible; the various categories that can be identified, however, are neither mutually exclusive nor collectively

exhaustive.

If geographic information is employed by the control plane, we speak of *geographic routing*. Protocols that do not employ geographic information, which is the case with Arbutus, follow a *cost-based* paradigm [36], whose central idea is the generation of a cost field rooted at the destination node. The cost field is set up as nodes estimate the cost of reaching the destination according to a given metric, *e.g.*, the distance in number of hops, and packets are routed to the destination through *reverse path routing* (data packets descend the cost field from the sensing area to the destination).

If routing decisions are made locally at each node, as in Arbutus, then the protocol is *distributed*, whereas if decisions are taken by a single node the protocol is said to be *centralized*. The one node that acts as a centralization point is typically a high-end node, in which case the network is heterogeneous. In heterogeneous multi-tiered architectures, clusters of low-end devices operate under the direction of a cluster head; if the centralized approach is adopted, each cluster head is used as the centralization point for the corresponding cluster.

Routing schemes are said to be *point-to-point* if they allow any node to route data to any other node. This *any-to-any* routing paradigm, however, is not needed in the largest class of sensor network applications, data collection, where a *collection tree* approach (*many-to-one* or *many-to-few*) is normally adopted; this is also the case with Arbutus.

A special category of data collection protocols follows a *data-centric* paradigm [37]: rather than collecting data from individual nodes (*address-centric*), a given kind of data is collected from a given region of the network. Arbutus is address-centric.

Protocols are said to be *sender-based* if nodes have specific parents to which

they unicast packets; this is the case with Arbutus. In *receiver-based* routing, a node estimates the cost of reaching the intended destination and includes it in the outgoing packet before broadcasting it. Only the neighbors that estimate a lower cost to the destination rebroadcast the packet, allowing it to descend a loop-free gradient towards the destination. The main problem with the receiver-based approach is the large overhead due to redundant forwarding [38].

Multipath routing schemes [39] distribute traffic over different paths (typically alternating the use of different paths at different times). Traffic can also be concurrently transmitted over separate paths for added reliability. Multipath routing also provides increased resilience to node failure: if multiple paths are maintained, alternate paths can be used in case the primary one fails. Given the energy cost of redundant transmissions, Arbutus does not follow a multipath approach.

Routing schemes can be classified based on when route selection is performed. In *proactive* protocols, routes are discovered before they are needed, whereas in *reactive* protocols routes are discovered on demand. In the case of cost-based data collection, one could argue that routing is reactive, as the sink initiates it by starting the set up of a cost field. One could also argue, however, that routing is proactive, because the nodes always know their next-hop neighbor thanks to periodic beaconing.

2.4.1 Proactive and reactive protocols

Proactive methods are typically *table-driven* and keep state in routing tables at each node [40]. In *distance-vector* schemes, based on the distributed Bellman-Ford algorithm [41], the routing table contains cost information regarding every other node in the network. Each node periodically broadcasts its routing table, so that its neighbors can update their own tables accordingly. In *link-state* protocols, based on Dijkstra's shortest-path algorithm [42], the routing table contains cost information

on the neighbors. Each node periodically sends its routing table to every other node in the network, typically through some form of flooding. While in distance-vector algorithms a node sends network-wide connectivity information only to its neighbors, in link-state schemes a node sends connectivity information about its neighbors to every other node. The common denominator is that each node computes the minimum cost path to every other node (generally the shortest path, with hop count as the cost function of choice). Link state protocols have a large route maintenance overhead (in terms of storage and communication), and the ad hoc network community has generally privileged distance-vector solutions, such as Destination-Sequenced Distance-Vector routing (DSDV), a distributed implementation of the Bellman-Ford algorithm with a special loop avoidance mechanism [40]. DSDV has been implemented in TinyOS for one destination and with the key modification of using link quality as a routing metric [43].

Reactive (on-demand) protocols emerged in the context of MANETs; the idea is to only set up routes as the need arises. Well-known examples are Ad hoc On-Demand Vector Routing (AODV) [44] and Dynamic Source Routing (DSR) [45]. The route discovery process in reactive schemes is a form of controlled flooding known as *route query and reply* [46]. In DSR, route discovery also relies on flooding and each route query packet contains the evolving source routing path: a node adds its address to the path before rebroadcasting the request and caches newly discovered routes [38]. This route caching mechanism identifies DSR as the reactive counterpart of link-state proactive schemes, in the sense that any node has access to one-hop connectivity information relative to (potentially) all other nodes. There exist TinyOS implementations of AODV (TinyAODV and MoteAODV), evaluated in [47] along with a TinyOS version of DSDV.

Collection tree protocols such as MintRoute and CTP employ a distance-vector

approach adapted to a many-to-one/few communication pattern: the routing table of each node contains state information about a subset of the node's neighbors, but only the cost of reaching the sink(s) gets advertised. Arbutus also uses a distance-vector approach, but does not employ a routing table, as each node only keeps track of its current parent.

One reactive protocol for ad hoc networks, AD-Hoc Multicast Routing (ADMR) [48], was recently implemented in TinyOS for CC2420-based motes [49] as TinyADMR. ADMR adds multiple-source support to the standard route query and reply process; data is multicast by sending packets to group addresses as opposed to node addresses. The use of ADMR on low-end sensor node hardware poses several challenges. In particular, the authors of [49] underscore the need for a reliability metric (simply using hop count leads to very low packet delivery rates), the limited amount of storage space for routing state, and the overhead due to control traffic. Like TinyADMR, Arbutus only keeps state for the current parent, and uses bottleneck quantities. (Tiny)ADMR, however, is point-to-point, cost-based, and, as the name says, employs multicast data transmissions, while Arbutus strictly uses unicasts for data traffic.

A centralized reactive protocol for multi-tiered WSNs, CentRoute [33], has also been implemented in TinyOS and tested in a mote testbed [50]. Another promising centralized scheme, the Flexible Control Protocol (FCP) [51], has been recently proposed for reliable data collection at ultra low duty cycles. FCP was also implemented in TinyOS and evaluated on a mote testbed. Differently from these schemes, Arbutus is fully distributed.

2.4.2 Data-centric routing

Directed Diffusion [52] is a family of cost-based data-centric protocols specifically developed for sensor networks. As opposed to transmitting sensor data from a given source to the sink, the idea is to transmit sensor data from a given area to the sink; in this sense, it is a data-centric (or *attribute-based*) protocol. It is irrelevant which node within the area senses the required data, as long as the request is matched. In the *one-phase pull* version of Directed Diffusion [53], the dissemination of *interests* is achieved by query flooding, and a cost field leading to the sink is set up. Multiple paths are found and route selection is operated on a per-node basis according to a given link metric. A TinyOS implementation of Directed Diffusion for low-end devices, TinyDiff, is available[2], but Directed Diffusion was designed for high-end sensor networks, such as networks of 32-bit microservers. Indeed, one-phase pull diffusion is used for the experimental results in [28] in a testbed of Stargate gateways with EmStar [54] (MICA2 motes are used as radio interfaces, but the processing is done in the Stargates), and the experimental comparisons between different diffusion schemes in [53] are performed on a network of high-end Sensoria WINSng 2.0 nodes. Address-centric tree-based protocols for mote networks such as MintRoute, CTP, and Arbutus also follow the cost-based one-phase pull approach of directed diffusion. A key difference is that, while in diffusion-based schemes nodes select their next-hop neighbor on the sole basis of local information, tree-based schemes for mote networks use a distance-vector approach. Arbutus, however, does not employ routing tables, and only keeps state for the current parent; in this sense, it is a hybrid between address-centric directed diffusion and distance-vector routing.

[2]For details, refer to `http://www.isi.edu/ilense/software/diffusion/index.html`.

2.4.3 Point-to-point routing

In point-to-point routing, next-hop selection is performed based on geographic information. Euclidean coordinates are employed in *geographic routing*. In a *greedy phase*, the neighbor that is closest to the destination is chosen as the next hop. To recover from local maxima and thus ensure that the destination is reached, many *face routing* schemes have been proposed [55]. As remarked in [56], however, most existing proposals are based on the disc model [30] and assume ideal radios with a circular range. Two face routing schemes that have been implemented in TinyOS and have been shown to work on motes are the Cross-Link Detection Protocol (CLDP) [57] and Lazy Cross-Link Removal (LCR) [55]. LCR and CLDP are essentially two different methods of building a topology; a routing engine must be associated with them to route over a given topology. While the routing engine used in the implementations of CLDP and LCR is Greedy Other Adaptive Face Routing [58], LCR and CLDP can also be used with other routing engines such as Greedy Perimeter Stateless Routing (GPSR) [59], which has also been implemented in TinyOS. A non-Euclidean notion of coordinates is employed in Beacon Vector Routing [60], where nodes use hop count as their notion of distance which is diffused by way of a cost field. BVR uses its own routing engine over a mesh topology with trees rooted at the nodes, and has been shown to work on motes.

Point-to-point schemes have a completely different purpose than many-to-one/few collection protocols; it is therefore not appropriate to use one of them as a benchmark for Arbutus. For a meaningful performance evaluation, we need to choose our benchmark in Arbutus's own subset of the design space.

2.4.4 Collection trees

The routing protocols most widely used by the TinyOS community are based on collection trees, and are sender-based and address-centric. MintRoute, MultiHopLQI, and Collection Tree Protocol (CTP) represent successive evolutions of a common cost-based paradigm defined in [11], which recognizes that the volatility of the wireless channel makes Boolean connectivity models not suitable for use in low-end sensor networks with low-power radios and limited resources. Link estimation is viewed as an essential tool for the computation of reliability-oriented route selection metrics. Routing is broken down into three major components: a link estimator that continuously assesses link quality, a routing engine that determines the address of the neighbor that provides the best progress toward the sink according to a given cost function based on the output of the link estimator, and a forwarding engine that injects its own traffic or relays upstream traffic by unicasting to the address determined by the routing engine. The rationale behind this partitioning is the separation of the data plane (forwarding engine) from the control plane (routing engine with the link estimator of choice). 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) uses routing tables and works with a link estimator based on the Expected Number of Transmissions (ETX) metric but can also employ packet delivery rate (PDR) estimates based on sequence numbers. MintRoute adopts a neighborhood management policy based on the FREQUENCY algorithm [61] as a link blacklisting solution; this keeps 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 LQI, 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, drastically reducing memory usage and control overhead. The most recent protocol in this family, CTP, 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. CTP also employs routing tables to allow nodes to quickly find a new parent upon parent loss. In parallel to the aforementioned protocols, a lightweight data collection protocol, Drain [62], 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-layer ACKs (also employed by MultiHopLQI). Drain, like MultiHopLQI, only keeps state for one parent. None of these protocols explicitly pursues load balancing.

2.5 Load balancing

Load balancing schemes have been proposed in the form of topology control, redundancy suppression, controlled mobility, and as part of lifetime-aware routing protocols; hybrid solutions across these categories also exist. Redundancy suppression may be used to enhance virtually any load balancing solution (for instance in the form of data aggregation). Siphon [63] is a hybrid between topology-control and routing (clustering-based routing). Siphon recognizes the challenge posed by the hot spot problem and proposes a multi-tiered solution (backed by experimental evidence on a mote testbed) based on the use of virtual sinks, special nodes equipped with a secondary radio that serve as local safety valves for overload traffic management.

Load balancing has been proposed as part of a geographic point-to-point routing protocol, curveball routing [64]. Starting with the observation that in a network with point-to-point routing the hot spot tends to be at the center of the deployment, the idea is to map the coordinates of the nodes to a spherical surface, and to

route greedily over it: on a spherical surface, the *crowded center* no longer exists. Experimental results obtained with a TinyOS implementation on a large testbed are shown.

As for schemes for many-to-one collection trees, Arbutus's network-layer load balancing is the only one that has been implemented and tested in mote networks. In [65] we have presented an early version of the architecture described in this dissertation.

2.6 Congestion control

Several congestion control schemes have been proposed, most of which follow a push-based paradigm whereby child nodes send their packets downstream and parent nodes request rate adaptation, either by way of hop-by-hop flow control, or through a source-limiting scheme. With hop-by-hop flow control, nodes signal local congestion to each other via backpressure signals, reducing packet loss rates and preventing the wasteful transmission of packets that are destined to be dropped downstream. The use of backpressure in the form of binary feedback was initially promoted as the Explicit Congestion Notification scheme in [66].

Hop-by-hop flow control can be achieved through channel sampling [67] (estimation of channel utilization by way of periodic sampling) or queue occupancy monitoring.

An example of source rate control is the token and bucket scheme: node i gets a token every time $p(i)$ has forwarded N packets, and each send costs 1 token, thereby setting the rate $f_i \leq f_{p(i)}/N$. MAC-level solutions also exist, such as adapting the CSMA backoff based on load, so that the higher a node's load, the shorter its backoff (backlogged nodes get prioritized) [68].

In [69], hop-by-hop flow control based on queue occupancy monitoring, rate-

limiting, and a prioritized MAC are combined into one scheme, Fusion, which is shown to be extremely effective. Experimental evidence on a large mote network is provided, but the interplay between congestion control and routing (Fusion is built on top of MintRoute) is not investigated.

A pull-based solution, PCP [3], has recently been proposed; i can only send to $p(i)$ if $p(i)$ provides a grant-to-send authorization. PCP eliminates the cause for ingress drops, but does so at the price of a large control overhead. The rationale behind its approach is that, in the push-based paradigm, any form of congestion control is a post-facto measure, whereas the pull-based paradigm operates preemptively.

In [70], a multipath geographic routing scheme, Biased Geographical Routing (BGR), is used along with two congestion control techniques: In-Network Packet Scatter, which redistributes traffic to avoid congested hot spots, and End-to-End Packet Scatter, a more aggressive scheme that operates source rate control. Experimental results using a TinyOS implementation of BGR on a mote testbed are shown (without the two congestion control schemes, though).

CHAPTER 3

ARBUTUS

3.1 Generalities

Arbutus is a novel routing architecture specifically designed for low-end homogeneous wireless sensor networks. In terms of the taxonomy presented in 2.4, Arbutus employs a collection tree approach and is address-centric and sender-based. Arbutus builds on the cost-based paradigm promoted in [11]; significant innovations are in both the control plane and the data plane. In the former, a load balancing scheme and a link estimator are embedded into the routing engine, and the topology module employs a novel tree construction scheme to enforce long-hop routing. In the data plane, several performance-enhancing features are implemented, most significantly a novel congestion control scheme.

In terms of routing performance, Arbutus's foremost goal is reliability. All packets from all nodes are assumed to be equally important, and Arbutus employs unconstrained retransmissions (as in PCP [3]) to ensure that every effort is made toward the delivery of every generated packet. At the same time, Arbutus adopts a best-effort approach to maximize the network goodput under the constraint of reliable delivery. Finally, Arbutus seeks to achieve energy-efficiency in terms of both relayed load (number of relayed packets per generated packet) and routing cost (number of transmissions per successfully delivered packet).

Though delivery reliability, goodput, and energy-efficiency are partly contrasting

goals, Arbutus leverages on the subtle interplay among these factors. Load balancing and congestion control, for one, are strongly related. The goal of load balancing is lifetime extension by preventing node overload. Congestion control aims at preventing ingress drops, but buffers overflow if nodes are overloaded, which is the very condition that load balancing tries to prevent. Therefore, the lack of a load balancing policy makes congestion (and packet loss) more likely.

Load balancing is achieved through parent rotation, which occasionally requires the use of suboptimal links. The typical argument against load balancing is that the best links should be employed as long as they are available; routing should be performed over suboptimal links only when there is no other option. Our main argument in favor of load balancing is congestion avoidance, but goodput and routing cost (and therefore lifetime) are also valid arguments. If a node is overused, it means that its descendants are contending for the same channel. This may cause a degradation of the goodput due to MAC backoffs (at least with CSMA schemes), and certainly causes packet loss due to interference: if a large number of nodes are contending for the channel, no MAC layer can avoid packet loss. This form of packet loss can be compensated for with retransmissions, but retransmissions increase the routing cost, not to mention that ACKs can also get dropped, injecting duplicates and aggravating overload and congestion. Since the idea behind using the best links is indeed to minimize routing cost and maximize goodput and delivery rate, it should now be clear that load balancing, goodput, and reliability are not necessarily contrasting goals, and indeed we will show that Arbutus can achieve all three goals within the constraints imposed by the network topology.

Arbutus can be broken down into a control plane and a data plane. The control plane is responsible for the setup of a cost field across the network. The cost field setup is initiated by the sink and continued by the other nodes as they receive con-

trol beacons from their downstream neighbors. Differently from other cost-based protocols, Arbutus employs a double cost field: an outer field for depth control, in order to avoid unnecessarily long routes, and an inner field based on CSI. State propagation leading to the setup of the double cost field is achieved through broadcast control beacons. A node uses beacons received from its neighbors to estimate its depth, *i.e.*, its hop distance from the sink, which is used as the notion of cost for the outer field. Arbutus is cost-based, and nodes do not have access to any kind of geographic information: they have no notion of Euclidean distance or position. Beacons from nodes advertising a lower depth are considered for further processing, while beacons from all other nodes are discarded, as they would lead to overly lengthy routes. Beacons considered for further processing are then passed through the inner field, which uses CSI and load estimates to award parent status to the node that provides the best compromise between reliability and load balancing.

The main drawback of this approach is that it relies exclusively on the uplink from prospective parents to children. Node i judges its downstream neighbor j based on CSI pertaining to (j, i), but its goal is to transmit over (i, j). This is why the cost estimates of both fields are corrected by a form of feedback from the data plane, which has access to information about the downlink in the form of ACK packets sent by the parent. If the CSI over (j, i) looks acceptable but the RNP is large $(M_i \gg 1)$, then the CSI estimates are corrected accordingly to facilitate the election of a new parent. Likewise, for the outer field, the depth estimate is adjusted so that same-depth nodes can also be considered for parent status if M_i is large enough.

Incoming data packets from upstream nodes enter the data plane, which buffers them along with the packets generated locally. The data plane unicasts all data packets to $p(i)$, the current parent determined by the control plane based on the cost fields. The control plane maintains the double cost field through periodic beacons

(synchronous beaconing). Asynchronous beacons are also sent in response to special requests of the data plane, put forth upon detection of abnormal conditions such as route breakage due to the formation of a routing loop or a local congestion state.

Arbutus runs at every node in the network in a distributed fashion. We now provide a detailed description of the operation of the data plane and the control plane.

3.2 Data plane

The data plane performs the basic functions of data packet generation and forwarding, which it complements with several performance-enhancing services: duplicate suppression, congestion control, routing loop detection, data packet buffering, and load monitoring.

Unconstrained retransmissions. Locally generated packets and packets received from upstream neighbors are queued in a FIFO buffer. Packet generation and FIFO buffer servicing leading to the unicast transmission of a data packet to $p(i)$ are only performed if $p(i) \neq z$. After a packet is sent to $p(i)$, if a positive layer 2 ACK is received, the FIFO buffer is serviced again (if not empty). A maximum of $N_{\max}$ retransmissions are performed at fixed intervals T_{r}; if they do not suffice to receive a positive ACK, more retransmissions are performed at increasing intervals $T_{\mathrm{v}} = g(M_i)$, where $g(x)$ is an increasing function of x. Arbutus performs unconstrained retransmissions: each packet is equally important and we do not wish to discard any particular packet to favor others. Note that the parent may of course change, in which case M_i is reset to be fair to the new parent, as in CTP (a new parent starts off with a clean slate and is not penalized for the shortcomings of the previous parent).

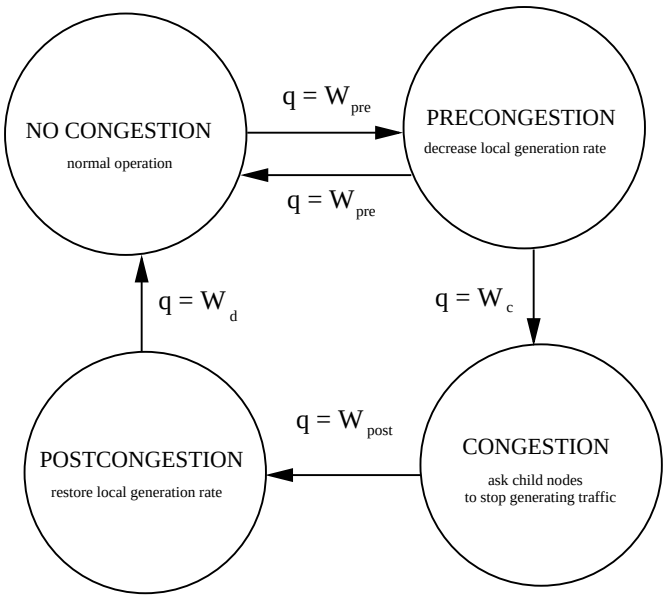


Figure 3.1. Selfless Up Selfish Down congestion control (SUSD): structure of the finite state machine.

Duplicate packet suppression. A unique instance of a packet is defined based on its origin address (address of its generator) and its sequence number set by its generator. Duplicate versions of packets may be the consequence of failed layer 2 ACKs (a received packet is not acknowledged). More often, however, duplicates are sent by i over (i, j) if the ACK from j gets dropped over (j, i); this happens if (i, j) is asymmetric with $\pi_{i,j} > \pi_{j,i}$. Arbutus keeps duplicates from entering the data buffer by checking each packet's sequence number versus the entry relative to the packet's generator in a look-up table. If a given packet from a node j received at i is determined not to be a duplicate, the entry of the look-up table pertaining to j is updated with the packet's sequence number. This simple mechanism avoids the useless and energy-inefficient propagation of duplicates at the price of extra RAM usage.

Congestion control. Given that the use of unconstrained retransmissions virtually eliminates channel-related packet loss, other than those induced by false positive ACKs or faulty CRCs, the other major factor that can cause packet loss is congestion. To mitigate it, Arbutus employs a finite state machine with hysteresis to send backpressure signals [66]. We define four thresholds corresponding to four different levels of FIFO buffer occupancy to be signaled: a pre-congestion threshold W_{pre}, a congestion threshold $W_{\text{c}} > W_{\text{pre}}$, a post-congestion threshold $W_{\text{post}} < W_{\text{pre}}$, and a decongestion threshold $0 \leq W_{\text{d}} < W_{\text{post}}$. The fraction of FIFO occupancy q is fed into the state machine after each access to the FIFO buffer. The hysteretic nature of the mechanism implies that, once congestion is detected, FIFO buffer occupancy must go below pre-congestion levels.

