Let the Tree Bloom: Scalable Opportunistic Routing with ORPL

Simon Duquennoy[1]
simonduq@sics.se

Olaf Landsiedel[2]
olafl@chalmers.se

Thiemo Voigt[1,3]
thiemo@sics.se

[1]SICS Swedish ICT AB, Sweden
[2]Chalmers University of Technology, Sweden
[3]Uppsala University, Sweden

ABSTRACT

Routing in battery-operated wireless networks is challenging, posing a tradeoff between energy and latency. Previous work has shown that opportunistic routing can achieve low-latency data collection in duty-cycled networks. However, applications are now considered where nodes are not only periodic data sources, but rather addressable end points generating traffic with arbitrary patterns.

We present ORPL, an opportunistic routing protocol that supports any-to-any, on-demand traffic. ORPL builds upon RPL, the standard protocol for low-power IPv6 networks. By combining RPL's tree-like topology with opportunistic routing, ORPL forwards data to any destination based on the mere knowledge of the nodes' sub-tree. We use bitmaps and Bloom filters to represent and propagate this information in a space-efficient way, making ORPL scale to large networks of addressable nodes. Our results in a 135-node testbed show that ORPL outperforms a number of state-of-the-art solutions including RPL and CTP, conciliating a sub-second latency and a sub-percent duty cycle. ORPL also increases robustness and scalability, addressing the whole network reliably through a 64-byte Bloom filter, where RPL needs kilobytes of routing tables for the same task.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Wireless Communication; C.2.2 [**Network Protocols**]: Routing Protocols

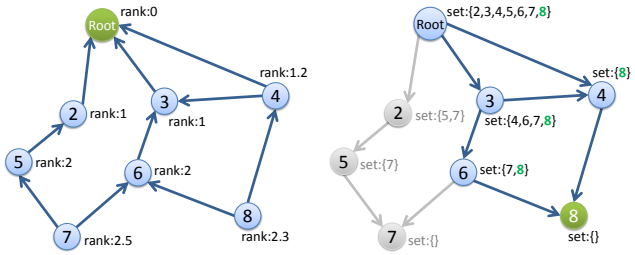
General Terms

Algorithms, Design, Experimentation, Performance

Keywords

Wireless Sensor Network, Energy Efficiency, Opportunistic Routing, RPL

SenSys'13, November 11–15, 2013, Rome, Italy.



(a) Routing Upwards (b) Routing Downwards

Figure 1: ORPL routes packets using multiple forwarders opportunistically. Routing upwards is done along a gradient. Routing downwards is done by going away from the root along nodes that have the destination in their routing set (either a bitmap or a Bloom filter).

1. INTRODUCTION

Over the past few years, a new era of sensor networks has emerged, where nodes are full-fledged addressable endpoints of a standards-based network, rather than data source of an ad-hoc periodic data collection. Applications are numerous: smart cities, building automation, smart grid or healthcare – to name a few. In building automation for example, networks of hundreds or thousands of nodes are envisioned [23] to help saving energy and enhance user comfort through the coordination of power sockets, light bulbs, and electrical appliances. To scale up and lower the costs, most scenarios consider battery-operated deployments, where nodes communicate wirelessly over multiple hops. In order to make heterogeneous devices interoperate in this context, the IETF Roll working group standardized RPL [39] in March 2012, a routing protocol that targets low-power IPv6 networks.

Communication in such general-purpose low-power networks is challenging. First, it must satisfy the *energy* requirements of the devices, to help operating on battery and to keep maintenance costs low. Second, it must support traffic with *any pattern* in time and space, to act as an application-agnostic routing infrastructure. Third, it must be *reliable and reactive*, enabling interactive applications. The above goals are hard to conciliate: saving energy requires radio duty cycling [8, 17], which increases latency, and makes it harder to support non-scheduled transmissions.

Recent work has shown that opportunistic routing is an efficient way to achieve low-latency yet energy-efficient data collection in WSN [24, 26, 38]. For instance, ORW [26], uses anycast targeting a set of potential forwarders, instead of relying on stable links only and waiting for a specific neighbor to wake up. By exploiting all available links, it improves CTP [21], a reference data collection protocol, in both energy and latency [26].

This paper presents ORPL, an extension of RPL that performs opportunistic routing. ORPL advances the state of the art by demonstrating the applicability of opportunistic routing to addressable low-power networks. As illustrated in Figure 1, nodes in ORPL route towards the network root using all available parents as potential next hops. This is performed through low-power anycast transmissions along a gradient. Routing away form the root is also done with anycast, directed by a set of reachable destinations stored locally at every node. This *routing set* is represented either as a bitmap or as a Bloom filter (in respectively static or dynamic scenarios), which both are significantly more compact than a routing table, and enable cheap dissemination through the network. When a Bloom filter is used, false positives may arise while checking a node against the set, occasionally leading to wrong routing decisions. In that case, our false positive recovery mechanism fires, and brings the data to its destination through another path.

Our experimental results in a 135-node testbed [11] show that: (1) By making RPL opportunistic, ORPL conciliates low latency and low energy while increasing robustness. In our many-to-one experiments, it attains a packet delivery ratio of 99.5% (against 97.4% for RPL), a latency of 0.47 s (against 1.14 s for RPL), and a duty cycle of 0.48% (against 0.99% for RPL). In any-to-any communication, the latency reduction is even more substantial, up to a factor of 14 on individual pairs of nodes. ORPL keeps its reliability above 98% during a network outage that causes RPL to drop 40–60% of the application data. (2) By basing its routing decision on a fixed-size routing set, ORPL scales to large networks while maintaining good performance. In our one-to-many experiments, ORPL addresses all nodes in the testbed using a 64-byte Bloom filter, with a packet delivery ratio of 98.8%. RPL achieves a delivery ratio of 95.4%, and can only address half of the testbed, due to memory restrictions.

The remainder of this paper is organized as follows: §2 provides required background and gives the intuition behind ORPL. §3 details the design of ORPL. §4 presents our experimental results. We discuss specific aspects of our approach in §5, review related work in §6 and conclude in §7.