Rather than asking descendants to lower their rate, a node whose FIFO occupancy reaches the pre-congestion threshold ($q = W_{\text{pre}}$) starts by slowing down its own generation rate. If FIFO occupancy reaches the congestion threshold ($q = W_{\text{c}} > W_{\text{pre}}$), however, the child nodes are asked to stop sending. Asking them to slow down may not suffice, since the control beacons may not be heard by all child nodes (the medium is highly congested). It is therefore much more effective to demand that they all stop; the goodput may suffer, but Arbutus's primary goal is reliability. In the congested state, a parent stops child nodes from pushing data traffic by advertising route breakage as often as desired (in our implementation, we use specific values of $q > W_{\text{c}}$ to trigger a route breakage advertisement).

As congestion falls below a post-congestion threshold ($q = W_{\text{post}}$), the previously congested node adopts a selfish approach and restores its normal generation rate. It does not allow its descendants to resume unicasting until $q = W_{\text{d}}$. All communication to the descendants is performed with beacons managed by the control plane. We refer to this scheme as SUSD (Selfless Up, Selfish Down); the state machine is

Table 3.1

STATE INFORMATION AT NODE i.

State variable	Symbol	In beacon
address	i	yes
current parent address	$p(i)$	no
former parent address	$\tilde{p}(i)$	no
advertised parent address	$p_a(i)$	yes
advertised former parent address	$\tilde{p}_a(i)$	yes
hop distance to sink	H_i	yes
bottleneck link RSS	Λ_i^{RSS}	yes
bottleneck link LQI	Λ_i^{LQI}	yes
bottleneck relayed load	B_i	yes
local relayed load estimate	β_i	no
cost of reaching the sink by way of $p(i)$	C_i	no
MTX to parent	M_i	no

shown in Figure 3.1. Our implementation uses $W_{\mathrm{pre}} = 0.5$, $W_{\mathrm{c}} = 0.85$, $W_{\mathrm{post}} = 0.25$, $W_{\mathrm{d}} = 0$: node i lowers its transmission rate if $q = 0.5$, enters the congested state as $q = 0.85$, restores its original rate as $q = 0.25$, and does not signal decongestion until $q = 0$. These values were determined empirically, based on experimental results.

Routing loop detection. If loops arise, it is up to the data plane to detect them and break them. A loop is detected if i receives a packet generated by i itself. Upon loop detection, duplicate suppression is disabled to avoid dropping packets in the loop (ingress drops). The control plane is then instructed to advertise route breakage.

Load monitoring. The data plane is also responsible for keeping track of the relayed load, which is fed into the control plane where, as we will see, it is used in the inner cost field to achieve network-layer load balancing.

3.3 Control plane

At node i, Arbutus only keeps state for the current parent $p(i)$; the identity of the latest parent prior to the adoption of $p(i)$ is also stored and denoted by $\tilde{p}(i)$. The addresses of the current and the former parent are normally advertised as part of the control beacons, but there are cases in which the advertised parent and the advertised former parent differ from the current and former parent. We denote the advertised parent as $p_a(i)$, and the advertised former parent as $\tilde{p}_a(i)$. Two kinds of beacons are employed: route beacons advertising a valid route (the advertised parent address is $p_a(i) = p(i) \neq z$), and no-route beacons advertising route loss ($p_a(i) = p(i) = z$). The former build up the double cost field, whereas the latter destroy it in the presence of abnormal conditions that would make it inconsistent. Beaconing is normally performed at regular intervals $T_b = 1/f_b$. In addition, asynchronous beacons may be broadcast if pathological behaviors are detected. Table 3.1 shows the most important state variables and indicates which ones are contained in the broadcast control beacons. We define a *link bottleneck function*

$$\Lambda_i^Y \triangleq \max_{r \in [i,s]_{p(i)}} Y_{r,p(r)}, \tag{3.1}$$

where r denotes a relay in the current route from i to s (over $p(i)$), and Y may represent either RSS or LQI. We also define a *load bottleneck function*

$$B_i \triangleq \max_{r \in [i,s]_{p(i)}} \beta_r. \tag{3.2}$$

3.3.1 DUCHY: a DoUble Cost field HYbrid link estimator

A modified version of DUCHY, our hybrid link estimator presented in [71], is embedded into the control plane. RSS is used to obtain soft information about good

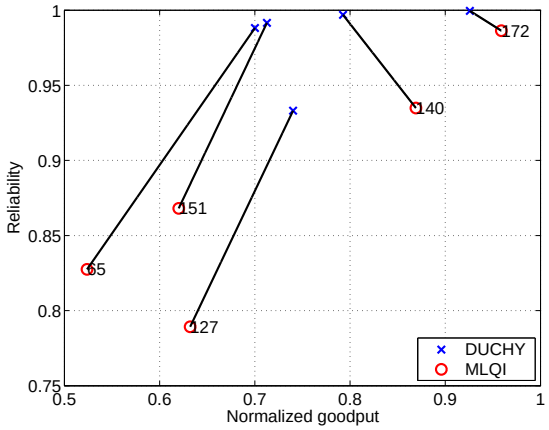


Figure 3.2. Goooodput-reliability performance.

links, LQI is used to get soft information about bad links, and we borrow from [14] by using data plane feedback (layer 2 ACKs for unicast outgoing data packets). Our estimator is hybrid in two different ways: it is driven by both CSI and PDR estimates, and it leverages on both broadcast control traffic and data traffic (like [11] and [14]). The design of the link estimator is specific to 802.15.4 (it uses LQI and link-level ACKs), but it can be adapted to non-802.15.4 stacks.

Any beacon from node b advertising $p(b) \neq z$ is passed through the double cost field. For the outer field, depth information is used, along with feedback from the data plane. In the inner field, the link estimator employs CSI measurements performed on received control beacons, also refined with data plane feedback.

In this subsection, we report the results of the experimental evaluation of DUCHY (as published in [71]) without load balancing (the version of DUCHY integrated in Arbutus employs network-layer load balancing).

In this evaluation, every node generates data packets at the rate $f_{\text{gen}} = 1\text{pkt/sec}$ with the goal of having them delivered to the sink. As a benchmark, just for this subsection, we choose a widely used table-free protocol based on MintRoute [11],

MultiHopLQI, which only employs LQI for link estimation and implements a maximum of 5 retransmissions; we will refer to MultiHopLQI's link estimation scheme as MLQI. Note that, while our protocol is optimized for reliability (infinite retransmissions), MultiHopLQI is optimized for goodput (it performs 5 retransmissions before moving on). Using MoteLab [9], a public testbed of TMote Sky nodes, we ran 10 experiments of variable duration (between 5 and 15 minutes) at 5 different sink assignments; in this dissertation we show the average results for each sink assignment. For ease of reference, the sink assignments are 65, 127, 140, 151, and 172. The network size varied between 10 and 35, depending on the sink assignment; the total number of functioning nodes at the time of the experiments (February 2008) was, on average, about 50 (in disjoint clusters, due to the particular connectivity conditions of the testbed at that time). We ran DUCHY and MLQI back-to-back, and only considered the outcome if the number of reachable nodes was the same for both runs (*i.e.*, no nodes ceased to work during either run).

Figure 3.2 shows the reliability-goodput performance of DUCHY and MLQI. As expected, using DUCHY with infinite retransmissions results in a superior reliability performance. DUCHY may also outperform MLQI in terms of goodput; this happens for sink assignments 65, 127, 151. Since changing the sink assignment means dealing with a different network, we characterize a given topology using two quantities: the network weight, defined as $\sum_{i \in \mathcal{N}} H_i$, and the critical set size.

Figure 3.3 shows where each sink assignment stands in terms of these newly defined topology indicators. We immediately note that sink assignments 65, 127, 151 correspond to a relatively large network weight and a small critical set; we call these *high-pressure topologies*, since the sink only has a few neighbors that relay the load from many other upstream nodes. In particular, sink assignment 127 only has one critical node. On the contrary, sink assignments 140 and 172

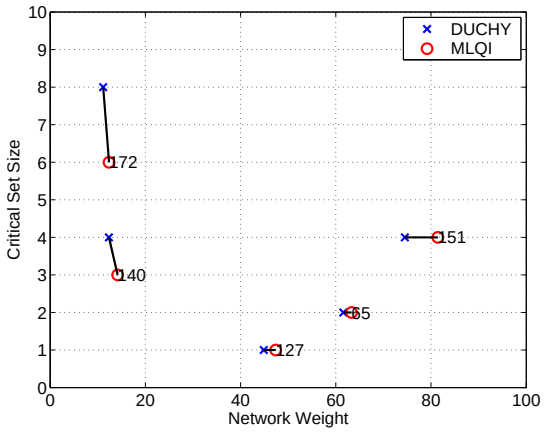


Figure 3.3. Quantitative characterization of the topologies used in the experiments.

are *low-pressure topologies*: the sink has a large number of neighbors that relay the load from a few other nodes. It should be noted that DUCHY never yields a smaller critical set than MLQI; a larger critical set becomes less critical (pressure is lowered) and inherently promotes load balancing. Going back to Figure 3.2, DUCHY completely outperforms MLQI (in terms of both reliability and goodput) in the three high-pressure cases, while in low-pressure topologies MLQI yields a slightly better goodput. DUCHY's outer cost field eliminates unnecessarily long routes, while its inner field selects the best links among those that made it through the outer field. For instance, if the choice is between a two-hop route over lossless links or a one-hop route over a link (i, j) with $M_i < 2$, DUCHY chooses the latter, as the former does not pass through the outer field. This way, DUCHY provides Arbutus with all the benefits of a lower hop count.

DUCHY's effectiveness is evident in Figure 3.4, which shows that, despite the infinite retransmissions implemented by Arbutus, DUCHY does not significantly increase the average number of transmissions per delivered packet in high-pressure topologies and reduces it in low-pressure ones. Moreover, DUCHY consistently yields shallower trees. Over all our experiments, DUCHY outperforms MLQI in

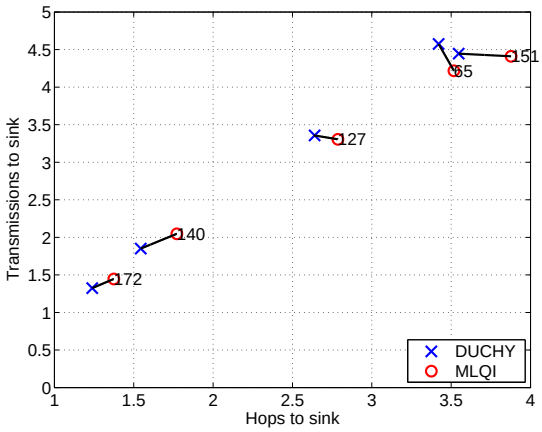


Figure 3.4. Average number of transmissions needed for delivery to s as a function of the average hop count.

terms of reliability in 96% of the nodes. The goodput is slightly worse in 20% of the nodes, and better in 60% of them.

3.3.2 Routing engine

Outer cost field. The primary purpose of the outer cost field is to limit the depth of the routing tree. At node i, the outer field filters out neighbors that are estimated to be at the same hop distance from s as i. The outer cost coincides with the corrected distance to the sink $A_i \triangleq H_i + y(M_i)$, where $y(x)$ is a non-decreasing function of x. If a large M_i is measured over $(i, p(i))$ upon data unicasting, then the outer cost is increased by the $y(M_i)$ term in order to enable beacons from a prospective parent b that advertises $H_b < A_i$ to pass through the outer field. RNP is measured based on layer 2 ACKs, whose number is passed through a low-pass filter that gets reset at every parent change.

This mechanism is very effective to minimize tree depth without suffering from asymmetric links. Suppose node i infers its hop distance to the sink to be $H_i = d$ and sets $p(i) = j$; evidently, $H_j = d - 1$. Now, suppose that (j, i) is an asymmetric

link and $\pi_{i,j} < \pi_{j,i}$. What happens is that i has no problem receiving j's beacons, but j has a hard time receiving i's data packets. In other words, (i, j) suffers from a very high RNP, which becomes a serious problem if i has no other neighbors at a hop distance of $d-1$ from the sink: without the RNP correction, i will not process beacons from nodes at depth d, and it will need, on average, a disproportionately large number of retransmissions to get its packets to j.

The presence of the outer cost field makes it harder for loops to form: neighbors deeper in the tree are discarded, and same-depth neighbors are only awarded parent status if the RNP over their links to the current parent becomes too large.

Inner cost field. If a beacon makes it through the outer field, the link estimator marshals the relevant argument for the computation of a routing cost to be associated with b, the beacon's sender. The cost C_b of using b to reach s, *i.e.*, the cost of $[i, s]_b$, is computed by i as a function of the normalized RSS and LQI measured over (b, i) upon beacon reception (respectively, $\mathrm{RSS}_{b,i}$ and $\mathrm{LQI}_{b,i}$) and the RSS, LQI, and load bottlenecks advertised by b (respectively, Λ_b^{RSS}, Λ_b^{LQI}, and B_b). In our implementation, we employ

$$C_b = \mathrm{RSS}_{b,i} + \mathrm{LQI}_{b,i} + \Lambda_b^{\mathrm{RSS}} + \Lambda_b^{\mathrm{LQI}} + B_b. \tag{3.3}$$

Accounting for the load bottleneck embeds load balancing into the routing protocol, as prospective parents with a large relayed load are penalized even if they advertise a reliable route. Node b is awarded parent status if $C_b + 1 < C_{p(i)} + g(M_i)$, where $g(x)$ is an increasing function of x. The RNP correcting factor is, again, used to avoid asymmetric links; as previously remarked, all the CSI derived from the beacons pertain to (b, i), but data must go over (i, b).

State management upon parent update. A key point is that no routing table is employed: state is only maintained for the current parent $p(i)$ and the former parent $\tilde{p}(i)$ (which is used for congestion management, as we will later describe). Changes in the state variables upon adoption of a new parent include the update of the hop count to the sink ($H_i \triangleq H_{p(i)}+1$) and the update of load, RSS, and LQI downstream bottlenecks.

Parent unavailability. If a parent becomes unavailable for any reason, its child nodes no longer receive its beacons. Using a timeout value to infer parent loss would add an unnecessary and harmful delay to parent rotation, so it is preferable not to take action and let the network mend itself: the data plane keeps unicasting to $p(i)$, the RNP blows up, so does A_i, and as soon as a beacon from a different neighbor j with $H_j \leq A_i$ is received, j is awarded parent status. Whatever the cost of using j, it will be exceeded by the cost of using $p(i)$, also blown up by the RNP correcting term.

Congestion and loop recovery. In the case of local congestion as detected by SUSD in the data plane, node j broadcasts an asynchronous no-route beacon with $p_a(j) = z$ and $\tilde{p}_a(j) = z$: j gets its descendants to stop sending data packets by advertising an imaginary parent loss (but $p(i) \neq z$). In the case of a routing loop or downstream congestion, a route disruption of undetermined duration is inferred; therefore, node j sets $\tilde{p}(j) = p(j)$ and $p(j) = z$, and broadcasts an asynchronous no-route beacon with $p_a(j) = z$ and $\tilde{p}_a(j) = \tilde{p}(j)$. The idea is to warn neighbors about the abnormal conditions at $\tilde{p}(j)$: in the case of a loop or local congestion at $\tilde{p}(j)$ (seen by j as downstream congestion), $\tilde{p}(j)$ should be avoided. Upon reception of a no-route beacon from j with $\tilde{p}_a(j) \neq z$, node i infers a parent loss either if $j = p(i)$ (its own

parent advertising route breakage) or if $j \neq p(i)$ and $p(i) = \tilde{p}_a(j)$.

No-route beacons. Upon reception of a no-route beacon, no action is taken if $p(i) = z$. In case that $p(i) \neq z$, a no-route beacon from the current parent $p(i)$ is a crucial event. If the beacon advertises $\tilde{p}_a(p(i)) = z$, the parent $p(i)$ is *pinned*: its identity is stored and no other node is awarded parent status until the current $p(i)$ becomes available again. This is done to prevent routing loops: $\tilde{p}_a(p(i)) = z$ means that $p(i)$ is experiencing local congestion and will become available again once congestion is over. An asynchronous no-route beacon is broadcast with $p_a(i) = z$ and $\tilde{p}_a(i) = f(i)$. The identity of the lost parent, $\tilde{p}(i)$, is advertised in case some common neighbors of i and $\tilde{p}(i)$ are descendants of $\tilde{p}(i)$ but missed the no-route beacon broadcast by $\tilde{p}(i)$.

CHAPTER 4

EXPERIMENTAL ROUTING ANALYSIS

4.1 Testbeds

We perform a comprehensive routing evaluation and benchmarking on large-scale testbeds. For many different reasons, the number of working nodes in a given testbed fluctuates over time: hardware may fail, software issues may occur, and human activity may disturb the nodes. As a reference, we provide the number of working nodes in each testbed that we use averaged over the timeframe of usage.

Most of the experimental work for this paper was carried out on the MoteLab testbed over several months. All the data presented herein pertains to two particular timeframes of testbed usage: March 2008, when the average number was 90, and April-May 2008, when the average number of working nodes was 155. The significant increase in the number of working nodes occurred due to a major overhaul performed by the maintainers of the testbed.

MoteLab nodes are distributed over 3 floors. The total deployment area is about 6500 square meters, which corresponds to a relatively low average density of approximately 0.025 nodes per square meter. Node placement is very irregular, and the node degree distribution has a very high variance.

We also present results obtained on the Telecommunication Networks Group Wireless Indoor Sensor network Testbed (Twist) at Technische Universität Berlin, and on the Tutornet testbed at the University of Southern California. In terms of

node placement regularity, Twist is at the other end of the spectrum with respect to MoteLab, as its 100 nodes are arranged in a fairly regular grid over 3 floors; each floor covers an area of 450 square meters, and the average density is a relatively high 0.075 square meters, 3 times the density of MoteLab.

Tutornet contains 91 TMoteSky nodes distributed over sections of 2 floors of 1400 square meters each, with an average density of about 0.065 nodes per square meter.

Using all three testbeds provides us with topological diversity: we have the two extremes of MoteLab's large scale and low-average variable density and Twist's regularity and high density, and Tutornet's relatively high-average variable density. Most of our work, however, is on MoteLab, due to its public nature.

4.2 Methodology

We assume a many-to-one application scenario where each node generates traffic at a fixed target rate f_{gen} (the target offered load, equal for all nodes) destined to a single sink node. It is assumed that all nodes are equal and all packets have the same importance: there is no benefit in dropping a given packet to favor another one. Due to the presence of a congestion control mechanism, the actual offered load may be lower than the target offered load.

In general, each experiment has two main parameters: the sink assignment and the target offered load. For each MoteLab data point, we typically present results obtained as an average over 3 experimental runs of a duration ranging from 10 to 30 minutes (most runs last 15 minutes). Experiments with Arbutus and the benchmark, CTP, are run back-to-back. Network-wide results are obtained by averaging over all nodes. For Twist we typically present results obtained as an average over 1-2 runs of an average 15-20 minutes. On Tutornet we run fewer but longer experiments,

typically 1 run per data point, ranging from 30 minutes to a few hours.

4.3 CTP as a benchmark

CTP falls into the same subset of the design space as Arbutus. The routing engine of its control plane also uses periodic beaconing to set up a cost field based on ETX, and employs a routing table for parent selection. The link estimator combines an ETX estimate based on the control traffic with data plane feedback in the form of MTX; the combined estimate is then passed through an exponentially weighted moving average filter. The data plane provides buffering, loop detection, and congestion detection. There are two levels of buffering: a message pool and a forwarding queue. The message pool concept comes from the SP architecture; the idea is that multiple network protocols might coexist and need to exchange information with the link layer. While the message pool concept is certainly applicable to Arbutus, we limit ourselves to using a forwarding queue of the same size as CTP's. Loop detection in CTP is based on cost values advertised in the header of the data packets: if $C_{p(i)} \geq C_i$ (where C is an estimate of the ETX), then a loop is detected. Congestion detection is implemented using a congestion bit that, if set, forces a child node to award parent status on a different entry of the routing table. Based on layer 2 ACKs, the data plane performs constrained retransmissions. All these features indicate that CTP privileges goodput over reliability, while the opposite is true for Arbutus.

4.4 Performance metrics

Delivery ratio. Since Arbutus is optimized for reliability, the primary performance indicator is the delivery ratio, whose average for the network is computed as the total number of delivered packets over the total number of sent packets. Since CTP

is optimized for goodput, it should only be considered a baseline in terms of delivery ratio.

Goodput. It is defined as the number of successfully delivered packets at the sink per time unit (we measure it in packets per second). Delivery ratio and goodput provide non-overlapping information: a protocol can achieve a relatively large goodput at the cost of a low average delivery rate (for instance, by adopting a best-effort approach and injecting more packets than can be accommodated), or, conversely, it can achieve a very high delivery rate at the cost of a goodput degradation and a larger delay (for example, through the use of unconstrained transmissions). Since CTP is optimized for goodput, it is an excellent benchmark for Arbutus in this dimension. It is reasonable to expect CTP to outperform Arbutus under light load, given Arbutus's use of unconstrained retransmissions over potentially suboptimal links (for purposes of load balancing).

Coverage. We define coverage with respect to target values, which are fractions ρ of the offered load. A node is said to be covered with respect to a target value ρ if its contribution to the goodput at the sink is no less than a fraction ρ of its offered load. As an example, if $\rho = 0.75$ and the offered load is 1 pkt/sec/node, then a node is covered with respect to ρ if it is able to contribute a goodput of no less than 0.75 pkts/sec at the sink. Note that this notion of coverage is agnostic to the delivery ratio, as what matters is only the number of packets delivered to the sink per time unit. In other words, in our numerical example, a node that delivers 0.8 pkts/sec and loses 0.2 pkts/sec is considered as covered, while a node that delivers 0.7 pkts/sec with no packet loss (thanks to the use of retransmissions) is considered as not covered.

Fairness. Fairness is a widely used metric in the congestion control literature [69]. It is usually applied to delivery rates according to Jain's equation [72]:

$$\phi \triangleq \frac{(\sum_{i \in \mathcal{N}} e_i)^2}{|\mathcal{N}| \sum_{i \in \mathcal{N}} e_i^2}, \tag{4.1}$$

where e_i is the delivery ratio of node i measured at s. While coverage is agnostic to the delivery ratio and only considers the goodput, fairness is agnostic to the goodput, and only considers delivery ratios.

Routing cost. We define routing cost as the average of the total number of transmissions needed to deliver a packet from any node to the sink. Ideally, the total number of transmissions should coincide with the hop distance of each node to the sink, but this is not the case due to the lossy nature of wireless links and the consequent need for retransmissions. Arbutus's use of suboptimal links may increase the routing cost, and its unconstrained retransmission policy may cause routing cost spikes. In this dimension, CTP is again an excellent benchmark.

4.5 Relative impact of the various features

To better grasp the relative importance of the various features of Arbutus, Figure 4.1 shows the performance of Arbutus in terms of goodput and packet loss for a particular sink assignment in MoteLab. The feature that matters the least is definitely the RNP correction on the inner field link cost metrics (inner field feedback), whose absence hardly modifies the overall performance. Interestingly, taking the load balancing correcting term away from the inner field leads to increased packet loss but does not affect the goodput, confirming that load balancing prevents congestion. One-shot beaconing means that a single round of beacons is used to bootstrap the inner field, which is never updated other than by the RNP feedback term; as one

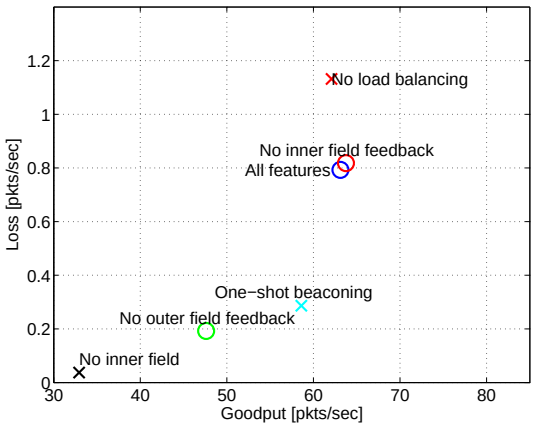
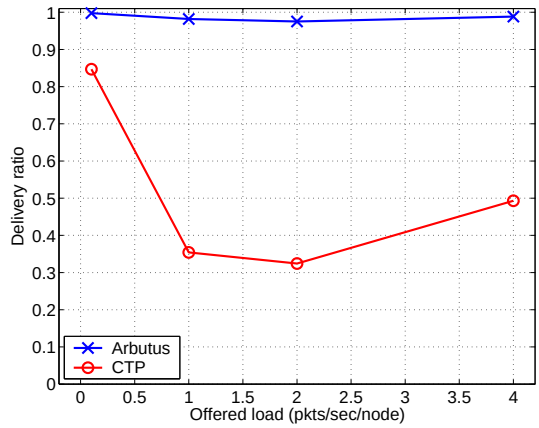


Figure 4.1. MoteLab, 90 nodes. Relative impact of some of Arbutus's features on the performance with a target offered load of 1 pkt/sec per node. Note that at this level of offered load, a value of 0.9 on the vertical axis maps to a 1% packet loss.

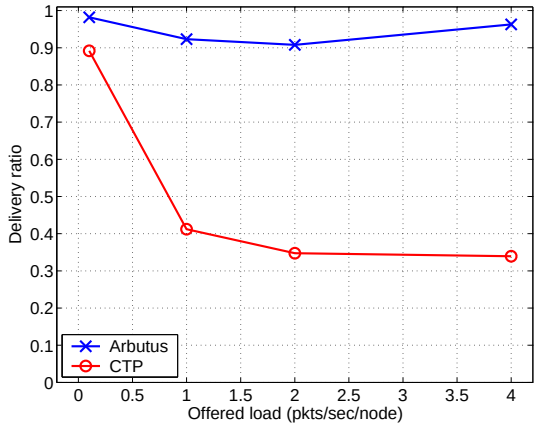
could expect, one-shot beaconing leads to a considerable routing inefficiency which translates into a noticeably lower goodput. The goodput degradation, however, is compensated by a reliability increase; this is because beaconing provides CSI on the uplinks, but data packets are routed over the downlinks. In other words, by foregoing the use of control beacons other than in the bootstrapping phase, elusive positive fading states on transitional links are not exploited: it is therefore natural to obtain a goodput loss compensated by a small reliability improvement.

A similar behavior is obtained by removing outer field feedback: now we are only using depth information based on beacons, which leads to a very rigid routing structure. Routing loops are not possible as same-depth nodes are altogether avoided, congestion is compensated by load balancing, but it takes, on average, more transmissions to get each packet across because the CSI estimates are not corrected with feedback from the data plane: again, we improve reliability but lose goodput, and the loss is more noticeable than the gain. Eliminating the inner field accentuates the behavior observed with one-shot beaconing: only links with a small RNP are

used, which optimizes the delivery ratio because congestion is less likely to occur. On the other hand, goodput suffers greatly because the system does not leverage on transitional links.



(a) MoteLab, 90 nodes.

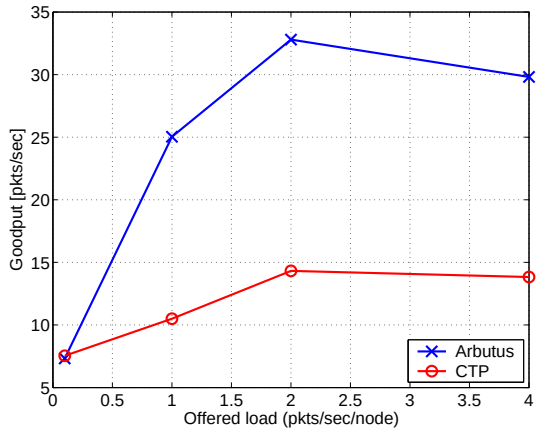


(b) MoteLab, 155 nodes.

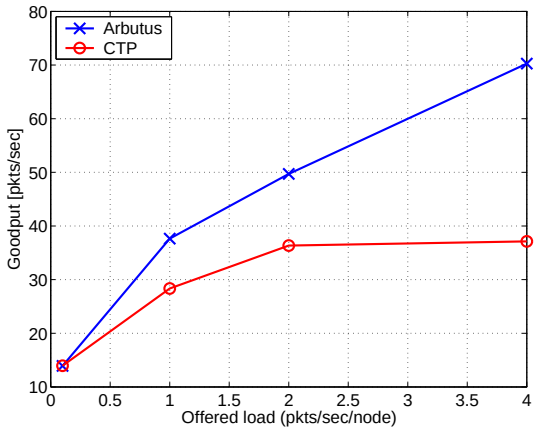
Figure 4.2. Delivery ratio with Arbutus and CTP at four different levels of offered load.

4.6 Performance as a function of the target offered load

Basic performance evaluation. For a basic performance evaluation, we arbitrarily fix the sink assignment in the 155-node MoteLab testbed (node 151) and vary the



(a) MoteLab, 90 nodes.



(b) MoteLab, 155 nodes.

Figure 4.3. Goodput with Arbutus and CTP at four different levels of offered load.

target offered load. For a given value of the offered load, we plot the average of a given performance indicator over 5 experiments performed within a few hours of each other. We consider four different offered load levels: $f_{\text{gen}} = 0.1$pkts/sec, $f_{\text{gen}} = 1$pkt/sec, $f_{\text{gen}} = 2$pkts/sec, and $f_{\text{gen}} = 4$pkts/sec. Figures 4.2(a) and 4.2(b) show the delivery ratio in, respectively, the 90-node and the 155-node testbed. Being optimized for reliability, Arbutus significantly outperforms CTP at all the offered load points. Indeed, Arbutus's performance is relatively insensitive to the variations in the offered load. The reason for this is that Arbutus follows through with every

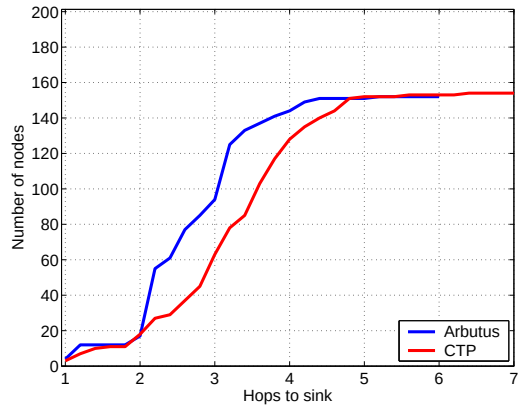
injected packet: it foregoes injecting new packets until the existing ones have been delivered. CTP, on the other hand, gives up on a given packet after 5 transmissions. Figure 4.2 shows that, while CTP's approach works almost as well at light traffic volumes, Arbutus's approach is preferable at higher offered load points. Figures 4.3(a) and 4.3(b) show the goodput achieved by Arbutus and CTP in, respectively, the 90-mote and the 155-mote testbed. Arbutus matches CTP's performance at the lower offered load levels, and outperforms CTP as the offered load increases. Figure 4.3(a) shows that, with Arbutus, the goodput in the 90-mote network peaks at $f_{\text{gen}} = 2$pkts/sec; this does not happen in the 155-mote network, where the goodput with Arbutus is monotonically increasing as a function of the offered load (Figure 4.3(b)). This is due to the lower density of the 90-mote network: there exist bottleneck nodes that get congested and thus limit the achievable goodput as the target offered load increases. In the 155-mote network, congestion is not as critical: the node density is significantly higher, and so is the achievable goodput at $f_{\text{gen}} = 4$pkts/sec. CTP does not deal with congestion as well as Arbutus, and its goodput saturates past $f_{\text{gen}} = 2$ pkts/sec at levels well below Arbutus's in both versions of the MoteLab testbed.

In the remainder of the paper, we will concentrate on only two offered load points: $f_{\text{gen}} = 0.1$ pkts/sec and $f_{\text{gen}} = 1$ pkt/sec, since the most remarkable performance changes occur in this interval. Any further increase of the offered load can only favor Arbutus, since its shallower, more efficient tree structure inherently reduces congestion. On the other hand, at lower offered load points, Arbutus's use of suboptimal links to enforce load balancing and reduce tree depth, as well as its use of unconstrained retransmissions, may reduce the overall goodput. Given that we work with a network size between 90 and 160 nodes, the offered load point $f_{\text{gen}} = 0.1$ pkts/sec will be henceforth referred to as *light load*, while $f_{\text{gen}} = 1$ pkt/sec will be

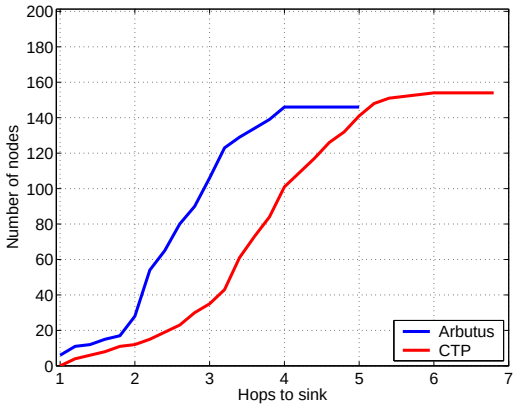
considered *heavy load*.

Hop count distribution. We now focus on the 155-node testbed and analyze the various performance indicators under light and heavy load. We begin with the routing tree structure; Figure 4.4 shows the distribution of the nodes as a function of the hop distance from the sink. We first observe that CTP always yields a deeper tree. The critical set, $\mathcal{N}_1$, is relatively small compared to the size of the network; some of the nodes in $\mathcal{N}_1$ must therefore be bottlenecks, bound to cause congestion if the offered load is large. Indeed, Figure 4.3(b) shows that, independently of the protocol, in the 155-mote testbed the average goodput drops to about 1/5 of the offered load under heavy load (in the 90-mote testbed, the average goodput drops to about 1/3 under heavy load, as shown in Figure 4.3(a)). Under heavy load, Arbutus achieves a slightly smaller coverage. Since Arbutus assumes that all nodes are of equal importance, a node never drops a given packet to favor another one, which means that nodes located upstream of particularly congested bottleneck nodes may never get a chance to get their packets through to the sink. While this is certainly not be desirable in a number of applications, a priority-based mechanism could easily be implemented on top of the basic Arbutus structure.

Routing cost. Figure 4.5 shows the node-by-node routing cost under light and heavy load. We plot the routing cost of each node versus its hop count, and we also show the overall routing cost as an average over all nodes below a given hop count (solid line). Despite the occasional outliers, on average Arbutus's routing cost is significantly lower than CTP's, and the performance gap between the two protocols widens as the offered load is increased. Note that the number of outliers dramatically increases for CTP under heavy load. In general, with Arbutus the



(a) $f_{\text{gen}} = 0.1$ pkts/sec.

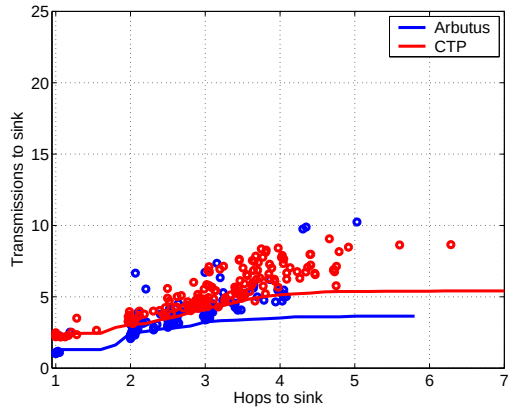


(b) $f_{\text{gen}} = 1$ pkts/sec.

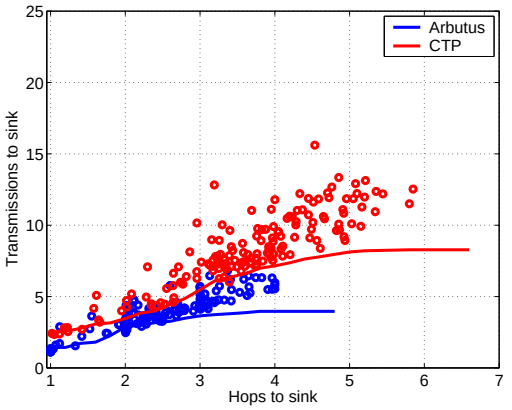
Figure 4.4. Hop count distribution of Arbutus and CTP. For each offered load point, we show the number of nodes below a given hop count.

routing cost remains consistent across different offered load points, while with CTP it suffers from a noticeable deterioration, as the average number of transmissions per delivered packet increases with the offered load.

Goodput. Figure 4.6 shows the number of nodes that achieve a given goodput under light and heavy load. In Figure 4.7, the normalized per-node goodput is plotted for each node (circles) along with its average over all nodes below a given hop count from the sink (solid line). Congestion is the main goodput-limiting factor



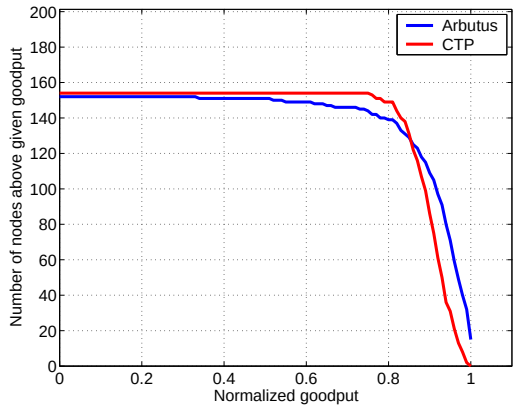
(a) $f_{\text{gen}} = 0.1$ pkts/sec.



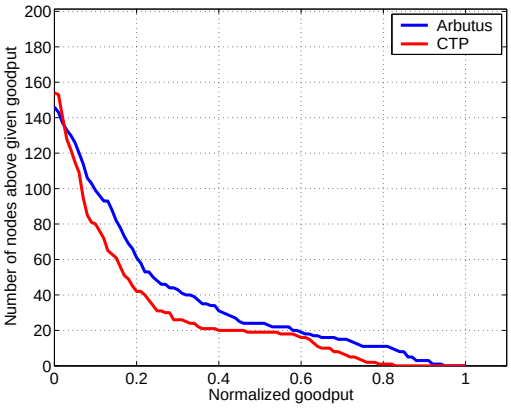
(b) $f_{\text{gen}} = 1$ pkts/sec.

Figure 4.5. Routing cost with Arbutus and CTP. We show the routing cost for each node (circles), along with the overall routing cost averaged over all nodes below a given hop distance from the sink (solid line).

under heavy load. The goodput degradation for nodes deeper in the tree is the consequence of the congestion control policies implemented by Arbutus and CTP. The dramatic increase in the number of goodput-challenged nodes for CTP under heavy load (Figure 4.6(b)) is the consequence of the increase in the number of routing cost outliers shown in Figure 4.5(b).



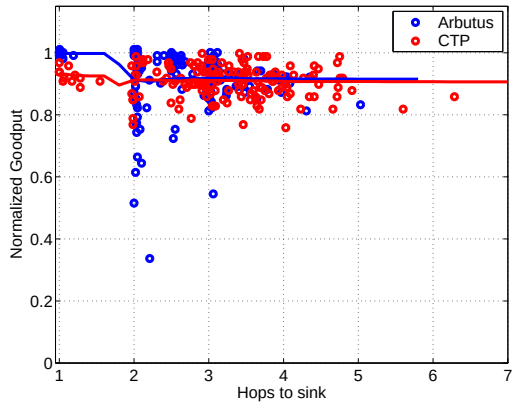
(a) $f_{\text{gen}} = 0.1$ pkts/sec.



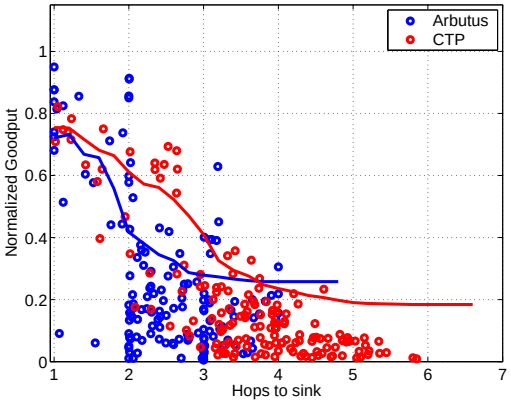
(b) $f_{\text{gen}} = 1$ pkts/sec.

Figure 4.6. Goodput with Arbutus and CTP. For each offered load point, we show the number of nodes above a given normalized goodput.

Delivery ratio. To determine whether the congestion control policy of Arbutus succeeds in preventing packet loss, we need to examine the delivery ratio, which is Arbutus's primary focus. We examine it from two different perspectives: Figure 4.8 shows the number of nodes that achieve a given delivery ratio under light and heavy load, while Figure 4.9 shows the delivery ratio achieved by all nodes at or below a given hop distance from the sink along with the average over all nodes (solid line). The delivery ratio of Arbutus does not experience a significant degradation as the offered load is increased; this is due to the fact that Arbutus's design is optimized



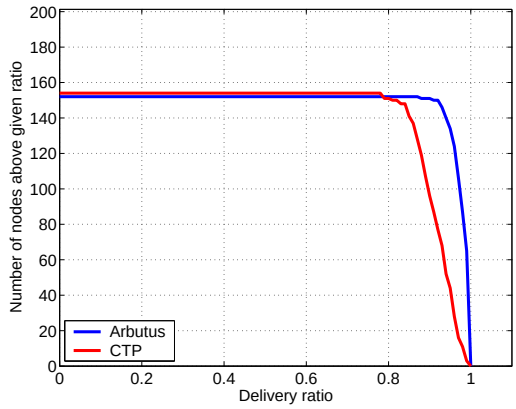
(a) $f_{\text{gen}} = 0.1$ pkts/sec.



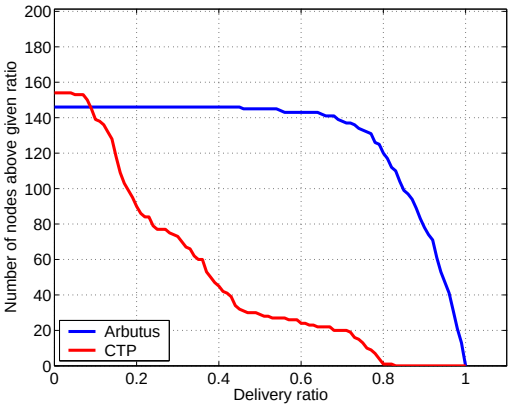
(b) $f_{\text{gen}} = 1$ pkts/sec.

Figure 4.7. With both Arbutus and CTP, the goodput is very sensitive to the offered load. The solid line indicates the goodput averaged over all nodes below a given hop count. The goodput for each node is also shown (circles).

for reliability, and packets are retransmitted until delivered. Another major contributing factor to the delivery rate performance of Arbutus is its congestion control scheme, whose only cost, at least with this sink assignment, is the slight coverage loss due to the fact that nodes are required to stop transmitting if their parent is congested.



(a) $f_{\text{gen}} = 0.1$ pkts/sec.



(b) $f_{\text{gen}} = 1$ pkts/sec.

Figure 4.8. Delivery ratio with Arbutus and CTP. For each offered load point, we show the number of nodes above a given normalized goodput.

4.7 Joint impact of topology and offered load

Topology as a parameter. The results in 4.6 have been obtained after arbitrarily fixing the sink assignment in the MoteLab testbed, but we cannot claim these results to be a good indication of the performance of Arbutus and CTP in a generic network topology. In the assessment of a routing protocol, network topology should be treated as a parameter, similarly to the target offered load. To emphasize how crucial topology is, we perform several experiments with different sink assignments.

Since it is not practical to show the amount of detail presented in 4.6 for every sink assignment, we distil the performance indicators down to their average value over all nodes in the network, after obtaining each per-node value as the average over at least 3 runs. In particular, we visualize the routing performance on a goodput-delivery ratio plane (henceforth referred to as *performance plane*), and we represent the routing cost on a hop count-total number of transmissions plane (henceforth named *cost plane*).