2. OVERVIEW

In this section, we first give necessary background on opportunistic routing, any-to-any routing and RPL. Then, we provide a high-level description of our protocol ORPL.

2.1 Opportunistic Routing in Sensornets

Traditionally, routing in multi-hop networks is done through a series of unicast transmissions along a path. Instead, opportunistic routing uses multiple potential forwarders through anycast transmissions. This exploits spatial diversity and embraces the bursty nature of wireless links [36]. In ExOR [4], for example, packets are anycasted to a set of potential next hops. The forwarding decision is made on the receivers' side. All potential forwarders have their own transmission time slot sorted by routing progress, and forward only if they have not overheard transmissions from others. This process selects the best available next hop in a distributed way and avoids duplicate forwarding.

ORW [26] showed that opportunistic routing is also beneficial in duty-cycled data collection networks. Instead of waiting for a specific neighbor to wake up, nodes anycast the packet until any valid forwarder receives and acknowledges it. ORW exploits *any* potentially useful link, rather than using stable links only (as traditional routing protocols do). This increases robustness and shortens the wakeup phase of low-power listening. ORW routes over multiple hops along a gradient, towards a sink, similarly to CTP [21]. It conciliates robustness, low latency and energy efficiency in periodic data collection.

ORPL enables opportunistic routing in a more general scenario, where nodes are addressable and perform on-demand any-to-any routing.

2.2 Any-to-Any Routing and RPL

Applications are now foreseen where nodes are not only data sources, but also addressable end points, with actuation capabilities, and possibly exposed as a resource on the Internet. Building automation and smart cities are examples of such. From a networking point of view, this implies that routing must not only be supported towards a single destination, but from any node to another.

We focus on RPL [39], a routing protocol that provides any-to-any routing in low-power IPv6 networks, standardized by the IETF in March 2012. Its design is largely based on CTP [21], the reference data collection protocol for sensor networks. The RPL topology is a DODAG (Destination-Oriented Directed Acyclic Graph) built in direction of the root, typically an access point to the Internet. Any-to-any traffic is routed first upwards, *i.e.*, towards the root, until a common ancestor of destination and source is found, and then downwards, following the nodes' routing table[1].

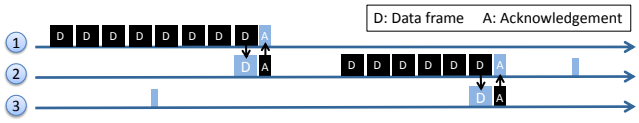
Note that RPL uses a simple rooted topology instead of a full mesh. Doing so, the entire effort – both in memory and control traffic – is devoted to the maintenance of reliable paths to a single destination. The purpose of this strategy is to scale to large networks while containing the routing overhead, at the price of increased hop count (routing via a common ancestor). ORPL adopts the same rooted topology, but alleviates latency problems through multipath routing and exploits shortcuts between branches of the topology.

2.3 ORPL in a Nutshell

ORPL brings opportunistic routing to RPL, aiming for low-latency, reliable communication in duty-cycled networks. ORPL focuses on providing a low-power mesh that supports *any-to-any* traffic with *arbitrary* patterns.

Routing upwards in ORPL is done similarly to ORW, opportunistically and along a gradient. At the MAC layer, ORPL uses anycast over a low-power-listening MAC, as illustrated in Figure 2. The selection of the next hop is done *during* the transmission: The first neighbor that, (1) wakes up, (2) successfully receives the packet and (3) is closer to the root, acknowledges the packet and forwards it.

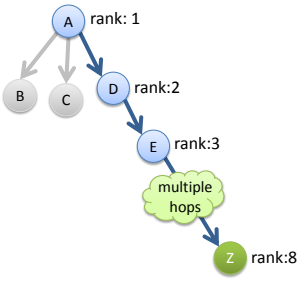
[1]RPL also has a so-called non-storing mode that performs source routing. We focus on the storing mode, for its higher efficiency in any-to-any routing (in source routing, the only usable common ancestor is the root).



(a) Low-Power Listening Unicast

(b) Low-Power Listening Anycast

Figure 2: Traditionally, routing uses unicast over stable links, aiming for stability. Low-power listening introduces a significant delay at every hop. ORPL uses anycast and transmits to the first awoken forwarder that receives the packet, regardless of link quality estimates.



(a) Example Topology

Node A	B → B
	C → C
	D → D
	E → D
	Z → D
Node D	E → E
	Z → E

(b) RPL Routing Tables

Node A	B, C, D, E, Z
Node D	E, Z

(c) ORPL Routing Sets

Figure 3: To reach Z, A has to route downwards with D as a next hop. In RPL, A uses a routing table and unicasts to D. In ORPL, D elects itself as next hop upon receiving A's anycast, based on the inclusion of Z in its own routing set and on the increasing rank.

Routing downwards, however, cannot rely on the gradient only, as merely routing away from the root is not enough to ensure that the destination will be reached. Since we use anycast, nodes do not need to choose a next hop (the next hop elects itself while receiving), and therefore do not need a traditional routing table. We introduce the notion of *routing set*, the set of nodes that are below in the DODAG (from a graph point of view, the sub-DODAG rooted at the node). This information tells whether a node is on a path to the destination or not, as illustrated by Figure 3. A neighbor forwards a packet downwards if it (1) wakes up, (2) successfully receives the packet, (3) is farther from the root and (4) has the destination in its routing set.

ORPL supports any-to-any traffic by first routing upwards to *any* common ancestor, and then downwards to the destination. The unique combination of a rooted topology with opportunistic routing is what enables basing the forwarding decision on a routing set – a key to ORPL's scalability.

ORPL has two variants. (1) The simpler assumes a network with a fixed set of elements known at deployment time. It uses a bitmap to represent the routing set. (2) The second variant allows new nodes to join the network at runtime. It uses Bloom filters [5] to represent the routing set in a compact way. This enables efficient and scalable propagation of routing information. The routing set typically fits a single link-layer frame and is piggybacked by ORPL's periodic control messages. When Bloom filters are used, ORPL is augmented with a false positive recovery mechanism that, despite increased latency, keeps the reliability high.