Performance plane. Figure 4.10 shows the performance plane for a number of network topologies in MoteLab. The goodput varies greatly depending on the sink assignment, especially under heavy load: changing the sink assignment is tantamount to dealing with a different network. For a number of sink assignments under light load CTP yields a better goodput; this is particularly noticeable in topologies 12, 25, and, in particular, 16, which appear to be performance outliers. For the most part, however, the goodput of Arbutus and CTP are fairly close, which leads us to conclude that Arbutus's approach does not generally cause a significant degradation of the goodput. This is mainly due to the interplay between routing, load balancing, and congestion control: a shallower tree means that nodes are never loaded unnecessarily, the network relayed load is minimized, and, thanks to load balancing, no individual node is overloaded, barring topological constraints (such as a node being the only way for a given subnetwork to reach the sink). The goodput-delivery ratio performance plane also shows that the delivery ratio of Arbutus is always superior. It varies relatively little across different topologies, and it is fairly insensitive to the offered load. On the other hand, CTP's reliability varies widely as a function of offered load and is not as consistent as Arbutus's across different topologies.

Figure 4.11 shows the performance plane for a number of sink assignments in

Twist. Due to its very regular topology, there is very little variability across different sink assignments, and the performance of both protocols is considerably better than in MoteLab. Arbutus achieves virtually 100% reliability under light load, and is always above 98% reliability under heavy load. CTP always outperforms Arbutus under light load, although Arbutus's goodput is always above 85% of the target offered load. Under heavy load, Arbutus typically outperforms CTP in terms of goodput.

Figure 4.12 shows the performance plane for a number of Tutornet experiments. Due to to Tutornet's higher density, the impact of the topology is not as noticeable as in MoteLab (but is nonetheless more noticable than it is in Twist).

Cost plane. In MoteLab, the routing cost, shown in Figure 4.13, is also affected by huge performance variations depending on the sink assignment. Arbutus typically achieves a smaller routing cost with fewer transmissions per delivered packet. Its routing cost is relatively insensitive to the offered load, while CTP's cost varies greatly across the two offered load points. In particular, note that, on average, CTP's routing cost experiences an approximate 40% increase between light and heavy load. This is due to Arbutus's more efficient tree structure: unnecessarily long routes are avoided, and a smaller relayed load is imposed onto the network.

The cost plane for our Twist experiments is shown in Figure 4.14. Given the regular grid-like layout of Twist, Arbutus's average routing cost is fairly consistent across different sink assignments and always lower than CTP's. Like on MoteLab, Arbutus's cost does not vary significantly as the target offered load is increased. Moreover, Arbutus always builds shallower trees. Tree size and routing cost are due to the link estimator, based on our DUCHY link estimator [71]. One key difference between the link estimator embedded into Arbutus in this dissertation and DUCHY

Table 4.1

MOTELAB, 155 NODES: OVERALL RESULTS, AVERAGED OVER ALL THE TOPOLOGIES THAT WE CONSIDERED, UNDER LIGHT LOAD. FOR EACH PERFORMANCE INDICATOR, WE PROVIDE THE MEAN μ AND THE STANDARD DEVIATION σ IN THE FORM $\mu \pm \sigma$.

Performance indicator	Arbutus	CTP
Delivery ratio	0.98 ± 0.02	0.89 ± 0.02
Normalized goodput	0.90 ± 0.07	0.92 ± 0.02
Hop count	2.65 ± 0.50	3.22 ± 0.56
Routing cost	4.25 ± 0.68	5.72 ± 0.74
Coverage at 5%	0.99 ± 0.01	0.99 ± 0.01
Coverage at 25%	0.99 ± 0.01	0.99 ± 0.01
Coverage at 50%	0.98 ± 0.01	0.99 ± 0.01
Control overhead	0.17 ± 0.03	0.09 ± 0.05
Duplicate suppression	0.99 ± 0.01	0.94 ± 0.03

is that here we take relayed load into account for link selection, which DUCHY does not do. As discussed in Section 4.5, load balancing prevents congestion, which means that fewer transmissions are needed for each delivery: a comparison of the cost results in this dissertation with the ones in [71], also obtained on MoteLab (though at a time when the network size was about 50), shows that the results in this document are considerably better, thereby confirming our point that load balancing reduces routing cost.

Overall results. In Table 4.2 we present an overview of our experimental results on the 155-node MoteLab testbed for the two offered load points of 0.1 and 1 pkt/sec/node, averaged over all topologies.

The primary goal of Arbutus is reliability; it achieves an average 10% improve-

Table 4.2

MOTELAB, 155 NODES: OVERALL RESULTS, AVERAGED OVER ALL THE TOPOLOGIES THAT WE CONSIDERED, UNDER HEAVY LOAD. FOR EACH PERFORMANCE INDICATOR, WE PROVIDE THE MEAN μ AND THE STANDARD DEVIATION σ IN THE FORM $\mu \pm \sigma$.

Performance indicator	Arbutus	CTP
Delivery ratio	0.92 ± 0.03	0.40 ± 0.06
Normalized goodput	0.34 ± 0.09	0.22 ± 0.03
Hop count	2.45 ± 0.41	3.04 ± 0.61
Routing cost	4.05 ± 0.48	7.67 ± 0.90
Coverage at 5%	0.78 ± 0.12	0.77 ± 0.11
Coverage at 25%	0.50 ± 0.16	0.32 ± 0.09
Coverage at 50%	0.27 ± 0.10	0.15 ± 0.05
Control overhead	0.05 ± 0.02	0.26 ± 0.20
Duplicate suppression	0.99 ± 0.01	0.85 ± 0.04

ment over CTP under low load, and as much as a 137% improvement under heavy load. The goodput is also satisfying: by leveraging on the interplay between reliability and congestion, Arbutus almost matches CTP's goodput under low load (it only loses an average 3%), and outperforms CTP by as much as 50% under heavy load, improving CTP's 0.22 to 0.34. Note that the 3% goodput drop under light load is the main drawback of Arbutus's use of long-hop routing, which pays off from most other viewpoints under light load, and from virtually all standpoints under heavy load. A considerable advantage of Arbutus's long-hop routing is the significant reduction of the routing cost. Independently of the offered load, Arbutus reduces the average node depth by an average of 19%, and reduces the number of transmissions per delivered packet by 47% under light load, and by 26% under heavy load. The coverage of both protocols is virtually identical under light load. Under heavy load,

however, Arbutus outperforms CTP. Arbutus's more aggressive congestion control policy, responsible for the excellent reliability performance, has a cost in the form of control overhead under light load, which is almost double for Arbutus with its 17% compared to CTP's 9%. Nevertheless, the situation is completely reversed under heavy load: Arbutus's control overhead drops to just 5%, only 20% of CTP's overhead. This is another indication of Arbutus's excellent scalability properties. Finally, we observe another appealing feature of Arbutus, which is its extremely effective duplicate suppression scheme. The exact cost depends on the implementation; in ours we need to set aside 2 bytes per network node, *i.e.* 310 bytes. Note that duplicate suppression plays a major role in congestion prevention, as it keeps useless packets from increasing the network load. Under light load, CTP injects a volume of duplicate traffic that on average equals 6% of its data traffic volume. If we add this data traffic overhead to the control overhead, it turns out that the total overhead of CTP (duplicate data plus control traffic) almost matches CTP's control overhead. Arbutus's duplicate suppression scheme, therefore, compensates for the extra control traffic injected by the congestion control mechanism.

4.8 Impact of the critical set size

Since we are dealing with many-to-one collection trees, the critical set size is the most natural way to numerically characterize a given topology. As a general rule, the larger the critical set, the easier it is to get more packets from more nodes to the sink at a lower cost. A relatively large sink degree can be expected to correspond to a relatively large goodput and relatively low routing cost. The delivery ratio, on the other hand, can only be expected to be marginally affected, at least for Arbutus. A performance degradation can be expected to occur in topologies with a small critical set, which are more likely to suffer from severe congestion. In

fact, if only a few critical nodes are relaying traffic from many upstream peers, interference and congestion are unavoidable. The problem with a similar scenario is that congestion begets more interference, as congestion management requires additional control traffic that interferes with the intense data traffic.

In this subsection we only consider results from MoteLab. We have already remarked the relatively small variability observed in our experiments with Twist, due to the regularity of its layout. All the Twist sink assignment that we considered have a critical set size between 40 and 60; with a network size between 95 and 100, the Twist results shown in Figures 4.11 and 4.14 confirm our observations on the benefits of a large critical set.

Figure 4.15 provides an overview of the joint impact of offered load and topology. Light and heavy load are considered, and results are averaged over all topologies within a given interval of critical set sizes. Three size classes are considered: less than 20 critical nodes, between 20 and 40, and over 40. Arbutus is shown to yield a significantly better goodput (Figure 4.15(a)) and coverage $\rho = 0.25$ (Figure 4.15(b)) with sink assignments with a larger critical set. A larger critical set also helps reduce the average hop count (Figure 4.15(c)) as well as the routing cost (Figure 4.15(d)); this is true for both protocols, but Arbutus achieves a lower average hop count (shallower tree) along with a lower routing cost. The amount of control overhead (Figure 4.15(e)) also varies considerably with the topology. We have seen in Table 4.2 that CTP has a lower control overhead under light load, but Figure 4.15(e) shows that this only happens if the critical set is small. CTP's duplicate suppression rate (Figure 4.15(f)) becomes worse as the critical set increases, whereas Arbutus's is not affected by the topology.

4.8.1 Results on the 155-node MoteLab testbed

Goodput. Figure 4.16 shows the goodput in MoteLab as a function of the critical set size imposed by each protocol. In 4.7 and Figure 4.10 we have observed that, under light load, Arbutus may be outperformed by CTP in terms of goodput, depending on the topology. Figure 4.16(a) shows that the topologies for which Arbutus is outperformed in terms of goodput all have a small critical set size (this is particularly true for 12, 16, and 25). Under heavy load, Figure 4.16(b) confirms that Arbutus nearly always exceeds (and occasionally matches) CTP's goodput. Arbutus's more aggressive congestion control policy is highly beneficial under heavy load, but occasionally over-restrictive under light load. Under light load, however, congestion may be induced by the topology (with a small critical set, bottlenecks cause congestion even if the offered load is low, as the few nodes in the critical set have a large number of descendants).

Coverage

We examine the coverage achieved by Arbutus and CTP using the target values $\rho = 0.25$ (a node is covered if it delivers 25% of its target offered load to the sink: an average of 0.25 pkts/sec under heavy load, and an average of 0.025 pkts/sec under light load). We only show the results under heavy load, because under light load both protocols achieve at least a 95% coverage with $\rho \leq 0.75$. Figure 4.17 shows that Arbutus consistently outperforms CTP, as expected based on its superior goodput under heavy load

Congestion control

We now examine the congestion control performance of both protocols using delivery ratio fairness as defined by Eq. 4.1. As can be expected from its superior reliability, Arbutus consistently outperforms CTP under heavy load, as shown in

Figure 4.18. The gap between the performances of Arbutus and CTP narrows down as the critical set size increases, as there are fewer bottlenecks and congestion becomes less of an issue. The results under light load are not shown, as the two protocols exhibit equivalent performances (a delivery rate fairness of 1 independently of the sink assignment).

4.8.2 Results on the 90-node MoteLab testbed

We now shift our attention to the 90-mote testbed of March 2008; we focus on heavy load conditions and examine a wide range of topologies by varying the sink assignment. Figures 4.19 and 4.20 show, respectively, the goodput and the reliability achieved by Arbutus and CTP as a function of the critical set size. Particular effects can be observed in the 90-node testbed due to its much more sparse connectivity. In particular, Figure 4.20 shows that sinks with very low degrees have a rather unpredictable performance. A low sink degree definitely hurts CTP's performance, which appears to be significantly more erratic due to CTP's use of a constrained number of transmissions. There are some outliers in Arbutus's performance as well: 12, 16, and 56 are truly awful for both protocols. They all have small degrees, but so do many other sink assignments that do not experience the total collapse suffered by these three outliers.

4.9 Impact of transitional links

Links in the transitional region are affected by a lack of long-term stability. We consider the neighborhood of node i to be stable if the links between i and the nodes in $\mathcal{N}_1$ have a stable PDR over time. We define *neighborhood instability* based on the fraction of links in the transitional region, *i.e.* the links whose quality lies below a given RSS threshold Θ_r and LQI threshold Θ_l. We will use $\Theta_r = -87$dBm [21] and $\Theta_l = 100$. Figure 4.21 shows that, in the 90-mote MoteLab testbed, sink assignments

with a poor delivery rate performance have instability values close or equal to 1, which means that a given sink has not reliable links to any other node: its links may be asymmetric, or even poor in both directions. A high instability is more likely to be experienced by nodes with a low degree due to their limited spatial diversity; this is the condition of the performance outliers in the 90-node MoteLab testbed (in particular, nodes 12, 16, and 56). Indeed, when these nodes are used as sinks in the 90-node MoteLab network, we have observed in our experiments that their beacons do not get heard easily, and/or their ACKs get dropped. Once their neighbors finally do hear their beacons and start routing data to them using the inbound links of the sinks, data flow may proceed relatively smoothly in the downstream direction (asymmetric link), or may again be hindered by a bidirectionally lossy link. While the latter case is certainly worse, asymmetry is also critical: if the outbound links of the sink are lossy, the ACKs sent by the sink back to its neighbors to confirm reception are typically lost. Consequently, the sink neighbors continue to retransmit, and the sink continues to receive and send ACKs that get dropped, causing the neighbors to further retransmit. The goodput suffers greatly, and the sink accumulates a very large number of duplicates. Arbutus has a very effective duplicate suppression scheme (the success rate averaged over all our experimental runs is over 0.99), whereas CTP's scheme is much more erratic, and in the case of the outliers (12, 16, 25) the sink sends a disproportionately large number of duplicates to the gateway, especially in the case of the outliers, as shown in Figure 4.22. Figure 4.23 shows that there is a definite correlation between CTP's duplicate suppression rate and instability: CTP has a much harder time suppressing duplicates in unstable topologies, simply because there are many more duplicates to suppress.

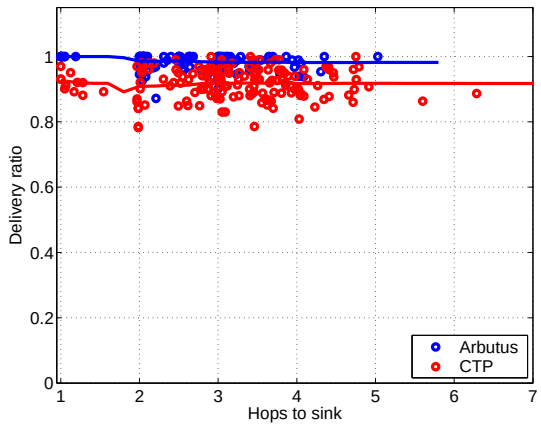
The reason for the behavior of the performance outliers lies in the interplay between link instability and congestion, as both Arbutus and CTP use broadcast

beacons for congestion control. If node is congested, it signals its condition with a broadcast beacon; if its outgoing links are very lossy (unstable), its descendants may not hear the beacon. ACKs are not expected of broadcast traffic in either CTP or Arbutus, and even if they were, congestion would still occur while beacons are being retransmitted. So if a node forms a bottleneck and happens to have unstable links, it may occasionally be unable to recover from congestion. There exists an issue with instability with any node that forms a bottleneck: instability on a relay's outbound links also affects the delivery ratio of its descendants. The closer the main source(s) of instability is (are) to the sink, the wider the impact on the delivery ratio.

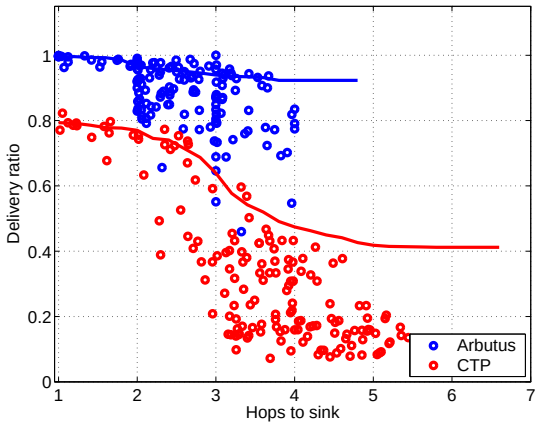
Instability is a byproduct of multipath fading; the effect of fading over a link can therefore be expected to display a frequency-dependent behavior. In fact, unstable topologies display a wide range of variability across different frequencies. Figure 4.24 confirms our expectation: we show the goodput-delivery performance of sink assignment 56 as we modify the 802.15.4 channel employed by the CC2420 transceiver. Channel 26 was employed in the experiments whose results are shown in Figure 4.24, and yields the worst performance at sink assignment 56, while channel 15 yields the best performance. The impact of interference from overlapping 802.11 channels does not appear to be the main issue; note that 802.15.4 channels 11 and 18 overlap with 802.11 channels 1 and 6, while 802.15.4 channels 15, 20, and 26 do not overlap. This experiment, therefore, confirms that fading may have a devastating impact on topologies with a small critical set.

The coexistence of 802.11 and 802.15.4 has been the subject of experimental studies [73–75]; in general, interference from 802.11 has a significant impact on the routing performance in 802.15.4 channels that overlap with 802.11. The Tutornet testbed suffers from a great amount of 802.11 interference: not only are there access points, but there is also a high-end tier of Stargates that are used for mote

programming and packet trace collection from the motes. We have run a few experiments on 802.15.4 overlapping channels, and noticed temporary drops in the delivery ratio due to 802.11 interference causing data packets, control traffic, and layer 2 ACKs to be dropped, which results in heavy congestion and ingress drops. To ensure consistency, all the experimental results shown in this paper were obtained on non-overlapping channels unless otherwise noted.

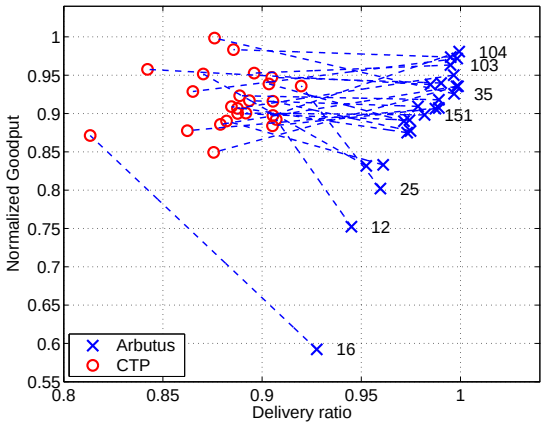


(a) $f_{\text{gen}} = 0.1$ pkts/sec.

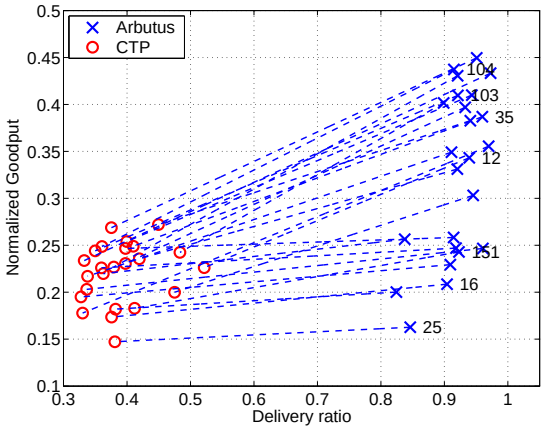


(b) $f_{\text{gen}} = 1$ pkts/sec.

Figure 4.9. The average delivery ratio with Arbutus does not suffer from a significant degradation as the offered load increases, differently from CTP. The solid line indicates the goodput averaged over all nodes below a given hop count. The goodput for each node is also shown (circles).

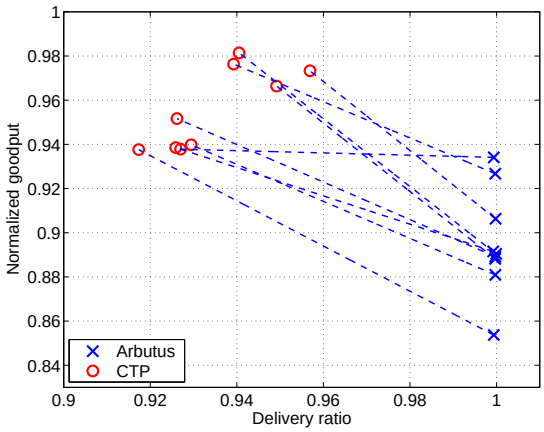


(a) $f_{\text{gen}} = 0.1$ pkts/sec.

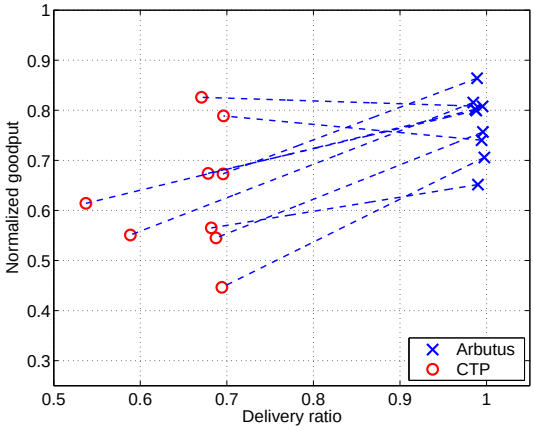


(b) $f_{\text{gen}} = 1$ pkts/sec.

Figure 4.10. Performance plane for Arbutus and CTP in MoteLab (about 155 nodes). Results at several sink assignments are shown, and in a few cases the sink assignment itself is indicated. Different sink assignments correspond to different network topologies. The impact of topology on the goodput is significant, particularly at higher offered load points (note the different scale of the two subfigures). Arbutus's delivery ratio is, however, relatively insensitive to the offered load.

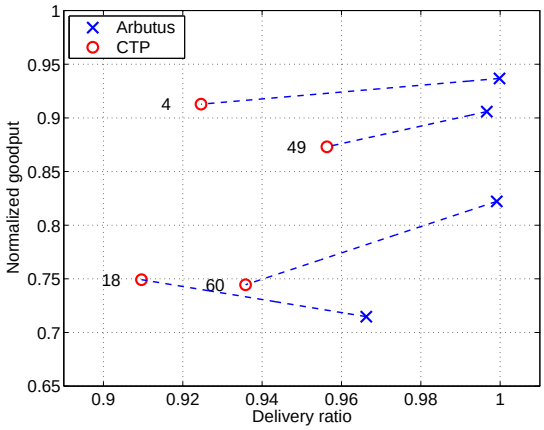


(a) $f_{\text{gen}} = 0.1$ pkts/sec.

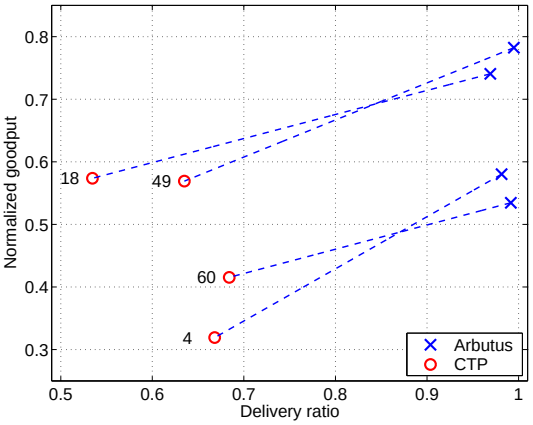


(b) $f_{\text{gen}} = 1$ pkts/sec.

Figure 4.11. Performance plane for Arbutus and CTP in Twist (about 100 nodes). The impact of topology on the goodput is not as significant as in MoteLab, given the regular grid-like structure of the Twist network.

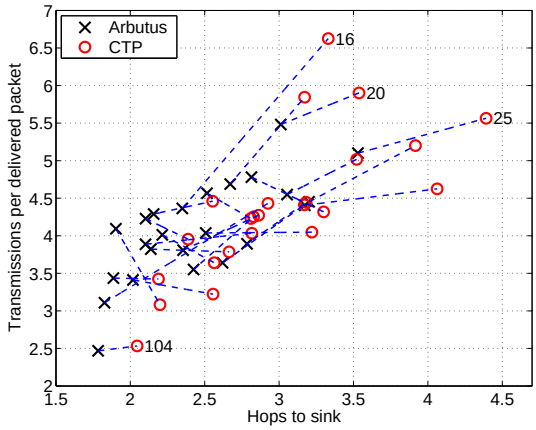


(a) $f_{\text{gen}} = 0.1$ pkts/sec.

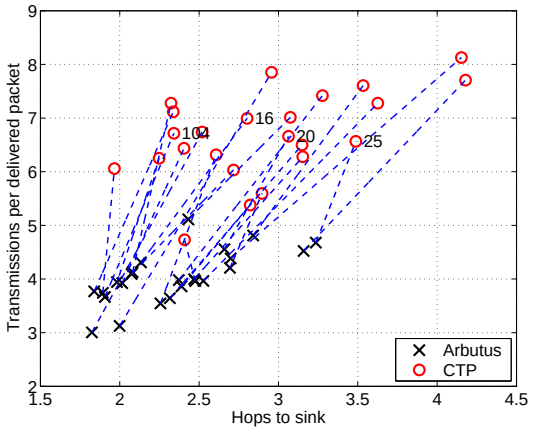


(b) $f_{\text{gen}} = 1$ pkts/sec.

Figure 4.12. Performance plane for Arbutus and CTP in Tutornet (about 90 nodes). The topology has an impact on the goodput, albeit not as much as in MoteLab, due to Tutornet's higher density.

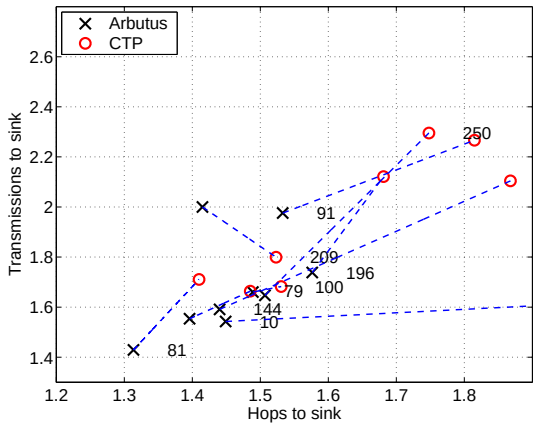


(a) $f_{\text{gen}} = 0.1$ pkts/sec.

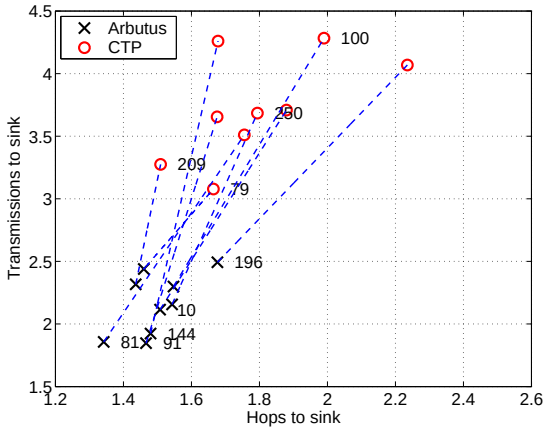


(b) $f_{\text{gen}} = 1$ pkts/sec.

Figure 4.13. Cost plane for Arbutus and CTP in MoteLab, 155 nodes. Arbutus greatly reduces the routing cost (transmissions per delivered packet) as well as the depth of the routing tree. Note the different scale in the two subfigures, and the relative consistency of Arbutus's routing cost as the offered load is increased.

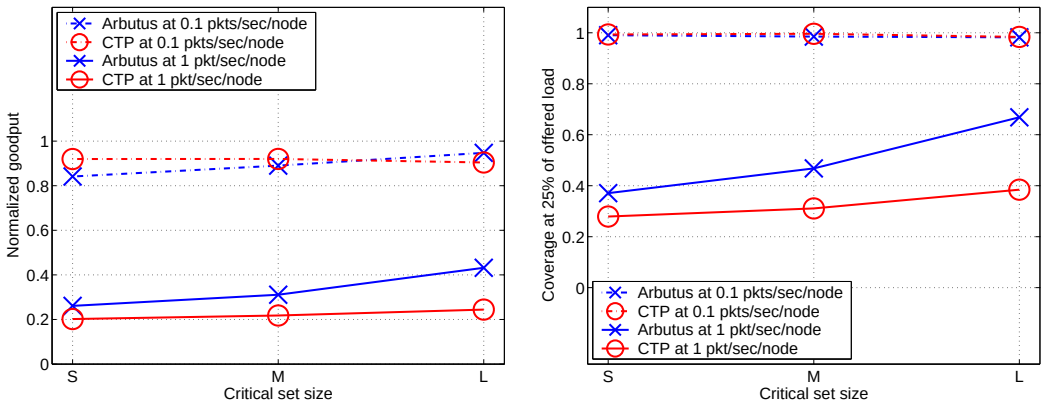


(a) $f_{\text{gen}} = 0.1$ pkts/sec.



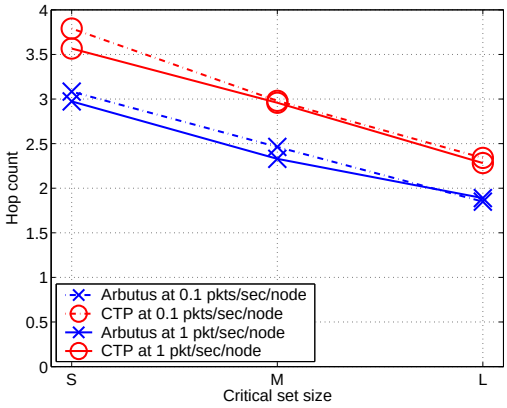
(b) $f_{\text{gen}} = 1$ pkts/sec.

Figure 4.14. Cost plane for Arbutus and CTP in Twist, about 95 nodes. Due to the regular grid-like layout of the network, all sink assignments correspond to benign topologies, and the performance is consistent across different topologies.

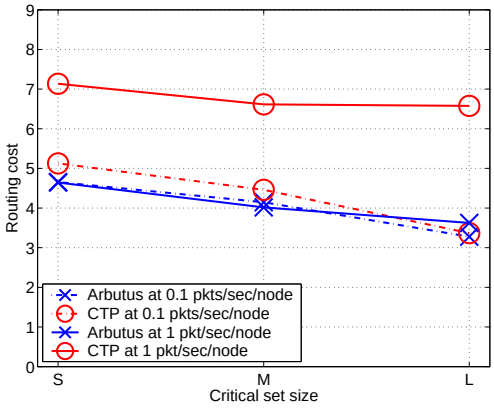


(a) Goodput.

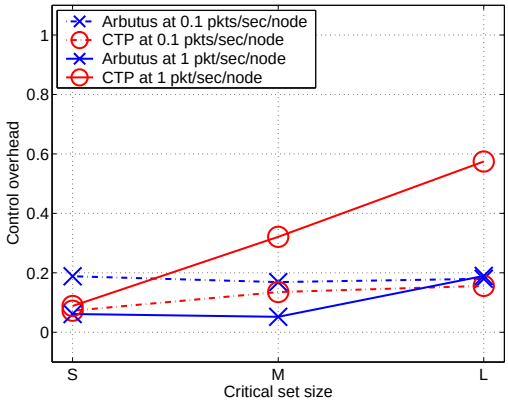
(b) Coverage at $\rho = 0.25$.



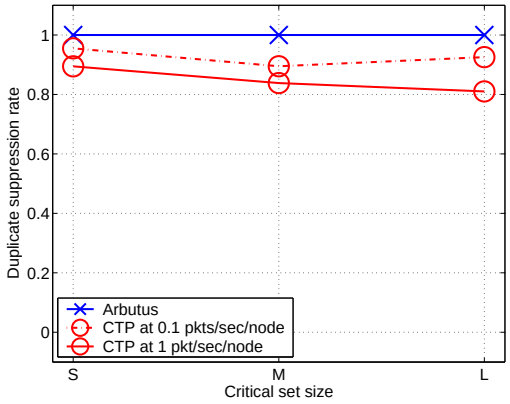
(c) Hop count.



(d) Routing cost.

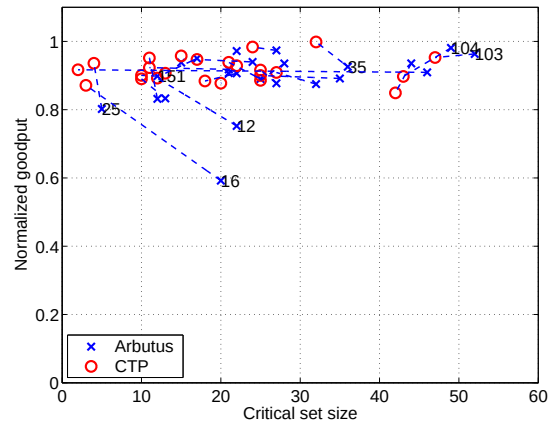


(e) Control overhead.

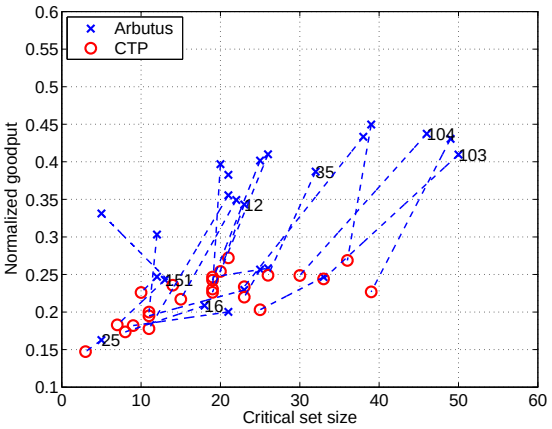


(f) Duplicate suppression ratio.

Figure 4.15. Overview of the routing performance of Arbutus: the results shown herein are averaged over all topologies with a critical set size within one of three classes: small (S, less than 20 critical nodes), medium (M, 20 to 40 critical nodes), and large (L, over 40 critical nodes).



(a) $f_{\text{gen}} = 0.1$ pkts/sec.



(b) $f_{\text{gen}} = 1$ pkts/sec.

Figure 4.16. MoteLab, 155 nodes. Goodput as a function of the size of the critical set.

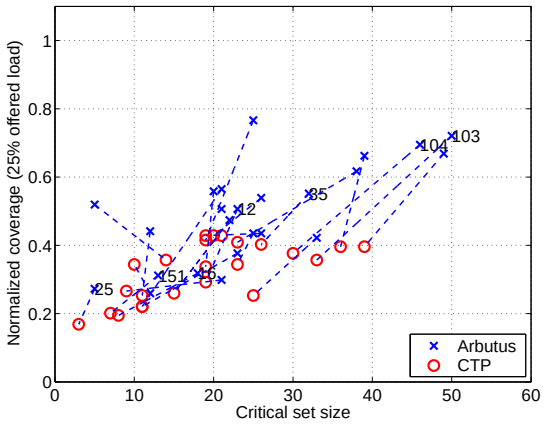


Figure 4.17. MoteLab, 155 nodes. Normalized coverage at 25% of the offered load under heavy load.

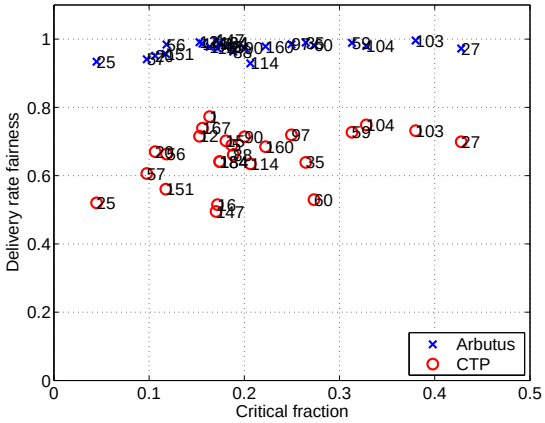


Figure 4.18. MoteLab, 155 nodes. Congestion control: delivery rate fairness under heavy load.

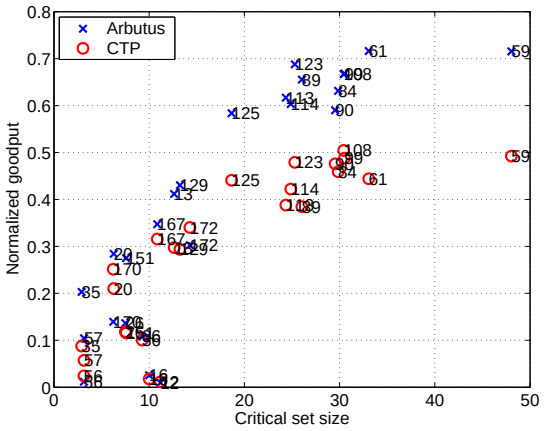


Figure 4.19. MoteLab, 90 nodes: goodput as a function of the critical set size.

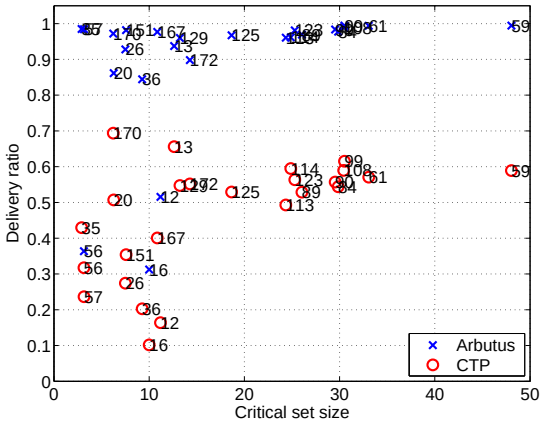


Figure 4.20. MoteLab, 90 nodes: reliability as a function of the critical area size.

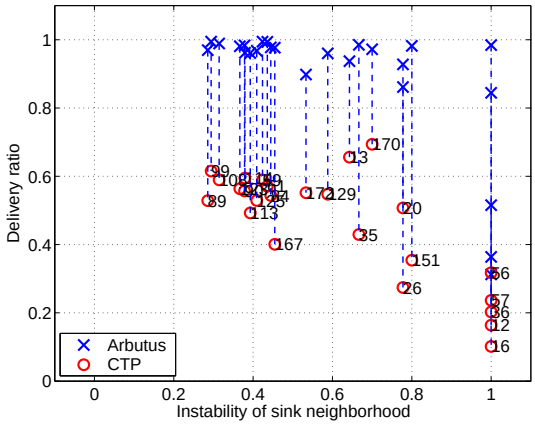


Figure 4.21. MoteLab, 90 nodes. Neighborhood instability is a good predictor of performance degradation. Topologies 12, 16, and 56 exhibit a very poor performance with both protocols.

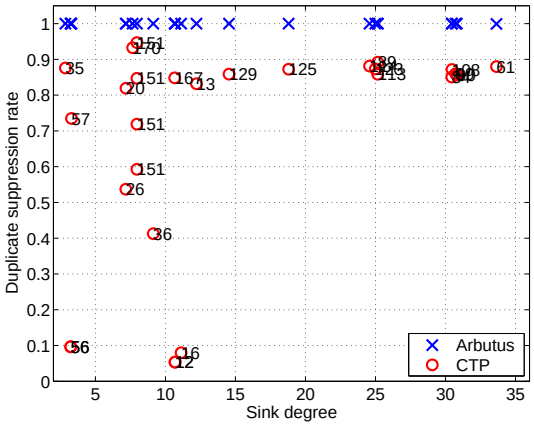


Figure 4.22. MoteLab, 90 nodes. Outliers 12, 16, and 56 have both a small critical set and a low CTP duplicate suppression rate, which is a good predictor of instability.

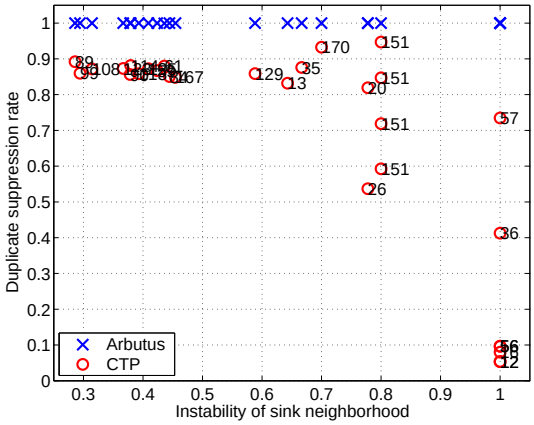


Figure 4.23. MoteLab, 90 nodes. Correlation between CTP duplicate suppression rate and instability; 12, 16, and 56 again stand out as outliers.

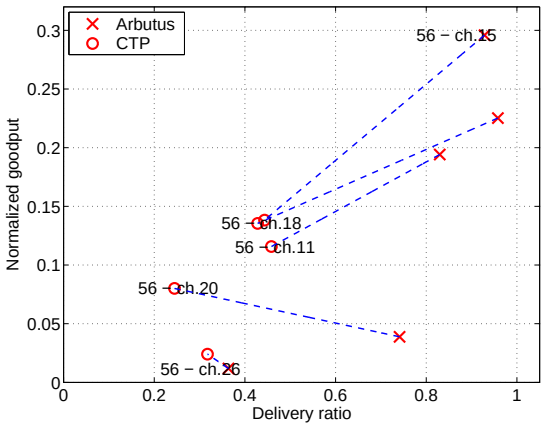
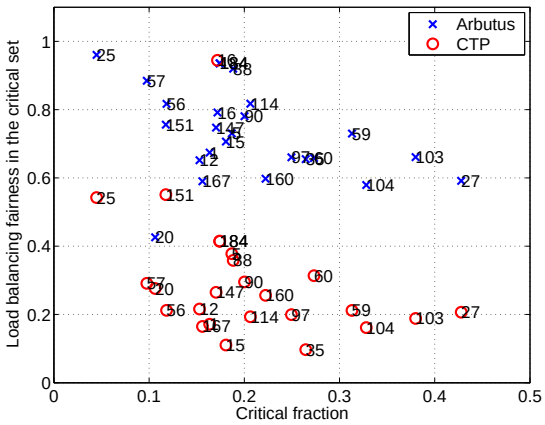
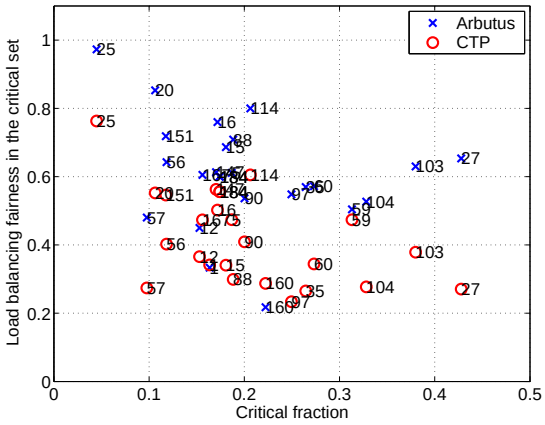


Figure 4.24. MoteLab, 90 nodes. Different 802.15.4 with sink assignment 56 produce very different results: for instance, fading is not an issue on channel 11, but is destructive on channel 26.

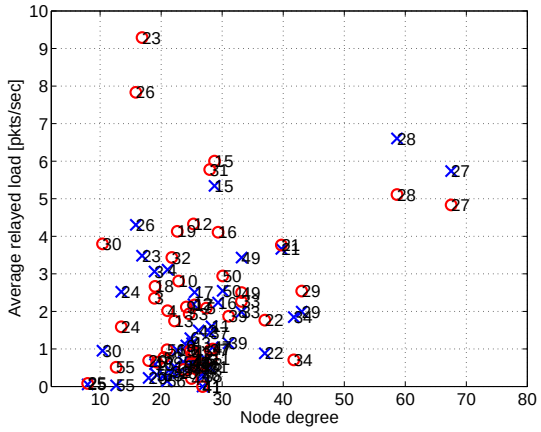


(a) $f_{\text{gen}} = 0.1$pkts/sec

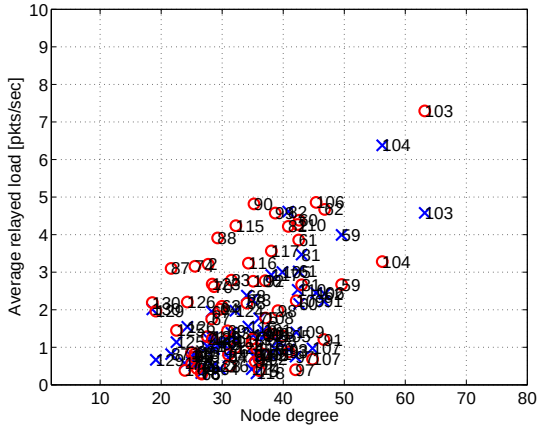


(b) $f_{\text{gen}} = 1$pkts/sec

Figure 4.25. MoteLab, 155 nodes: load balancing fairness.