3. DESIGN OF ORPL

This section presents the core mechanisms of ORPL. It encompasses topology construction and inconsistency detection, routing decisions, and routing set representation.

3.1 Topology and Routing Metric

ORPL uses a DODAG as topology, based on the EDC (Expected Duty Cycles) metric [26]. EDC represents the number of MAC wakeup periods required to reach the root. It is the multipath equivalent of ETX [10] (Expected Transmission Count, the most commonly used metric in RPL and CTP). EDC is calculated as the sum of two components: first, the expected time to deliver a packet to any forwarder, and second, the expected remaining time to reach the root. The latter is calculated as a weighted average of the neighbor's EDC, with link estimates as weight. The cost of forwarding is accounted for through a parameter w, typically between 0 and 1, which sets the balance between opportunism and number of hops. For more details on EDC calculation, forwarder set selection, convergence and effects of node density, we refer the reader to the ORW paper [26].

ORPL uses periodic broadcasts with a Trickle Timer [27] to propagate routing information, like RPL or CTP. Note that in the RPL terminology, the distance from a node to the root according to the routing metric is called *rank*.

3.2 Routing Decision

In ORPL, nodes anycast packets instead of choosing a next hop and unicasting to it. Nodes receiving a packet decide whether to forward it or not, and send a link-layer acknowledgment only if they choose to act as next hop.

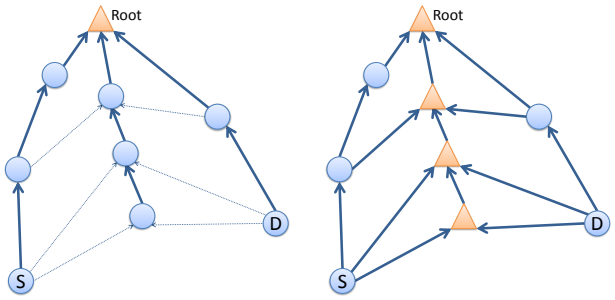
Any-to-any routing is performed by sending the packet upwards until it reaches a common ancestor, then downwards to the destination. Note that multipath routing exploits more nodes as common ancestors, as illustrated in Figure 4.

The routing decision at node N for a packet anycasted by A with destination D is as follows:

Upwards N forwards the packet upwards *iff* it is marked by A as going upwards (this information is included in RPL's IPv6 Hop-by-Hop extension header) and $Rank_N + w < Rank_S$. This guarantees strictly decreasing rank, ensuring the upwards direction. The parameter w avoids forwarding in case of insufficient routing progress (without compromising connectivity [26]).

Downwards N forwards the packet downwards *iff* $Rank_N > Rank_A + w$ and $D \in RoutingSet_N$. Assuming a converged topology, this guarantees that the destination will eventually be reached. This is explained as follows: because $D \in RoutingSet_N$, N is in a possible path from A to D. This implies $Rank_D > Rank_N$, and as $Rank_N > Rank_A + w$, we know that N is closer to D than A is.

Note that the decision to route downwards is intentionally independent from whether the packet is currently marked as going up or down. This enables direct transmission between distinct branches of the topology in the case of any-to-any



(a) RPL (b) ORPL

Figure 4: RPL nodes use only one current parent. When routing from S to D, the only usable common ancestor is the root. In contrast, ORPL uses every link in the DODAG. Many paths are available from S to D, through 4 different common ancestors (marked as triangles).

routing (avoids routing up to a common ancestor and down again). This feature comes inherently with the opportunistic nature of ORPL: ancestors of the destination from other branches simply decide to forward the packet if they overhear it. The same would be difficult to put in practice with RPL, as it requires to share routing tables among siblings.

The duty-cycled anycast used by ORPL sometimes results in multiple nodes forwarding the same packet, and generating duplicates. We filter out duplicates at the routing layer to reduce unnecessary forwarding. The MAC wakeup interval and parameter w have a sensible impact on duplicate traffic, measured and discussed in §4.2.

3.3 Construction of the Routing Set

Instead of using routing tables, ORPL nodes use a routing set storing the nodes that are below them in the DODAG. This routing set only contains destinations, as opposed to a routing table made of ⟨next hop, destination⟩ pairs. It can be represented in a more compact way than a traditional routing table. In scenarios where the whole network is known at deployment, the routing set is represented as a bitmap (*i.e.*, one bit per node in the network). In scenarios where new nodes may join the network after deployment, we use a Bloom filter instead.

In RPL, nodes propagate their routing entries through unicast (so-called DAO messages) to their parents. In ORPL, the routing set is broadcasted instead (appended to the Trickle beacons), and the decision to consider a node as a child is made by the parent based on the ranks. This way, routing set propagation is also opportunistic and maximizes the number of links available for downwards routing. Nodes trigger a transmission of their routing set every time they update it. We avoid broadcast storms through simple delayed transmissions with duplicate suppression.

Upon receiving a routing set from a neighbor C, a node N checks whether (1) the neighbor is a child ($Rank_N > Rank_C + w$) and (2) the link quality from N to C is greater than a threshold. If so, N inserts C in its set, and merges C's set with its own. Sets are simply merged through a bitwise OR, in both cases of a bitmap or a Bloom filter. We use a link-quality threshold of 50% reception rate, simply filtering out bad links.

3.3.1 Link Quality Estimation

Nodes are inserted in routing sets based on link quality estimates between the parent and the child. This ensures that children are reachable from their parents. ORPL uses beacon counting to estimate links. This way, link estimates are maintained even in the absence of application traffic.

In our current design, we use acknowledged broadcasts for beacon counting. With low-power-listening, broadcasts are sent repeatedly for exactly one wakeup period. Nodes receiving a broadcast send a link-layer acknowledgment extended with the receiver's address. Link estimates are based on the number of acknowledgments received by the neighbors. We add a small jitter in the wakeup schedule to avoid repeated collisions due to synchronization.