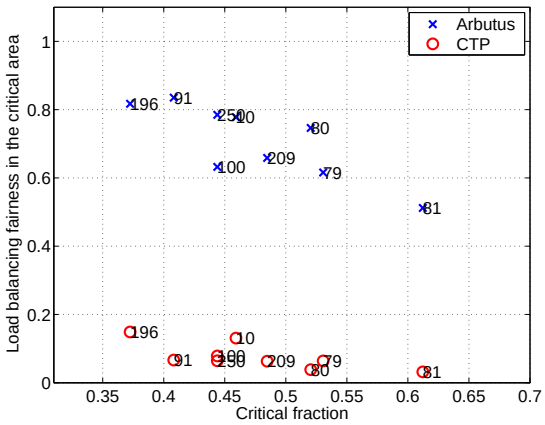


(a) Floor 1, $f_{\text{gen}} = 1$ pkts/sec.

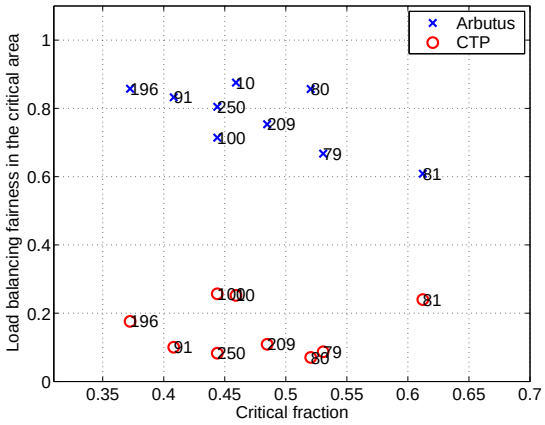


(b) Floor 2, $f_{\text{gen}} = 1$ pkts/sec.

Figure 4.26. Per-node relayed load, averaged over all runs, as a function of the node degree, for two of the three MoteLab floors under heavy load.

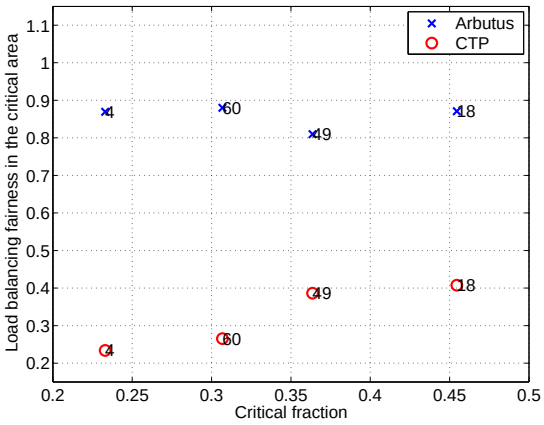


(a) $f_{\text{gen}} = 0.1$pkts/sec

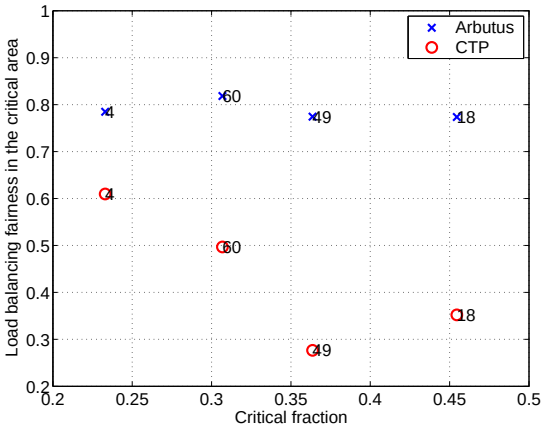


(b) $f_{\text{gen}} = 1$pkts/sec

Figure 4.27. Twist: load balancing fairness.

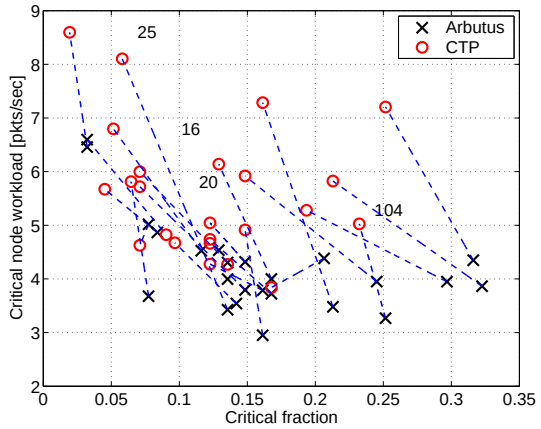


(a) $f_{\text{gen}} = 0.1$pkts/sec

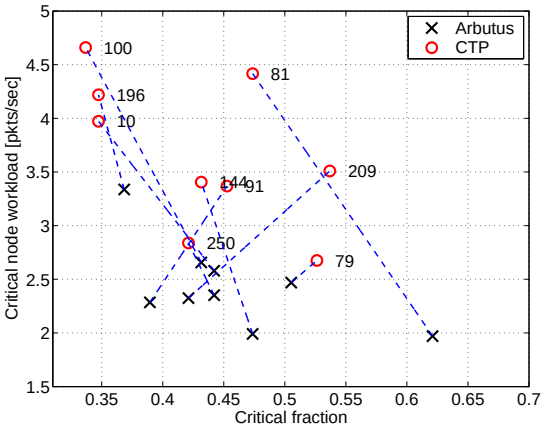


(b) $f_{\text{gen}} = 1$pkts/sec

Figure 4.28. Tutornet: load balancing fairness.



(a) Motelab, 155 nodes, $f_{\text{gen}} = 1$pkts/sec



(b) Twist, $f_{\text{gen}} = 1$pkts/sec

Figure 4.29. Arbutus reduces the average per-node workload in the critical set, which is easier to do in Twist, where the critical fraction is generally much larger, thanks to the regular grid topology.

CHAPTER 5

EXPERIMENTAL LOAD BALANCING ANALYSIS

5.1 Load balancing fairness

Arbutus implements network-layer load balancing by factoring load into the routing metric. To evaluate the load balancing performance, we begin by adapting the fairness metric (described in Section 4.4) as a load balancing metric by redefining it over the critical set and replacing delivery rate with relayed load as

$$\phi_\beta \triangleq \frac{(\sum_{r\in\mathcal{N}_1}\beta_i)^2}{|\mathcal{N}_1|\sum_{i\in\mathcal{N}_1}\beta_i^2}. \tag{5.1}$$

This redefinition of the fairness metric serves to quantify the fairness of a protocol in the distribution of the relayed load over the critical nodes. Figure 4.25 shows the load balancing fairness of Arbutus and CTP in the 155-node MoteLab testbed under light and heavy load for several topologies, identified by their critical size.

In terms of load balancing, CTP should be seen as a baseline rather than a benchmark, since it does not explicitly pursue load balancing: we expect Arbutus to outperform it in this regard. We considered choosing a different benchmark, but to our knowledge, there exist no strenuously tested implementations of network-layer load balancing schemes. Rather than producing our own implementation of an existing scheme, we continue to use the strenuously tested CTP.

By actively pursuing load balancing, Arbutus outperforms CTP under light load.

The performance gap, however, becomes much narrower under heavy load, and Arbutus is occasionally outperformed. The reason for this behavior is the presence of topological bottlenecks in the network, *i.e.*, nodes that need to be used by a given subnetwork to deliver packets to the sink. Figure 4.26 shows the average relayed load for two floors in the 155-node MoteLab testbed under heavy load; nodes with large critical sets are topological bottlenecks for several sink assignments. Nodes 27 and 28 on floor 1, and nodes 59, 103, and 104 on floor 2 have the largest critical sets and are overloaded by both protocols.

With Arbutus's depth-limited routing tree, however, nodes never act as relays unnecessarily: the relayed load is kept small by using fewer, longer hops whenever possible. With CTP, on the other hand, a large number of nodes with comparatively small critical sets are overloaded.

In Twist, where the node density is higher, the layout is regular, and there are no bottlenecks, Arbutus always outperforms CTP, as shown in Figure 4.27. A comparison of the horizontal scales of Figures 4.25 and 4.27 shows that all of the Twist sink assignments have as large a critical fraction as the MoteLab sink assignments with the largest critical sets. It is therefore not surprising that, just like Arbutus outperforms CTP in terms of load balancing for all the Twist topologies that we considered, it also outperforms CTP for all the MoteLab topologies with a critical fraction larger than 0.25, independently of the offered load. The results obtained on Tutornet, shown in Figure 4.28, are similar to the ones on Twist (the critical size range over the various sink assignments is similar to what we have in Twist).

These results show that Arbutus can achieve a considerable degree of load balancing in benign topologies with a large critical set and relatively high density. CTP, on the other hand, overloads a few nodes and uses up their resources, thus

curtailing their lifetime. Under heavy load, CTP's concentration on a small number of relays adds congestion to the network, and contributes to the degradation of CTP's goodput. Arbutus's longer hops extend the critical set: more critical nodes become less critical, as the load can be redistributed. Arbutus avoids overloading nodes unless they form topological bottlenecks; it also reduces the overall relayed load by keeping the routes as short as possible so that no node relays data packets unnecessarily. This approach keeps congestion to its topology-induced minimum, and allows Arbutus to meet its reliability goals.

To confirm this, Figure 4.29 shows that Arbutus reduces the average per-node workload in the critical set. The comparatively higher critical fractions of Twist's sink assignments (Figure 4.29(b)) are obviously more conducive to workload reduction in the critical set.

5.2 Transport geography

We employ tools from the field of transport geography for the interpretation of the elusive aspects of the load balancing performance analysis: the impact of topological bottlenecks. Starting from a basic transport geographical concepts, the valued-graph matrix, we provide a quantitative characterization of topological bottlenecks.

Valued-graph matrix. The valued-graph matrix $\mathbf{L}$ [18] can be obtained as follows. Given a link cost function $\lambda_{i,j}$ for link (i,j), the 1st order $\mathbf{L}^{(1)} = \{l_{i,j}^{(1)}\}$ matrix is constructed as follows:

$$l^{(1)}{}_{i,j} = \begin{cases} \lambda_{i,j} & \text{if } i \neq j \text{ and } \lambda_{i,j} \neq 0, \\ 0 & \text{if } i = j, \\ \infty & \text{otherwise} \end{cases}$$

For $2 \leq n \leq \delta$, $\mathbf{L}^{(n)} = \{l_{i,j}^{(n)}\}$ is constructed based on $\mathbf{L}^{(n-1)} = \{l_{i,j}^{(n-1)}\}$ as follows:

$$l_{i,j}^{(n)} = \min_{k \in \mathcal{N}} (l_{i,k}^{(n-1)} + l_{k,j}^{(n-1)})$$

The valued-graph matrix is equal to $\mathbf{L} = \mathbf{L}^{(\delta)} \triangleq \{l_{i,j}\}$, and $l_{i,j}$ is the cost of the minimum cost route between node i and j. Different $\mathbf{L}$ matrices can be defined for the same network starting from different metrics. The Shimbel matrix [76] can be obtained as a special case of the valued-graph matrix using hop count as the link cost metric. The link cost metric that we employ is unidirectional ETX $\lambda_{i,j} = 1/\pi_{i,j}$, which provides a good compromise between hop count and reliability.

Link robustness matrix. We define $\mathbf{G} = \{G_{i,j}\}$ such that $G_{i,j} = L_{i,j}^{-1}$ if $L_{i,j} > 0$ and $G_{i,j} = 1$ if $L_{i,j} = 0$ to avoid unwieldy infinite terms; we will refer to $\mathbf{G}$ as the link robustness matrix. Each element $G_{i,j}$ can be viewed as the quality of the generalized link $[i, j]$.

Network bottlenecks If a large number of nodes contend to communicate to a small number of their downstream peers, a network bottleneck is formed. In large networks with low-density areas, it is not infrequent that a subset of the nodes can only communicate to the sink by way of a few gateway nodes. In the worst case, one particular node is the only way for a given subnetwork to reach the sink. Using transport geography, we can develop an indication of whether a given node represents a communication bottleneck due to the topology of the network. Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be two clusters of nodes, subsets of $\mathcal{N}$, and assume a unidirectional packet flow from $\mathcal{C}_1$ to $\mathcal{C}_2$ (assume that $s \in \mathcal{C}_2$: all nodes in $\mathcal{C}_1$ want to communicate to $\mathcal{C}_2$, but not vice-versa). We define the *exit potential* for node $i \in \mathcal{C}_1$ into $\mathcal{C}_2$ as

$$\omega_i = \sum_{j \in \mathcal{C}_2} G_{i,j}. \tag{5.2}$$

The exit potential quantifies the ability of node $i \in \mathcal{C}_1$ to get packets into $\mathcal{C}_2$. Likewise, we quantify the ability of node $j \in \mathcal{C}_2$ to receive packets from $\mathcal{C}_1$ by defining the *entry potential* for node $j \in \mathcal{C}_2$ into $\mathcal{C}_2$ as

$$\epsilon_j = \sum_{i \in \mathcal{C}_1} G_{i,j}. \tag{5.3}$$

5.3 Topological bottlenecks: a transport geographical interpretation

In the presence of bottlenecks, there exist topological constraints that make it impossible to completely balance the load: some nodes get overloaded independently of the routing and load balancing strategy. We employ transport geography to analyze two examples of topology-constrained load balancing in the 155-node MoteLab testbed.

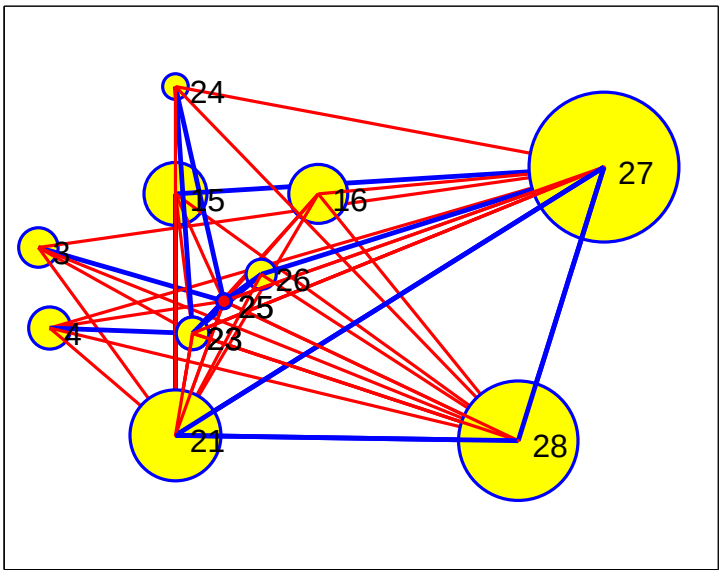
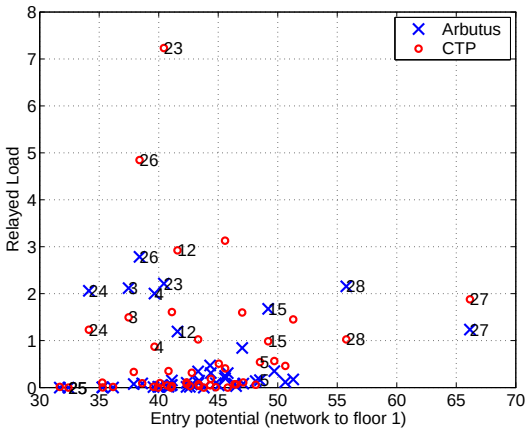
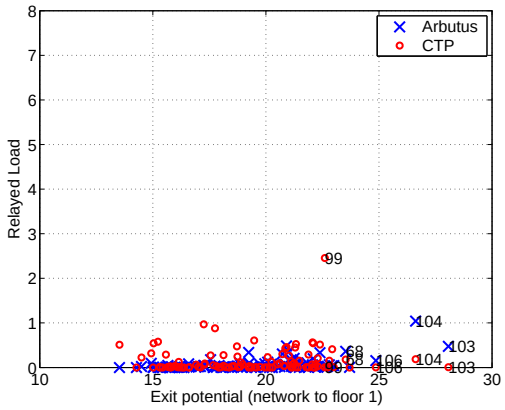


Figure 5.1. Basic layout of the critical set of sink 25. The shading of the link is proportional to their PDR; lighter-shaded links are much less reliable than darker-shaded ones. The size of the nodes is proportional to their degree.



(a) Entry potentials under light load.



(b) Exit potentials under light load.

Figure 5.2. Sink assignment 25 in the 155-node testbed: exit and entry potentials into floor 1 from the rest of the network.

Case study: very small critical set

Sink assignment 25 on MoteLab's floor 1 has a very small critical set. Figure 5.1 shows that node 25 only has stable links to low-degree nodes, and unstable links to higher-degree nodes such as 27, 28, and 21. Figure 4.26(a) shows that nodes 27 and 28 are major floor 1 gateways, and Figure 5.2(a)) shows the transport-geographical reason for this: node 27 and 28 have the highest entry potential of all floor 1 nodes. Therefore, as we go back to Figure 4.16, it is not surprising

to see that sink assignment 25 yields the worst goodput among all the topologies we considered. Moreover, despite the typically consistent reliability exhibited by Arbutus across different topologies, we have seen in Figure 4.10 that topology 25 yields one of the worst reliability performances. Intuitively, a sink with a small degree, whose neighbors in turn also have relatively low degrees, is guaranteed to be affected by the presence of topological bottlenecks and heavy congestion. The excellent load balancing fairness of this topology seen in Figure 4.25 is only due to the fact that most of the load does not make it past bottlenecks 27 and 28, due to their poor connectivity to 25. This is true for both protocols, and is mirrored by the poor goodput as well as the very small relayed load imposed on nodes outside of floor 1 shown in Figure 5.2(b), which indicates that a relatively small fraction of the offered load from floors 2 and 3 makes it to the sink, independently of the protocol. Under heavy load, for example, 58% of all packets delivered to the sink with Arbutus are from floor 1, 27% are from floor 2, and only 14% are from floor 3. CTP achieves similar figures: 64% from floor 1, 23% from floor 2, 12% from floor 3. Note that 40% of the nodes are on floor 2, while the remaining nodes are half on floor 1 and half on floor 3.

Thanks to its long-hop routing approach Arbutus is topology-aware: Figure 5.2 shows the transport-geographical optimality of Arbutus's load distribution. Only topological bottlenecks (27, 28, 103, 104) are overloaded, while CTP overloads nodes 23 and 26, though their entry potential is small compared to more than half of the nodes. The same is true of node 99, overloaded by CTP despite its comparatively small exit potential.

The sink assignment is itself transport-geographically inefficient: the sink, node 25, is shown in Figure 5.2(a) to have one of the lowest entry potentials among all floor 1 nodes.

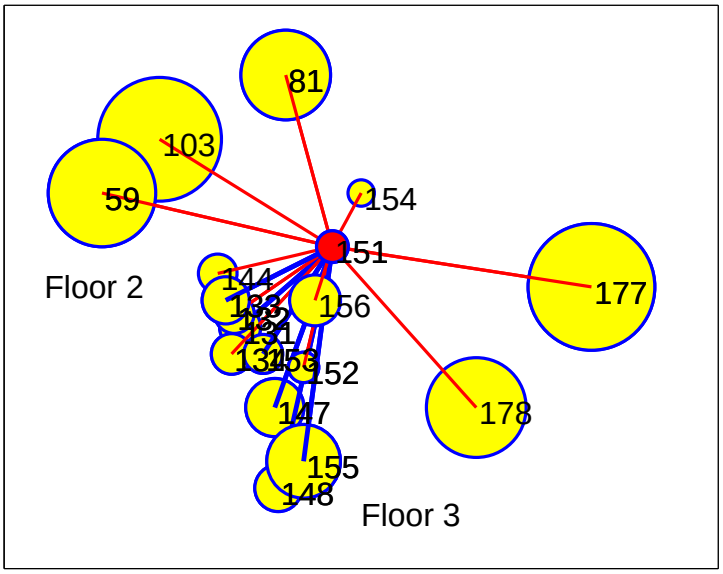
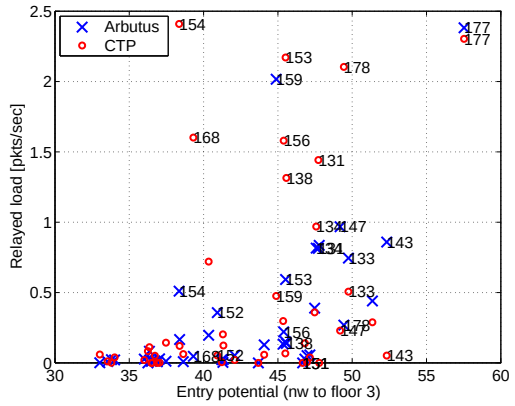


Figure 5.3. Basic layout of the critical set of sink 151. The shading of the link is proportional to their PDR; lighter-shaded links are much less reliable than darker-shaded ones. The size of the nodes is proportional to their degree.

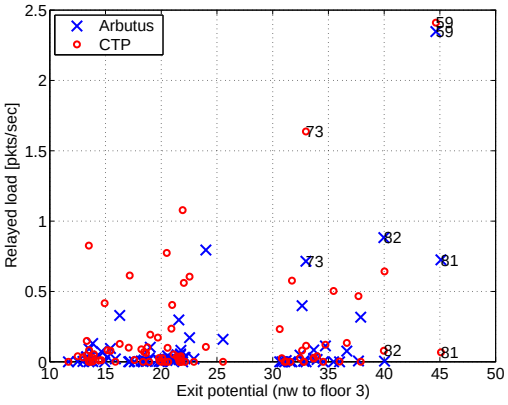
5.3.1 Case study: bottlenecks in the critical set

Sink assignment 151, located on floor 3, was the subject of our case study for routing performance as a function of offered load in Section 4.6. Sink 151 has a relatively large critical set, but most of its critical nodes are also on floor 3, and all communication with the other floors (1 and 2) occurs by way of very few topological bottlenecks, as shown in Figure 5.3. In particular, point-to-point measurements showed us that node 59, located on floor 2, is a major gateway to floor 2 for floor 1 nodes.

Figure 5.4 provides an overview of entry and exit potentials for this particular topology. Node 59 is clearly a topological bottleneck: it is equally overloaded by each protocol, and has the highest exit potential among all nodes outside of floor 3, as shown in Figure 5.4(a). Likewise, node 177 is a topological bottleneck within floor



(a) Entry factors at $f_{\text{gen}} = 0.1$ pkts/sec.



(b) Exit factors at $f_{\text{gen}} = 0.1$ pkts/sec.