3.3.2 Aging

In low-power networks, links are dynamic, forcing the routing topology to adjust. Because the routing set is built by aggregation, it would be erroneous to remove an element when a single children has lost its link to it. For instance, assume node N has D in its routing set, because D is reachable from either child B or C. When B loses its link to D, N should not remove D from its set, because a route to D still exists through C.

To tackle this problem, every node maintains exactly two routing sets, one *active* and one *warmup*. Routing decisions are made on the active set only, whereas insert and merge operations are applied to both sets. Periodically (following the Trickle timer), the two sets are permuted, and the warmup one is emptied. As this technique does not require element removal, it is suitable not only for bitmaps but also for Bloom filters (this is a common way to implement aging Bloom filters [6]).

3.3.3 Inclusion of the 1-hop Neighborhood

As an optimization, and to increase the number of available paths, we extend the routing set with the 1-hop neighborhood of the nodes in the sub-DODAG. To do so, nodes include (but do not merge the set of) all neighbors with a good link in their routing set, including those that are not children. The routing decision remains unchanged: going down ensures the packet will eventually reach either the destination or a node having it in radio range.

3.4 Bloom Filters

Using bitmaps for storing the routing set requires complete knowledge of all nodes in the network. Letting new nodes join the topology would require to reset the network and propagate new bitmaps. To support node insertion without the need to reset the network, ORPL can also represent routing sets as Bloom filters.

3.4.1 Bloom Filter Basics

A Bloom filter is a space-efficient, probabilistic data structure for set insertion, membership query, and merging [5]. A filter is an array of m bits. Inserting an element is done by setting k bits of the filter. The position of the k bits is obtained deterministically by applying k hash functions on the input. Checking the membership of an element is done by hashing the element again and checking if all k bits are already set. If not, one can conclude the element is not in the set. If yes, then the element may be in the set, but there is a possibility of false positives.

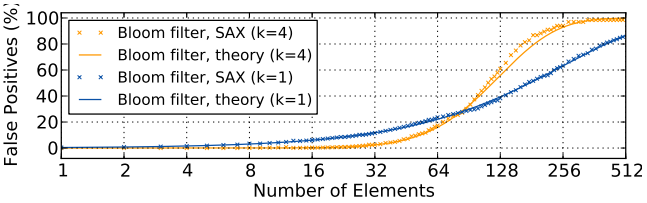


Figure 5: The false positive rate in a Bloom filter that uses SAX hash is close to the theoretical bound. The number of hashes k tunes the system towards optimality at either more full or more empty filters.

As an example setup for ORPL, assume an expected number of elements (addressable nodes in the network) $n = 60$, a and Bloom filter size $m = 256$ *bits*. The optimal number of hashes is $k = \frac{m}{n} \times \ln(2) \approx 2.96$. Using $k = 3$, the probability of false positive p is [5]:

$$p = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \approx 0.12928$$

which is a false positive rate of about 13%, for a space efficiency of $\frac{m}{n} = 4.26$ bit per inserted element.

3.4.2 Choice of the Hash Function(s)

A Bloom filter needs k hash functions that produce hashes of size $log_2(m)$ bits. A common way to implement this is to use a single hash function producing $k \times log_2(m)$ bits, and to split its output in k parts. With ORPL, nodes use their Bloom filter during packet reception to decide whether to acknowledge or not. Therefore, we need a hash function that executes fast enough (less than the radio time of a frame). We also need the hash function to be of reasonable memory complexity, so it fits the target devices.

We choose a SAX (Shift-and-Xor) hash, which was identified by Ramakrishna *et al.* [33] as one of the simplest hash functions that conciliates the properties of: uniformity, universality, applicability, and efficiency. It is computed as h_n, where $\forall i \in [1; n], c_i$ are the input bytes, and where h_i is:

$$h_i = \begin{cases} h_{i-1} \oplus (h_{i-1}\texttt{<<}5 + h_{i-1}\texttt{>>}2 + c_i) & \text{if } i > 1; \\ 0 & \text{if } i = 1. \end{cases}$$

The quality of a simple hash function like SAX heavily depends on the input it works with. To assess SAX in the context of ORPL, and to calibrate our system, we run SAX offline on an input set similar to what the routing set will contain: IPv6 addresses that share the same prefix.

Figure 5 shows, for $m = 256$, $k \in \{1, 4\}$, and $n \in [1; 512]$, the false positive rate obtained by SAX against the theoretical bound. With this setup, SAX remains close to the theoretical bound, under-performing it by a maximum of 10 percentage points. Note that the cases $k = 1$ and $k = 4$ intersect at $n = 83$. For ORPL, this means that choosing $k = 1$ optimizes for nodes having the most full routing sets, whereas $k = 4$ optimizes for more lightly loaded sets. We evaluate the practical effects of m and k while running ORPL in §4.5.

3.4.3 False Positive Recovery

Bloom filters imply false positives (FP), which, in the context of routing sets, lead to wrong routing decisions (a node forwards a packet downwards although the destination is not in its sub-DODAG). As routing sets are propagated upwards in the routing topology, false positives also propagate. When a FP occurs, the packet will be forwarded downwards until hitting the source of the FP, a node having no potential next hop for the current destination. After a fixed number of failed transmissions, ORPL triggers a false positive recovery mechanism that tries to reach the destination via other routes. The recovery mechanism follows a two-step process:

Blacklisting and Returning When a node detects a FP, it inserts the packet sequence number in a (short-lived) blacklist, and then unicasts the packet back to the previous hop. The packet is marked as being recovered.

Exploring Upon receiving a packet that is in recovery, the parent makes another attempt to route downwards. If it fails, the process is repeated recursively, *i.e.*, the packet is blacklisted and sent upwards. In the worst case, this results in a full in-depth exploration of the subset of the DODAG having the destination in its routing set. In practice, with reasonably low false positive rates, only a small portion of this subset is explored before finding a correct route.

Note that ORPL cannot distinguish an actual false positive from and a loss due to bad link conditions or topology changes. Node disconnections have a similar impact as false positives, leading to routing dead ends. To ensure the packet will be dropped in reasonable time in bad link conditions (*e.g.*, external interference), we limit the number of transmission attempts during recovery[2].

3.5 Dealing with Network Dynamics

Wireless links in low-power networks are often bursty [36, 37], resulting in instability of the logical topology. To maintain a consistent topology, ORPL extends RPL's datapath validation and rank hysteresis mechanisms, adapting them to the requirements of multipath routing and routing sets.

Datapath Validation Datapath validation consists in using data traffic to detect inconsistencies and heal the topology. To detect inconsistencies in the routing set, ORPL performs a lookup in the local routing set for every packet that is received or overheard. If the packet source is a child but is not present in the set, it is added. If the packet source is not a child, but is present in the routing set, ORPL triggers a self-healing process: the node switches its routing set (aging), and resets its Trickle timer.

Rank Hysteresis RPL's rank hysteresis mechanism prevents nodes from switching parent for too little rank improvements. ORPL extends it by applying it to all rank updates, not only parent switches. Each node maintains both a *real* and *advertised* rank. It does not advertise a new rank until the change (positive or negative) reaches the hysteresis threshold. Doing so, we make the topology more stable, and make routing set inconsistencies less frequent.

3.6 Implementation Aspects

We implement ORPL[3] in the Contiki OS [14], based on RPL. We review a number of relevant features that are specific to this implementation.

ContikiMAC At the MAC layer, we use ContikiMAC [13], a low-power-listening MAC, that we extend to support anycast. ContikiMAC is similar to

[2]Our implementation triggers recovery after 5 attempted anycasts. Subsequent transmissions use only 2 attempts.

[3]ORPL is available at `https://github.com/simonduq/orpl`

Network Settings	Experiment	Protocol	PDR (%)	Latency (s)	Duty Cycle (%)
Tx power: 0 dBm Diameter: 5.7 hops Density (min): 3 nodes (avg): 17.1 nodes (max): 33 nodes	Many-to-one	ORPL (+ ContikiMAC)	99.50	0.47 (max: 1.38)	0.48 (max: 2.58)
		RPL (+ ContikiMAC)	97.39	1.14 (max: 4.56)	0.99 (max: 12.92)
		ORW (+ BoX-MAC)	98.08	0.82 (max: 2.31)	1.08 (max: 2.96)
		CTP (+ BoX-MAC)	98.46	1.98 (max: 12.29)	2.20 (max: 4.99)
		LWB	99.88	2.82 (max: 3.58)	1.24 (max: 2.09)
		LWB (small payload)	99.93	1.69 (max: 2.08)	0.61 (max: 1.25)
	One-to-many	ORPL	98.96	0.96 (max: 1.90)	0.57 (max: 2.45)
		RPL	91.92	2.09 (max: 6.72)	1.74 (max: 21.63)
	Any-to-any	ORPL	98.48	1.08 (max: 2.35)	0.47 (max: 1.94)
		RPL	71.29	3.75 (max: 12.75)	2.16 (max: 12.99)
Tx power: -10 dBm Diameter: 9.8 hops Density (min): 1 node (avg): 9.3 nodes (max): 24 nodes	Many-to-one	ORPL	98.85	1.22 (max: 3.66)	0.83 (max: 5.45)
		RPL	96.19	2.17 (max: 6.63)	1.35 (max: 13.32)
	One-to-many	ORPL	98.10	1.60 (max: 3.60)	0.69 (max: 2.92)
		RPL	90.76	2.82 (max: 6.95)	1.82 (max: 16.06)
	Any-to-any	ORPL	95.45	1.97 (max: 3.45)	0.41 (max: 2.94)
		RPL	74.44	4.27 (max: 26.17)	1.16 (max: 12.66)

Table 1: Experiments Summary. ORPL improves latency and energy consumption over other solutions, for all traffic patterns as well as with reduced network density. LWB is the most robust protocol, but has a substantially higher latency and consumes more energy than ORPL, even when used with reduced payload.

BoX-MAC-1 [30], in that it uses the data packet itself as preamble. It has two major differences with BoX-MAC-1. (1) Its wakeup consists of two clear channel assessments that last $192\mu s$ (similarly to BoX-MAC-2), instead of radio listening for a few milliseconds. (2) ContikiMAC has a phase-lock mechanism, where senders record their neighbor's wake-up phase, and use it to make the next transmissions cheaper. Note that phase lock is used for unicast only, not broadcast nor ORPL anycast.

We extend the ContikiMAC default phase lock guard time from 15 to 63 ms, which we found necessary to avoid losing synchronization when long wakeup intervals are used. We use Contiki's default number of transmission attempts, 5, with an exponential backoff, which we found to achieve a good balance between reliability and latency.

Software Acknowledgments We implement anycast for the TelosB platform and its cc2420 radio chip. We disable hardware acknowledgments and configure the chip to trigger an interrupt after receiving the 802.15.4 frame and IPv6 headers. From the interrupt, we check the destination IPv6 against the routing set, compare ranks, make the forwarding decision, and send an ACK if we decide to forward the packet.

Control Traffic On the data plane, ORPL uses the standard RPL Hop-by-Hop extension header, that contains, among others fields, the current routing direction (up or down) and the rank of the sender. On the control plane, ORPL extends the periodic DIO beacons with the current routing set of the sender. In our implementation, the routing set is sent as a separate frame, as the default DIO almost fills out a complete 802.15.4 frame.

Memory We review the amount of volatile memory used by RPL and ORPL, in two parts: (1) Neighbor tables, which scale linearly with network density. This includes link quality estimates, MAC addresses and other small bits of information. In the current implementation, each neighbor consumes 18 bytes, for both RPL and ORPL. (2) Routing tables, which scale linearly with network size. In RPL, each routing entry consumes 48 bytes (global and local IPv6 addresses, lifetime and other data). In ORPL, two routing sets (one active, one warmup) are enough to cover the whole network. In our evaluation, we use bitmaps of 17 bytes, or Bloom filters between 8 and 80 bytes, to address 135 nodes (from 0.47 to 4.74 bits per entry and per filter).