Figure 5.4. Entry and exit potential from floors 1 and 2 into floor 3 under light offered load.

3: it has the largest entry potential, and it is equally exploited by both protocols, as shown in Figure 5.4(b). Figure 5.4(a) also shows that node 151 has itself a mid-range entry potential, which makes it a suboptimal sink assignment (but certainly much better than 25).

CHAPTER 6

LIFETIME BENEFITS OF LOAD BALANCING

6.1 Generalities

Motivation. In load balancing, the idea is to occasionally leverage on suboptimal links in alternative to the most reliable ones to ensure a more uniform usage of the energy resources across the network [77]. Always choosing the most reliable relays is a greedy form of data collection aimed at mining the network as much as possible; while using only reliable channels avoids energy waste due to packet loss, it also drains the resources of the nodes that offer the most reliable channels. On the contrary, the alternate use of optimal and suboptimal relays represents a more sustainable form of data harvesting based on giving up on some of the collected data now in order to continue collecting data for a longer time: load balancing is the WSN equivalent of what ecologists call *sustainable development*. In this section, as a complement to the performance analysis in Chapters 4 and 5, we present the results of an experimental study on the lifetime benefits of load balancing. In the first part of the study, we estimate the lifetime gain provided by Arbutus over a baseline protocol. In the second part, we emulate battery depletion and we measure the lifetime gain.

Benchmark. We choose MultiHopLQI as a baseline. Key differences between Arbutus and MultiHopLQI lie in the tree construction process, link estimation, and

route selection. In MultiHopLQI, a parent is chosen if it offers a better route to the sink; link costs are uniquely based on LQI and added 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.

For the experimental work in this chapter, it is important to note that Arbutus and the baseline operate in a best-effort mode, with no ARQ and no congestion control. An early version of Arbutus is used, with two main differences with respect to the complete version used in Chapters 4 and 5: no data plane feedback is employed (since no ARQ is used), and CSI-based blackslisting is enforced (links with relatively low values of RSS and LQI are avoided).

Testbed. The work in this chapter was carried out on the MoteLab testbed in the summer and fall of 2007, at a time when the testbed was not completely stable, and periodic node failures were relatively frequent. The exact number of working nodes at any given time was subject to large fluctuations. The topology of the testbed was heavily clustered, often with rather scarce inter-cluster connectivity, which made the testbed extremely challenging and therefore ideal for the adversarial evaluation of routing protocols. The testbed was completely overhauled after February 2008, which is when the experimental work shown in Chapters 4 and 5 was carried out.

In general, the maximum possible coverage varies depending on the time of the experiment and the sink assignment: like all real-world WSNs, the MoteLab testbed has time-varying connectivity properties. The environment where the nodes are deployed is continuously modified by human activity, and even small changes (such as repositioned furniture or open doors) cause major fluctuations in the multipath

fading patterns.

6.2 Estimated lifetime gain

6.2.1 Cost-to-benefit ratio

We intend to benchmark Arbutus against MultiHopLQI to ascertain that the use of suboptimal links does not deteriorate the overall performance of the protocol. We compare the two protocols with respect to routing performance metrics such as average best-effort[1] delivery ratio at the sink (computed as the ratio of the sum of the packets received from all nodes to the number of nodes) and average 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 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. We define the cost-to-benefit ratio η as

$$\eta \triangleq \frac{\sum_{i \in \mathcal{N}} \overline{\beta}_i}{\sum_{i \in \mathcal{N}} e_i}, \tag{6.1}$$

where $\overline{\beta}_i$ is the normalized relayed load (packets relayed per packet transmitted).

6.2.2 Experimental results on the MoteLab testbed

We first focus on a relatively small 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) for every node. 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. 6.1 shows the average delivery ratio at the sink, computed as the average of the delivery ratio for each node over the

[1] As we have already remarked, for the work in this chapter, Arbutus and the benchmark operate in a best-effort mode without retransmissions and congestion control.

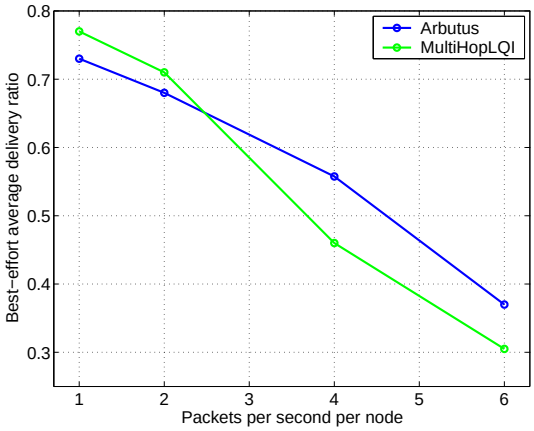


Figure 6.1. Average delivery ratio for Arbutus and MultiHopLQI. At low loads, the use of suboptimal links comes at the cost of a slightly lower delivery ratio.

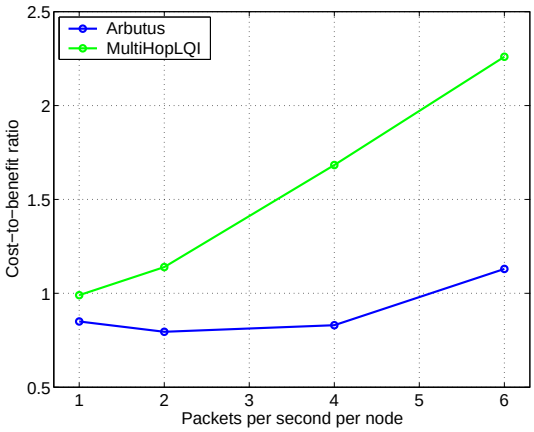


Figure 6.2. Cost-to-benefit ratio η for Arbutus and MultiHopLQI.

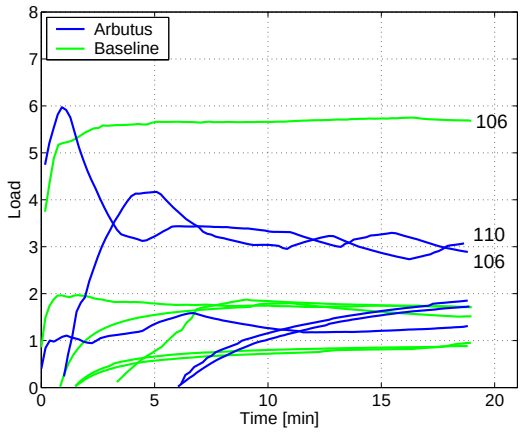


Figure 6.3. Time evolution of the load of all the relays in our sample experiment. The considerable difference in the workload of node 106 is due to the load balancing action of Arbutus reflected by the value of η (0.8 for Arbutus, 1.2 for the baseline protocol MultiHopLQI).

whole experiment, and Figure 6.2 shows the cost-to-benefit ratio η. These results 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 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 delivery ratio. In order to appreciate the meaning of η and its impact on performance, Figure 6.3 shows one instance of the time evolution of the load of each relay with both protocols. In this particular experiment ($f_{\text{gen}} = 4$ pkts/sec), $\eta = 0.8$ for Arbutus and $\eta = 1.2$ 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

Table 6.1

BENCHMARKING RESULTS ON THE COMPLETE MOTELAB TESTBED ($f_{\text{gen}} = 1$ PKT/SEC).

	Arbutus	Baseline	Ratio
Average number of nodes reached	77.5	71.7	1.08
Average fraction of nodes reached	0.88	0.80	1.10
Average delivery ratio	0.33	0.27	1.21
Average same-floor delivery ratio	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

also show in terms of delivery ratio (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. At the time of the experiments, the nodes were 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 6.1. The average delivery ratio is computed as the average of the delivery ratio for each working node in the testbed measured by the sink over the whole experiment. Similarly, the average same-floor delivery ratio 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

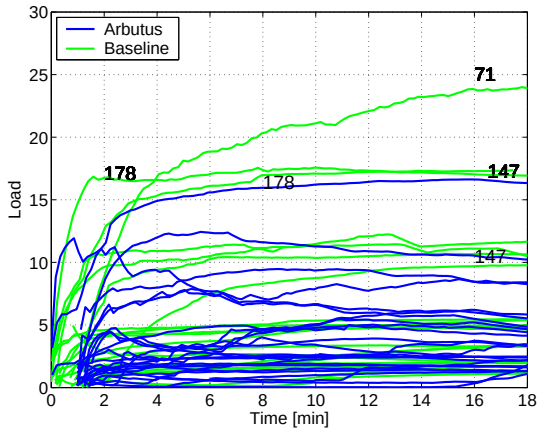


Figure 6.4. Time evolution of the load of all relays in a sample experiment with the complete MoteLab testbed.

number of working nodes over all experiments is 89), this benchmarking serves as a confirmation of 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 or queue overflow) 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 [26] is the main reason for the significant differences between day and night experiments; results also vary considerably depending on the particular sink assignment. Over our experiments, the standard deviation σ on η is 2.5 for Arbutus and 2.8 for MultiHopLQI, whereas the σ of the average delivery ratio is 0.06 for both protocols.

Figure 6.4 shows the time evolution of the load of each relay in one of the experiments run on the complete MoteLab testbed. The designated sink is located on the 2nd floor of the building. With Arbutus the node with the highest workload

is 178, which represents a topological bottleneck (it happens to be a major gateway into the second floor for third floor traffic); in other words, any routing scheme has to pass through 178. Indeed, node 178 gets even more use with MultiHopLQI, which however imposes the heaviest workload on another node on the 2nd floor, 71, located at depth 7 in MultiHopLQI's tree and definitely not a topological bottleneck (its load with Arbutus is only 0.8). The structure imposed by Arbutus to the tree construction process eliminates non-topological protocol-induced bottlenecks (such as node 71 with MultiHopLQI), with the key advantage that no nodes get depleted unnecessarily.

6.3 Measured lifetime gain

6.3.1 Energy depletion emulation

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.

Our goal is to verify that the reduction in the cost-to-benefit ratio indeed corresponds to a lifetime gain. Since all nodes in the MoteLab testbed are wall-powered, we need to emulate the energy depletion process (on all nodes other than the sink). The energy expenditure of the MCU is similar across protocols of comparable complexity, and sensing energy is obviously independent of the routing protocol. Therefore, we emulate lifetime evolution by only factoring in the radio-related energy expenditure. Every node is assigned the same energy credit X; a credit unit is lost

for every packet sent or received. It is essential to note that *every packet counts*: both data packets and control packets are factored in, no matter whether they have been sent, received, or overheard. A packet sent by node a is *received* by node b if it can be correctly demodulated (as guaranteed, for instance, by a cyclic redundancy check) by b and b is indeed its intended receiver ($b = p(a)$). A packet sent by node a is *overheard* by b if it can be correctly demodulated by b but b is not its intended receiver ($b \neq p(a)$). Once a node has exhausted 95% of its credit, it warns its upstream neighbors (child nodes) of its upcoming demise by advertising parent loss; this is done to avoid unnecessary packet losses on the part of the child nodes after an emulated depletion event at the parent.

6.3.2 Figures of merit

Lifetime. Following [78], we define network $\eta-$lifetime as the time between the inception of network operation and the moment when the sink ceases to receive sensor data from at least a fraction η of the nodes. Other noteworthy lifetime indicators are the minimum node lifetime (time when the earliest node depletion occurs) and the mean node lifetime.

Total delivered data (TDD). The TDD is the total number of packets received by the sink; it is an excellent indicator of the overall $\eta-$lifetime performance of a network. Given that the application is continuous data collection from all nodes, the TDD is proportional to the network lifetime.

Coverage. For the analysis in this chapter, we use a different notion of coverage, which we now define as the number of nodes from which the sink receives at least 100 packets (over the course of the emulated lifetime).

6.3.3 System operating point and implementation details

We benchmark our protocol against MultiHopLQI, for which we use the code in the TinyOS distribution (complemented by an implementation of the same depletion emulation scheme that we use for Arbutus). Since load balancing is most needed when the traffic load is high, we operate at a high traffic generation rate: each node in the network is a source and generates data packets at a rate f_{gen} of 4 packets per second. This causes a large amount of in-network interference as well as congestion; again, Arbutus and the benchmark purposefully operate in best-effort mode, with no ARQ and no congestion control.

Our operating point causes congestion and in-network interference: not only are there lossy links, but also ingress drops, and therefore heavy packet loss is to be expected. For our benchmarking, we focus on the ratio between the performance indicators (the figures of merit in 6.3.2) of the two protocols as we run them back-to-back one after the other to ensure reasonable continuity in the connectivity properties of the network. Given the constraints on MoteLab use (it is a shared resource with a user quota allocation policy), we employ a credit of $X = 2000$ packets for depletion emulation, which allows us to emulate network depletion in slightly less than 10 minutes.

6.3.4 Proof of concept

As a proof of concept, we show an example of the performance of Arbutus and the baseline protocol on the MoteLab testbed. The particular sink assignment is node 134, which lies within a relatively dense cluster that at the time of the experiment had virtually no connectivity to the rest of the network (as confirmed by point-to-point connectivity measurements taken prior to and after the routing experiments). Figure 6.5 shows the evolution of the TDD as routing is underway

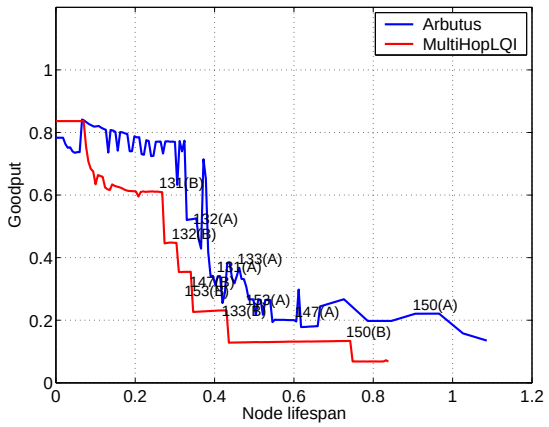


Figure 6.5. Comparative total data delivery (TDD) performance of Arbutus and the baseline protocol on a 14-mote network (subset of the MoteLab testbed).

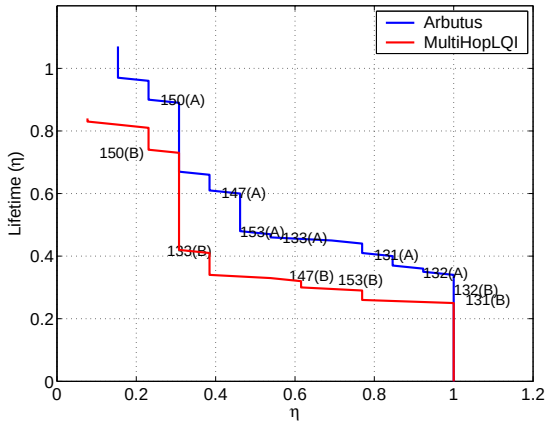


Figure 6.6. The $\eta-$lifetime profile shows that individual nodes live substantially longer with our lifetime-aware routing protocol.

Table 6.2

OVERALL PERFORMANCE METRICS FOR THE EXPERIMENTAL PROOF OF CONCEPT.

	Arbutus	Baseline
TDD [packets]	8691	6933
Coverage [nodes]	13	12
Minimum node lifetime	0.35	0.26
Mean node lifetime	0.69	0.39
Control overhead	0.05	0.10

and energy depletion is emulated. Qualitatively, what happens is that Arbutus distributes the load over all the nodes, while the baseline protocol overloads the nodes that offer the best channel to the sink, namely 131 and 132, which have the shortest lifetime along with nodes that depend on them for connectivity to the rest of the network. Figure 6.6 shows the value of the $\eta-$lifetime as the function of the fraction η of responding nodes. Given a value of η, we can read the amount of time that a fraction η of the 13 nodes in the network respond to the sink. By extending the lifespan of the individual nodes, Arbutus always has a better or equal $\eta-$lifetime profile with respect to the baseline. More details can be found in Table 6.2. The 54% TDD gain of Arbutus over the baseline is considerable, and is reflected by a significant lifetime gain: the mean node lifetime is increased by 77%, and the earliest depletion time for a node (minimum node lifetime) increases by 35%. The use of suboptimal links, however, has a small cost that shows in the form of a 7% reduction of the average per-node delivery ratio. The 50% reduction of the control overhead is another considerable benefit of our approach, and it is mostly due to the more efficient organization of the routing tree.

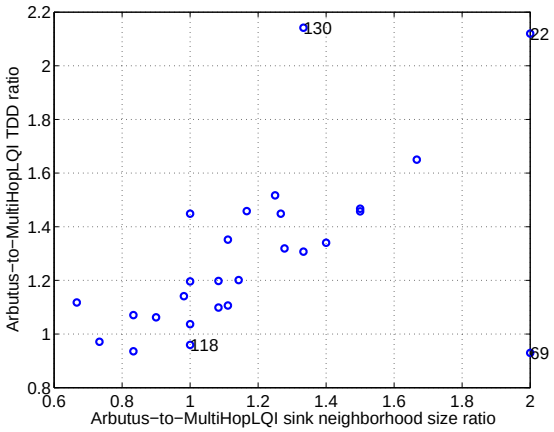


Figure 6.7. The extension of the sink neighborhood is the main mechanism whereby Arbutus achieves a load balancing gain.

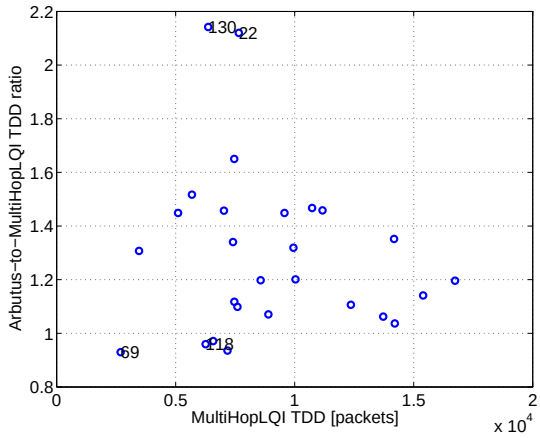


Figure 6.8. The baseline only outperforms Arbutus in terms of TDD if scarce connectivity in the network causes heavy packet loss, thus reducing the traffic load.

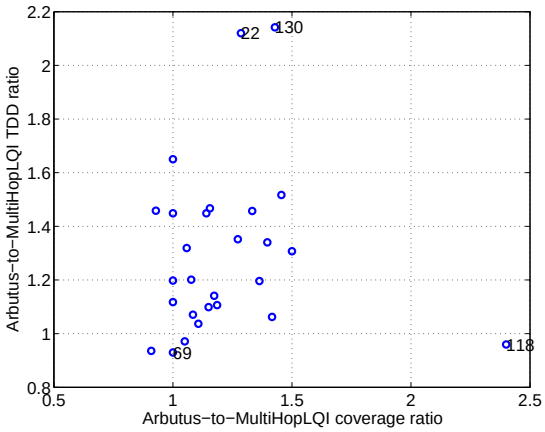


Figure 6.9. Load balancing typically comes with a significant coverage benefit, as shown by this plot of the Arbutus-to-baseline TDD ratio versus the coverage ratio.

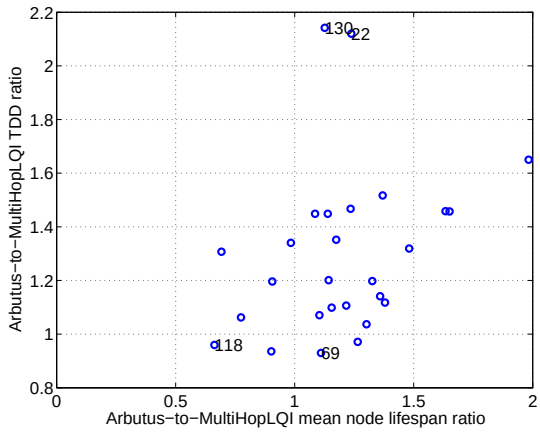


Figure 6.10. It is intuitively pleasing that, as a general trend, an increase in TDD comes with an increase in the mean node lifespan.

Table 6.3

OVERALL PERFORMANCE METRICS FOR THE EXPERIMENTAL EVALUATION ON MOTELAB (A INDICATES ARBUTUS AND B THE BASELINE).

A-B TDD ratio - average	1.30
A-B TDD ratio - standard deviation	0.31
A-B Coverage ratio - average	1.22
A-B Coverage ratio - standard deviation	0.29
A-B Earliest node depletion time ratio - average	1.18
A-B Earliest node depletion time ratio - standard deviation	0.50
A-B Mean node depletion time ratio - average	1.49
A-B Mean node depletion time ratio - standard deviation	0.41
A-B Control overhead ratio - average	0.87
A-B Control overhead ratio - standard deviation	0.23

Table 6.4

ARBUTUS PROVIDES BOTH A COVERAGE GAIN AND A TDD GAIN IN 78% OF OUR EXPERIMENTS.

Fraction of experiments with TDD gain	0.81
Fraction of experiments with coverage gain	0.93
Fraction of experiments with coverage gain and TDD gain	0.78
Fraction of experiments with coverage loss and TDD gain	0.04
Fraction of experiments with TDD loss and coverage gain	0.15
Fraction of experiments with coverage loss and TDD loss	0.04

6.3.5 Performance evaluation and benchmarking

We chose 25 different sink assignments and ran a total of over 50 experiments on MoteLab. We average values from different experiments with the same sink assignment, and finally take the sample mean of the average values for the individual sink assignments, which are shown throughout this section. We express our figures of merit as the ratio of Arbutus's performance over the baseline's. Table 6.3 shows the results of the benchmarking of our protocol against the baseline, MultiHopLQI. The main result is that, on average, the use of load balancing increases the TDD (and therefore the network lifetime) by 31%, and the coverage by 22%.

The lifetime benefits are significant: on average, load balancing delays the earliest depletion event by 18% and increases the life expectancy of a node by as much as 49%. Only nodes that do get depleted are considered (nodes that are far enough from the sink may not get depleted at all because their gateways to the sink get depleted first, depriving them of a route to the sink). There is a relatively large variance in the results: the standard deviation is about 0.3 for both the TDD and the coverage ratio. Table 6.4 shows a breakdown of the performance results based on what particular benefit (TDD or coverage gain) is or is not achieved. With most sink assignments, Arbutus provides both a TDD and a coverage gain.