In our experiments, with a maximum of 33 neighbors and 135 nodes, RPL consumes $33 \times 18 + 135 \times 48 = 7074$ bytes, whereas ORPL uses $33 \times 18 + 2 \times 80 = 754$ bytes of RAM. Note that the routing tables of RPL could be made substantially smaller through address compression, but reaching the compactness of ORPL's routing sets seems difficult (routing tables contain strictly more information).

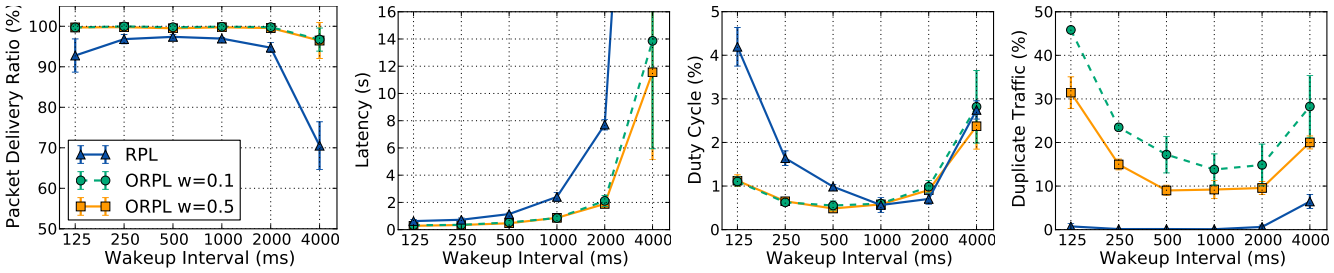
4. EVALUATION

In this section, we evaluate ORPL experimentally through our implementation. We first show, in a data collection scenario, that ORPL increases RPL's reliability, latency and energy-efficiency. ORPL also outperforms other state-of-the-art solutions such as the Collection Tree Protocol (CTP) [21], Opportunistic Routing for WSN (ORW) [26] or the Low-Power Wireless Bus (LWB) [18]. Second, we focus on our routing-set-based forwarding mechanism, and demonstrate that opportunistic routing also improves downwards routing. Then, we focus on the effects of Bloom filters and the false positive recovery mechanism. We then demonstrate the benefits of ORPL in robustness, through experiments that emulate a network outage in the testbed. Finally, we evaluate ORPL in a scenario where individual nodes request and respond to one-another. We find that ORPL is even more beneficial to any-to-any routing than it is to upwards or downwards routing alone, because of its ability to shortcut different branches of the topology. A summary of our experiments is presented in Table 1.

4.1 Methodology

We evaluate ORPL in the Indriya testbed [11] which has 135 TelosB nodes spanning three floors of an office building[4]. We use node 20 as the network root, at the third floor,

[4]Indriya has a total of 139 nodes, of which 135 were available at the time we ran our experiments.



(a) Reliability (b) Latency (c) Energy (d) Transmissions

Figure 6: Upwards Routing. At any wakeup interval, ORPL outperforms RPL in reliability and latency, despite its significantly higher duplicate rate. In its most energy-efficient setting (interval of 1000 ms), RPL has a duty cycle of 0.57% and a latency of 2.51 s. ORPL, at an interval of 500 ms, conciliates a duty cycle of 0.48% and a latency of 0.47 s.

maximizing the network diameter. In most experiments, we use the maximum transmission power of the cc2420 radio chip, *i.e.*, 0 dBm. This leads to a diameter of 5.7 hops (averaged over our RPL runs) and a density ranging from 3 to 33 neighbors, 17.1 on average. To assess ORPL in a less dense environment, we also run experiments with a transmission power of -10 dBm, the minimum power that led to a fully-connected network. With -10 dBm, the diameter is 9.8 hops and the density between 1 and 24 neighbors, 9.3 on average. In all experiments, the application payload is 64 bytes. In the RPL and ORPL cases, all traffic is carried in UDP datagrams over 6LoWPAN.

4.1.1 Protocols

Throughout our evaluation, we compare ORPL against RPL. In the case of periodic data collection, we also include CTP [21], ORW [26] and LWB [18] as reference points.

RPL We used RPL as the basis of ORPL's design and implementation. Comparing against it allows to isolate the effects of opportunistic routing from other factors such as MAC layer or OS. We run RPL over Contiki with ETX [10] as a routing metric.

CTP CTP is the default collection protocol in TinyOS, often used as a benchmark. CTP informed the design of RPL, and, in many-to-one scenarios, the two protocols resemble each other. We run CTP over BoX-MAX, the default low-power MAC in TinyOS.

ORW ORW extends CTP with opportunistic routing over low-power anycast. To the best of our knowledge, it is the only low-power opportunistic routing protocol implemented for sensors and tested experimentally (ORW runs over TinyOS and BoX-MAC). ORPL, when routing upwards, operates similarly to ORW.

LWB The Low-Power Wireless Bus (LWB) is a protocol that follows a radically different approach. It bases every communication on Glossy [19], a network flooding mechanism that exploits constructive interference of synchronous transmissions. Targeting periodic traffic applications, LWB schedules Glossy rounds to satisfy the load advertised by the nodes in the network. It was shown to outperform state-of-the-art collection and many-to-many protocols under various network conditions [18].

We use a Contiki-based implementation of LWB that was provided by its authors. We enable the low-latency mode, in which the network coordinator schedules new rounds as aggressively as possible, trading energy for latency [18]. The minimum period is set to 1 second, the maximum to 30 seconds. Glossy rounds are sensitive to the network diameter and frame size. We run LWB in two different settings: (1) supporting full 64 byte payload (round time of 20 ms), (2) supporting smaller payload (15 byte, with a round time of 10 ms [18]). Because LWB is designed for periodic traffic, we use it with constant inter packet interval, instead of randomized transmissions within an interval.

4.1.2 Metrics

We focus on three key metrics. (1) The Packet Delivery Ratio (PDR) is the fraction of packets received over those sent, in end-to-end communication. It tells how reliable the protocol is. (2) The packet latency is also measured end-to-end, from the application requesting transmission at the source node, to the application receiving the data at the target node. To minimize latency in random access networks is one of the main goals of ORPL. (3) We use duty cycle, the portion of radio on-time, as a metric for energy efficiency. This metric is a good proxy for power, because typical sensor platforms have their power profile dominated by the radio chip, and because transmit and listen operations commonly have a similar current draw. Furthermore, duty cycle, unlike the actual power consumption in Watts, is a platform-independent metric. We measure the nodes' duty cycle in software using Contiki's energy profiler [15].

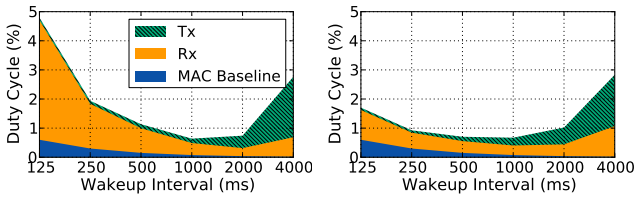
Unless otherwise mentioned, all our experiments run for a duration of one hour. The results are averaged over at least three runs, and error bars show the standard deviation.

4.2 Upwards Routing

Our first series of experiments focuses on the most traditional traffic pattern in sensor networks: periodic data collection. Nodes send packets randomly with an average interval of 4 min. The resulting network load is 0.56 packet/second, which, in most settings (exception of RPL with a wakeup interval of 4000 ms), is well below network saturation.

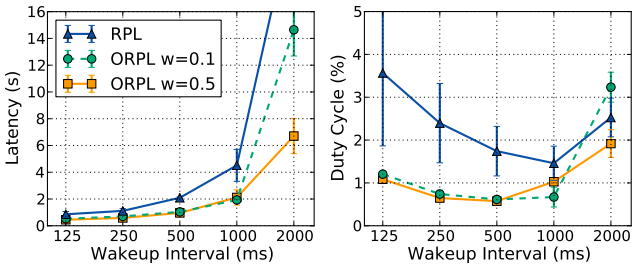
4.2.1 Effect of Wake-up Interval

Figure 6 summarizes our results with RPL and ORPL for various ContikiMAC wake-up intervals and parameter w. With all settings, ORPL is more reliable than RPL, and maintains a reliability above 99.5% at any wakeup interval



(a) RPL (b) ORPL

Figure 7: Energy Profiles (Upwards Routing). RPL and ORPL have different optima, because of a different balance between reception and transmission.



(a) Latency (b) Energy

Figure 8: Downwards Routing. ORPL, through its routing-set-based forwarding, enables opportunistic downwards routing, saving energy and latency.

below 4000 ms (Figure 6a). RPL reaches a maximum packet delivery ratio of 97.39%. As the wakeup interval increases, the performance of RPL drops due to network congestion.

Figure 6b shows that ORPL also reduces latency. In most cases, its end-to-end latency is below the ContikiMAC wakeup interval, *i.e.*, packets are forwarded up to the root in less than a cycle time (*e.g.*, latency of 0.47 s for a wakeup interval of 500 ms). This is a result of low-power anycast exploiting the first awoken node that provides routing progress. Figure 6c shows that RPL and ORPL have different energy tradeoffs. ORPL is most energy efficient at a wakeup interval of 500 ms (duty cycle of 0.48%) and RPL at an interval of 1000 ms (duty cycle of 0.57%). We find $w = 0.5$ to produce the best results in this data collection experiment.

Figure 6d shows that ORPL produces an significant number of duplicates, accounting for between 9% to nearly 50% of the traffic in our runs. This is an inherent drawback of ORPL, which becomes more pronounced as the wakeup interval decreases (or as the density increases). At an interval of 4000 ms, the duplicate rate is high due to network congestion that triggers more collisions and retransmissions.

Overall, we find a 500 ms wakeup interval to offer a good compromise, with both RPL and ORPL achieving a latency below 2 seconds and a duty cycle below 1%. We use a default period of 500 ms and $w = 0.5$ for next experiments.

4.2.2 Limits of ORPL Against RPL

In our experiments, ORPL nearly constantly outperforms RPL in all metrics. However, we believe the phase-lock mechanism of ContikiMAC can be improved with more accurate synchronization and tighter timings.

Figure 7 details the energy profile of RPL and ORPL. At a period of 1000 ms or 2000 ms, RPL achieves a duty cycle lower than ORPL, thanks to phase lock. However, increasing the period further causes nodes to lose synchronization and waste energy. In other words, the growing green area at the right end of Figure 7a is an artefact of the current phase lock implementation having loose timings. With a better synchronization and shorter guard times, we expect RPL to be the most energy-efficient solution as the period increases (while still having ORPL superior in reliability and latency).

4.2.3 Comparison with CTP and ORW

Table 1 includes a comparison against CTP [21] and ORW [26]. Note that both protocols run atop TinyOS and BoX-MAC, whereas RPL and ORPL use Contiki and ContikiMAC. We use a wake-up period of 2 seconds for BoX-MAC, which leads to the best average duty cycle in our experimental setup.

As expected ORW improves CTP in reliability, latency and energy, in a similar proportion as ORPL improves RPL. We also find that ContikiMAC-based solutions (RPL and ORPL) outperform BoX-MAC-based solutions (CTP and ORW). This is explained by ContikiMAC's cheaper wakeup and phase-lock mechanism, which lead to a substancially lower baseline than BoX-MAC. At the network layer, in the particular case of data collection, RPL and ORPL operate similarly to respectively CTP and ORW.

4.2.4 Comparison with LWB

Table 1 includes a comparison against LWB [18]. In our experiments, LWB is more reliable than ORPL, with a delivery ratio of 99.93% against 99.76%. This is a result of the Glossy flooding that exploits constructive interference. ORPL achieves significantly lower latency than LWB, with an average of 0.47 s against 1.69 s to 2.82 s. This is explained by the fact that LWB which targets periodic traffic applications requires nodes to wait for the next schedule from the network coordinator before transmitting.