Given a sink assignment, performance variations still occur because of network topology variations due to node failures. The performance variations over different sink assignments are, however, much more dramatic. In Figures 6.7-6.10, we represent various flavors of the principal figure of merit, the Arbutus-to-baseline TDD ratio (averaged over all datasets with the same sink assignment). To facilitate our interpretation, the particular sink assignment is shown for four outliers that stand out in different ways. While 22 and 130 always correspond to an outstanding performance (with respect to TDD, coverage, and node lifespan), 69 and 118 are

examples of undesired behaviors of Arbutus, and will be the focus of our discussion. Figure 6.7 shows a definite correlation between sink neighborhood size and TDD: the more Arbutus extends the number of sink's neighbors, the larger its TDD gain with respect to the baseline. The sink's neighborhood contains the aforementioned critical nodes that have to carry the weight of the whole network as they are called to relay data packets from their upstream peers. Increasing the number of such critical nodes for load redistribution (to make them less critical) means using suboptimal links to reduce the hop count of some routes, which normally pays off through the decongestion of the best relays. The presence of an outlier, 69, shows a slight TDD loss despite the doubling of the size of the critical area; this is an example where the use of suboptimal links does backfire. Figure 6.8 shows the Arbutus-to-baseline TDD ratio as a function of the TDD of the baseline protocol. It is not surprising to discover, in Figure 6.8, that sink assignment 69 corresponds to the smallest TDD over all the sink assignments that we employ (in this case the TDD is even smaller with Arbutus, as shown by the Arbutus-Baseline TDD ratio below 1 in Figure 6.7). Due to the network topology seen from node 69 (with the particular connectivity conditions of the testbed when the experiments were run), sink assignment 69 only yields less than half the average TDD over all sink assignments. Under such lighter traffic, the critical area is not so critical and the use of suboptimal links becomes (slightly) detrimental. To further prove our point, all other sink assignments that cause Arbutus to incur in a slight TDD loss lie at low volumes of sink-bound traffic (*e.g*, node 118). Figure 6.9 confirms Table 6.4 by showing that a coverage gain usually accompanies a TDD gain. With outlier 118 we have a relatively small volume of acquired packets (see Figure 6.8) but no sink neighborhood extension (see Figure 6.7): although Arbutus's tree structure improves coverage, the topology is the bottleneck, and the additional packets from the extra nodes that get covered

drain the very limited relaying resources of the critical area. Figure 6.10 shows the TDD ratio as a function of the mean node depletion time. It is not surprising to find that Arbutus has the worst lifetime performance with sink assignment 118. It should also be noted the general trend whereby an increase in the mean node lifespan corresponds to a larger TDD, which is intuitively pleasing and comes as a confirmation of the long term benefits of load balancing.

CHAPTER 7

CONCLUSIONS

Long hops. Our extensive experimental results, obtained on large-scale public testbeds of low-end sensing nodes, provide further evidence that long-hop routing is generally more efficient than breaking long hops into shorter hops, especially under heavy load.

Reliability. Users considering applications for which reliability is the most important performance dimension should consider using Arbutus independently of the offered load or the topology, as Arbutus's reliability has been shown to always outpeform CTP's. A key property is the fact that the delivery ratio of Arbutus is, for the most part, not sensitive to either offered load or network topology. Another appealing property is that the performance gap between Arbutus and CTP widens as the offered load increases.

Scalability. The offered load does not only depend on the application, but also on the network size. Since the performance gap between Arbutus and CTP widens as the offered load increases, Arbutus scales better and is a better choice for large networks.

Load balancing. Arbutus enforces load balancing whenever possible; the larger the critical set, the better the results. Aside from reducing congestion and improving goodput and reliability, load balancing has a direct impact on network lifetime. If the network load is balanced, then nodes are generally not overloaded, and battery depletion rates are fairly constant across the network. By emulating battery depletion, we have measured that Arbutus's load balancing yields a 30% increase in network lifetime; this figure matches our estimate using cost-to-benefit ratio (packets relayed by i over packets delivered by i to the sink) as a predictor of network lifetime.

Node placement and sink selection. Our analysis has shown that, independently of the protocol, node placement has a huge impact on routing performance. Given a particular node placement pattern, sink selection is also critical. Twist's high-density regular grid-like topology, for instance, is conducive to a much better and more consistent routing performance across different topologies and offered load points. While in actual deployments there is often a limited amount of freedom in node placement and sink selection, care should nonetheless be exercised to ensure enough spatial diversity at the sink by way of a relatively large critical set (compared to network size). Ideally, every node should have a large critical set, to facilitate load balancing by providing path diversity and thus ensure a longer network lifetime.

Routing tables. Although Arbutus does not employ a routing table and only keeps state for the current parent (as well as the former one, in some cases), we have shown that it outperforms CTP in most dimensions. The main advantage of a routing table is that, when the current parent becomes unavailable due to congestion, a new parent can be looked up in the table. Arbutus, however, employs load

balancing, normally enforcing parent rotation; congestion, therefore, only occurs at topological bottlenecks, for which there are no alternatives by definition. Therefore, as confirmed by our experimental evidence throughout Chapter 4, it is not necessary to use a routing table in many-to-one low-power WSNs if load balancing is implemented.

Exploring the design space. The Arbutus architecture lends itself to further explorations of the routing design space. In particular, an adaptation of Arbutus for point-to-point routing over a mesh is also possible: a node would have to keep state for all its intended destination, which would be feasible due to the fact that Arbutus saves memory by not relying on a routing table.

Low-power operation. A future study could be devoted to investigating the operation of Arbutus on top of the Low Power Listening protocol (LPL) [79] or Low Power Probing [51]. A version of CTP with LPL is already available, and it would again serve as a benchmark. Given its energy-efficiency, it would be of particular interest to use Arbutus for backbone routing in a mostly-off sensor network where the use of sleep modes is widespread.

BIBLIOGRAPHY

[1] E. Shih, S. Cho, N. Ickes, R. Min, A. Sinha, A. Wang, and A. Chandrakasan. Physical layer driven protocol and algorithm design for energy-efficient wireless sensor networks. In *7th annual international conference on Mobile computing and networking (ACM MobiCom'04)*, Rome, Italy, July 2001.

[2] F. Stann and J. Heidemann. RMST: Reliable Data Transport in Sensor Networks. In *1st IEEE International Workshop on Sensor Net Protocols and Applications (SNPA'03)*, Anchorage, AK, USA, May 2006.

[3] M. Wachs, J. Choi, J. Lee, K. Srinivasan, Z. Chen, M. Jain, and P. Levis. Visibility: A New Metric for Protocol Design. In *5th ACM Conference on Embedded Networked Sensor Systems (ACM SenSys'07)*, Sydney, Australia, November 2007.

[4] K. Langendoen, A. Baggio, and O. Visser. Murphy Loves Potatoes: Experiences from a Pilot Sensor Network Deployment in Precision Agriculture. In *14th Intl. Workshop on Parallel and Distributed Real-Time Systems (WPDRTS'06)*, Rhodes, Greece, April 2006.

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

[6] Q. Wang, M. Hempstead, and W. Yang. A Realistic Power Consumption Model for Wireless Sensor Network Devices. In *Third IEEE International Conference on Sensor and Ad Hoc Communications and Networks (SECON'06)*, Reston, VA, USA, September 2006.

[7] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System Architecture Directions for Network Sensors. In *9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'00)*, Cambridge, MA, USA, November 2000.

[8] T. He, J. Stankovic, C. Lu, and T. Abdelzaher. A Spatiotemporal Communication Protocol for Wireless Sensor Networks. *IEEE Transactions on Parallel and Distributed Systems*, 16:995–1006, October 2005.

[9] G. Werner-Allen, P. Swieskowski, and M. Welsh. MoteLab: a Wireless Sensor Network Testbed. In *4th international symposium on Information Processing in Sensor Networks (IPSN'05)*, Los Angeles, CA, USA, April 2005.

[10] V. Handziski, A. Koepke, A. Willig, and A. Wolisz. TWIST: A Scalable and Reconfigurable Testbed for Wireless Indoor Experiments with Sensor Networks. In *2nd international workshop on Multi-hop Ad Hoc Networks: from Theory to Reality (REALMAN'06)*, Florence, Italy, 2006.

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

[12] J. Choi, J. Lee, M. Wachs, Z. Chen, and P. Levis. Fair Waiting Protocol: Achieving Isolation in Wireless Sensornets. In *5th ACM Conference on Embedded Networked Sensor Systems (ACM SenSys'07)*, Sydney, Australia, November 2007.

[13] J. Choi, J. Lee, M. Wachs, Z. Chen, and P. Levis. Opening the Sensornet Black Box. In *Proceedings of the International Workshop on Wireless Sensornet Architecture (WWSNA'07)*, Cambridge, MA, USA, April 2007.

[14] R. Fonseca, O. Gnawali, K. Jamieson, and P. Levis. Four-Bit Wireless Link Estimation. In *Sixth Workshop on Hot Topics in Networks (HotNets-VI)*, Atlanta, GA, USA, November 2007.

[15] J. Hauer, V. Handziski, A. Koepke, A. Willig, and A. Wolisz. A Component Framework for Content-based Publish/Subscribe in Sensor Networks. In *IEEE European Workshop on Wireless Sensor Networks (EWSN'08)*, Bologna, Italy, January 2008.

[16] N. Kothari, T. Millstein, and R. Govindan. Deriving State Machines from TinyOS Programs using Symbolic Execution. In *7th international symposium on Information Processing in Sensor Networks (IPSN'08)*, St Louis, MO, USA, April 2008.

[17] L. Filipponi, S. Santini, and A. Vitaletti. Data Collection in Wireless Sensor Networks for Noise Pollution Monitoring. In *"International Conference on Distributed Computing in Sensor Systems (DCOSS'08)"*, Santorini, Greece, June 2008.

[18] J.-P. Rodrigue, C. Comtois, and B. Slack. *The Geography of Transport Systems.* Routledge Education, Oxon, UK, 2006.

[19] A. Cerpa, J. Wong, M. Potkonjak, and D. Estrin. Temporal Properties of Low Power Wireless Links: Modeling and Implications on Multi-Hop Routing. In *"ACM/IEEE Fourth International Conference on Information Processing in Sensor Networks (IPSN'05)"*, Los Angeles, CA, USA, April 2005.

[20] J. Zhao and R. Govindan. Understanding Packet Delivery Performance in Dense Wireless Sensor Networks. In *1st International Conference on Embedded Networked Sensor Systems (SenSys'03)*, Los Angeles, CA, USA, November 2003.

[21] K. Srinivasan and P. Levis. RSSI is Under Appreciated. In *Third Workshop on Embedded Networked Sensors (EmNets'06)*, Cambridge, MA, USA, May 2006.

[22] D. Ganesan, B. Krishnamachari, A. Woo, D. Culler, and D. Estrin. Complex Behavior at Scale: An Experimental Study of Low-power Wireless Sensor Networks. Technical Report UCLA/CSD-TR 02-0013, UCLA, 2002.

[23] M. Zuniga and B. Krishnamachari. An Analysis of Unreliability and Asymmetry in Low-Power Wireless Links. *ACM Transactions on Sensor Networks*, 3(2):1–30, 2007.

[24] D. Puccinelli and M. Haenggi. Multipath Fading in Wireless Sensor Networks: Measurements and Interpretation. In *International Wireless Communications and Mobile Computing Conference (IWCMC'06)*, Vancouver, BC, Canada, July 2006.

[25] K. Srinivasan, M. Kazandjieva, S. Agarwal, and P. Levis. The Beta-Factor: Improving Bimodal Wireless Networks. Technical Report SING-07-01, Stanford University, 2007.

[26] 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.

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

[28] O. Gnawali, M. Yarvis, J. Heidemann, and R. Govindan. Interaction of Retransmission, Blacklisting, and Routing Metrics for Reliability in Sensor Network Routing. In *First IEEE Conference on Sensor and Adhoc Communication and Networks*, pages 34–43, Santa Clara, CA, USA, October 2004.

[29] A. Cerpa, N. Busek, and D. Estrin. SCALE: A tool for Simple Connectivity Assessment in Lossy Environments. Technical Report CENS Technical Report 0021, UCLA.

[30] E. N. Gilbert. Random Plane Networks. *Journal of SIAM*, 9:533–543, October 1961.

[31] D. Puccinelli and M. Haenggi. Wireless Sensor Networks-Applications and Challenges of Ubiquitous Sensing. *IEEE Circuits and Systems Magazine*, 5:19–29, August 2005.

[32] J. Al-Karaki and A. Kamal. Routing Techniques in Wireless Sensor Networks: A Survey. *IEEE Wireless Communications*, 11:6–28, Dec 2004.

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

[34] D. Culler, P. Dutta, C. Ee, R. Fonseca, J. Hui, P. Levis, J. Polastre, S. Shenker, I. Stoica, G. Tolle, and J. Zhao. Towards a Sensor Network Architecture: Lowering the Waistline. In *10th International Workshop on Hot Topics in Operating Systems (HotOS'05)*, Santa Fe, NM, USA, June 2005.

[35] C. Ee, R. Fonseca, S. Kim, D. Moon, A. Tavakoli, D. Culler, S. Shenker, and I. Stoica. A Modular Network Layer for Sensorsets. In *7th symposium on Operating systems design and implementation (OSDI'06)*, Seattle, WA, USA, 2006.

[36] R. Poor. Gradient Routing in Ad Hoc Networks. 2000. Available at `http://www.media.mit.edu/pia/Research/ESP/texts/poorieeepaper.pdf`.

[37] B. Krishnamachari, D. Estrin, and S. Wicker. Modelling Data-Centric Routing in Wireless Sensor Networks. In *Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'00)*, Tel Aviv, Israel, March 2000.

[38] A. Woo. *A Holistic Approach to Multihop Routing in Sensor Networks*. PhD thesis, University of California at Berkeley, Fall 2004.

[39] D. Ganesan, R. Govindan, S. Shenker, and D. Estrin. Highly Resilient, Energy Efficient Multipath Routing in Wireless Sensor Networks. *ACM Mobile Computing and Communication Review*, 1(2):10–24, 2002.

[40] C. Perkins and P. Bhagwat. Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers. In *SIGCOMM'94*, London, UK, August 1994.

[41] R. Bellman. On a Routing Problem. *Quarterly of Applied Mathematics*, 16(1):87–90, January 1958.

[42] E. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1:269–271, 1959.

[43] M. Yarvis, W. Conner, L. Krishnamurthy, J. Chhabra, B. Elliott, and A. Mainwaring. Real-world Experiences with an Interactive Ad Hoc Sensor Network. In *International Conference on Parallel Processing Workshops (ICPP'02)*, Vancouver, BC, Canada, August 2002.

[44] C. Perkins and E. Royer. Ad hoc On-Demand Distance Vector Routing. In *2nd IEEE Workshop on Mobile Computing Systems and Applications*, New Orleans, LA, USA, February 1999.

[45] D. Johnson, D. Maltz, and J. Broch. DSR: The Dynamic Source Routing Protocol for Multi-Hop Wireless Ad Hoc Networks. In *Ad Hoc Networking, edited by Charles E. Perkins*, pages 139–172. Addison-Wesley, 2001.

[46] M. Pearlman and Z. Haas. Improving the Performance of Query-based Routing Protocols through Diversity Injection. In *IEEE Wireless Communications and Networking Conference (WCNC'99)*, New Orleans, LA, USA, September 1999.

[47] M. Pearlman and Z. Haas. Performance of routing protocols for real wireless sensor networks. In *International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS 2007)*, San Diego, CA, USA, July 2007.

[48] W. Zhao, M. Ammar, and E. Zegura. Adaptive Demand-Driven Multicast Routing in Multi-Hop Wireless Ad Hoc Networks. In *The Second ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'01)*, Long Beach, CA, USA, October 2001.

[49] B. Chen, K. Muniswamy-Reddy, and M. Welsh. Ad-Hoc Multicast Routing on Resource-Limited Sensor Nodes. In *Second International Workshop on Multi-hop Ad Hoc Networks: from Theory to Reality*, Florence, Italy, May 2006.

[50] T. Stathopoulos, M. Lukac, D. McIntire, J. Heidemann, D. Estrin, and W. Kaiser. End-to-End Routing for Dual-Radio Sensor Networks. In *" 26th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'07)"*, Anchorage, AK, March 2007.

[51] R. Musaloiu-E., C. Liang, and A. Terzis. Deriving State Machines from TinyOS Programs using Symbolic Execution. In *7th international symposium on Information Processing in Sensor Networks (IPSN'08)*, St Louis, MO, USA, April 2008.

[52] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva. Directed Diffusion for Wireless Sensor Networking. *ACM/IEEE Transactions on Networking*, 11:2–16, 2000.

[53] J. Heidemann, F. Silva, and D. Estrin. Matching Data Dissemination Algorithms to Application Requirements. In *The First ACM Conference on Embedded Networked Sensor Systems (SenSys'03)*, Los Angeles, CA, USA, November 2003.

[54] L. Girod, J. Elson, A. Cerpa, T. Stathopoulos, N. Ramanathan, and D. Estrin. EmStar: a Software Environment for Developing and Deploying Wireless Sensor Networks. In *The First Symposium on Networked Systems Design and Implementation (USENIX'04)*, San Francisco, CA, USA, March 2004.

[55] Y. Kim, R. Govindan, B. Karp, and S. Shenker. Lazy Cross-Link Removal for Geographic Routing. In *4th International Conference on Embedded Networked Sensor Systems (SenSys'06)*, Boulder, CO, USA, November 2006.

[56] Y. Kim, R. Govindan, B. Karp, and S. Shenker. On the Pitfalls of Geographic Face Routing. In *Workshop on Foundations of Mobile Computing (DIAL M-POMC'05)*, Cologne, Germany, September 2005.

[57] Y. Kim, R. Govindan, B. Karp, and S. Shenker. Geographic Routing Made Practical. In *2nd Symposium on Networked Systems Design and Implementation (NSDI'05)*, Boston, MA, USA, May 2005.

[58] F. Kuhn, R. Wattenhofer, and A. Zollinger. Worst-Case Optimal and Average-Case Efficient Geometric Ad-Hoc Routing. In *The Fourth ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'01)*, Annapolis, MD, USA, June 2003.

[59] B. Karp and H. Kung. GPSR: Greedy Perimeter Stateless Routing for Wireless Networks. In *6th Annual International Conference on Mobile Computing and Networking (MobiCom'00)*, Boston, MA, USA, 2000.

[60] R. Fonseca, S. Ratnasamy, J. Zhao, C. Ee, D. Culler, S. Shenker, and I. Stoica. Beacon-Vector Routing: Scalable Point-to-Point Routing in Wireless Sensor Networks. In *"2nd Symposium on Networked Systems Design and Implementation (NSDI'05)"*, Boston, MA, USA, May 2005.

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

[62] 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.

[63] C. Wan, S. Eisenman, A. Campbell, and J. Crowcoft. Overload Traffic Management in Sensor Networks. *ACM Transactions on Sensor Networks*, 3(4), October 2007.

[64] L. Popa, A. Rostamizadeh, R. Karp, C. Papadimitriou, and I. Stoica. Balancing Traffic Load in Wireless Networks with Curveball Routing. In *8th ACM international symposium on Mobile ad hoc networking and computing (MobiHoc'07)*, Montreal, QC, Canada, September 2007.

[65] D. Puccinelli and M. Haenggi. Arbutus: Network-layer load balancing for wireless sensor networks. In *IEEE Wireless Communications and Networking Conference (WCNC'08)*, Las Vegas, NV, USA, March 2008.

[66] K. Ramakrishnan and R. Jain. A Binary Feedback Scheme for Congestion Avoidance in Computer Networks. *ACM Transactions on Computer Systems*, 8(2):158–181, May 1990.

[67] C. Wan, S. Eisenman, and A. Campbell. CODA: Congestion Detection and Avoidance in Sensor Networks. In *1st international conference on Embedded networked sensor systems (SenSys'03)*, Los Angeles, CA, USA, 2003.

[68] A. Woo and D. Culler. A Transmission Control Scheme for Media Access in Sensor Networks. In *7th International Conference on Mobile Computing and Networking (MobiCom'01)*, Rome, Italy, May 2001.

[69] B. Hull, K. Jamieson, and H. Balakrishnan. Mitigating Congestion in Wireless Sensor Networks. In *2nd ACM Conference on Embedded Networked Sensor Systems (ACM SenSys'04)*, Baltimore, MD, USA, November 2004.

[70] L. Popa, C. Raiciu, I. Stoica, and D. Rosenblum. Reducing Congestion Effects in Wireless Networks by Multipath Routing. In *IEEE International Conference on Network Protocols (ICNP'06)*, Santa Barbara, CA, USA, November 2006.

[71] D. Puccinelli and M. Haenggi. DUCHY: Double Cost Field Hybrid Link Estimation for Low-Power Wireless Sensor Networks. In *Fifth Workshop on Embedded Networked Sensors (HotEmNets'08)*, Charlottesville, VA, USA, 2008.

[72] R. Jain. *The Art of Computer Systems Performance Analysis*. Wiley, 1991.

[73] K.Srinivasan, P. Dutta, A. Tavakoli, and P. Levis. Some Implications of Low Power Wireless to IP Networking. In *Fifth Workshop on Hot Topics In Networks (HotNets-V)*, Irvine, CA, USA, November 2006.

[74] D. Fortin A. Sona L. Angrisani, M. Bertocco. Assessing Coexistence Problems of IEEE 802.11b and IEEE 802.15.4 Wireless Networks through Cross-layer Measurements. In *IEEE Instrumentation and Measurement Technology Conference (IMTC'07)*, Warsaw, Poland, May 2007.

[75] R. Musaloiu-E. and A. Terzis. Minimising the effect of wifi interference in 802.15.4 wireless sensor networks. *International Journal of Sensor Networks*, 3(1):43–54, 2008.

[76] A. Shimbel. Structure in Communication Nets. In *Proc. of the Symposium on Information Networks*, 1955.

[77] J. Chang and L. Tassiulas. Energy Conserving Routing in Wireless Ad Hoc Networks. In *Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'00)*, Tel Aviv, Israel, March 2000.

[78] X. Liu and M. Haenggi. Towards Quasi-Regular Sensor Networks: Topology Control Algorithms for Improved Energy Efficiency. *IEEE Transactions on Parallel and Distributed Systems - Special Issue on Localized Communication and Topology Protocols for Ad Hoc Networks*, 17:975–986, September 2006.

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