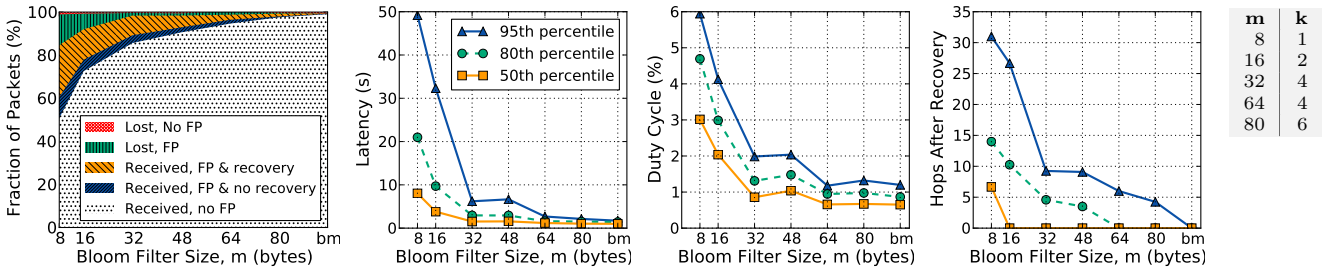
ORPL outperforms LWB in average energy, but note that LWB spreads the load more evenly among nodes. Indeed, the most loaded node with LWB has a duty cycle of 1.25 or 2.09% against 2.58% for ORPL. Load balancing is not the first target of ORPL, but we believe the protocol could be improved in that direction, for example by taking the energy level into account in the rank calculation, affecting the depth in the logical topology.

Finally, a major difference between the two protocols is that while LWB is tailored for periodic traffic, ORPL is a generic routing infrastructure supporting on-demand traffic, which makes it appealing for IP-based networks.

4.3 Downwards Routing

We now investigate the downwards routing mechanism of ORPL, in which nodes perform opportunistic routing directed by routing sets. At this stage, we represent the routing set as a bitmap, to rule out the effect of Bloom filter false positives. The experiment has two steps: (1) network convergence, where the ORPL or RPL topology is built until the root has all destinations in its routing set, and (2) application execution, where the root sends a packet to a randomly chosen node every 4 s.

Due to memory restrictions of the TelosB nodes, RPL was unable to address the whole testbed. We restrict the set of addressable nodes to half of the testbed (nodes with

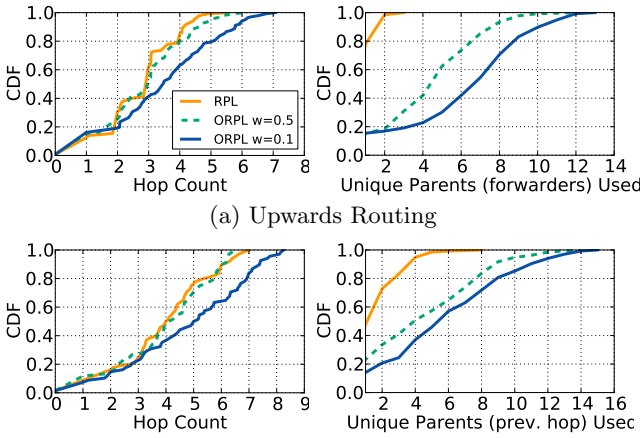


(a) Reliability (b) Latency (c) Energy (d) Hop Count After Recovery

m	k
8	1
16	2
32	4
64	4
80	6

(e) k used

Figure 10: Effect of the Bloom Filter Size m. A Bloom filter of 32 bytes or more achieves a PDR above 98% and performs nearly as well as a bitmap (marked as "bm"). Below that size, the false positive rate increases, more recoveries are triggered, which leads to increased latency and duty cycle.



(a) Upwards Routing

(b) Downwards Routing

Figure 9: Hop Count and Unique Parents (wakeup interval = 500 ms). In the downwards case, ORPL exploits even more routing diversity than in the upwards case, but also has higher hop count.

an even ID), *i.e.*, 67 nodes. The resulting RPL routing table consumes 3.1 kB of RAM. In contrast, ORPL needs only two (one active and one warmup) bitmaps of 9 bytes to address the same subset of the testbed.

Figure 8 shows that our routing-set-based anycast makes ORPL not only superior in data collection, but also in downwards traffic. Both ORPL and RPL perform worse downwards than upwards, because their routing topology is built and optimized in the upwards direction. At a wakeup period of 500 ms, ORPL reduces RPL's latency from 2.09 s to 0.96 s (see Figure 8a), and the duty cycle from 1.74% to 0.57% (see Figure 8b). ORPL is also more reliable, with a PDR of 99.0% against 91.9% for RPL.

Figure 9 takes a closer look at the routing topology, in number of hops and parent diversity, at a wakeup period of 500 ms. The figure shows that ORPL exploits routing diversity, including in the downwards case, with a unique parent count up to 16 on individual nodes. This comes at the price of a higher hop count, especially when w is set to a low value. It shows that our routing-set-based forwarding mechanism exploits spatial diversity in downwards routing – a necessary step to make any-to-any routing possible.

4.4 Effect of Network Density

Because ORPL utilizes as many neighbors as possible for forwarding, its performance depends on the network density. Table 1 shows that even with a reduced density (transmission power set to -10 dBm), ORPL improves RPL for all traffic patterns in reliability, latency and energy. The relative benefits of ORPL over RPL are slightly decreased, but remain significant. For instance, in many-to-one traffic, the latency is improved by a factor of 1.8 (this factor was 2.2 at 0 dBm), and the duty cycle by a factor of 2.6 (was 3.1 at 0 dBm). In earlier work, we reported similar ratios when comparing ORW to CTP in data collection with different network densities [26]. With all traffic patterns including any-to-any, ORPL keeps the average latency below 2 s, in a 9.8-hop network and with a wakeup interval of 0.5 s.

4.5 Bloom Filters and Recovery Mechanism

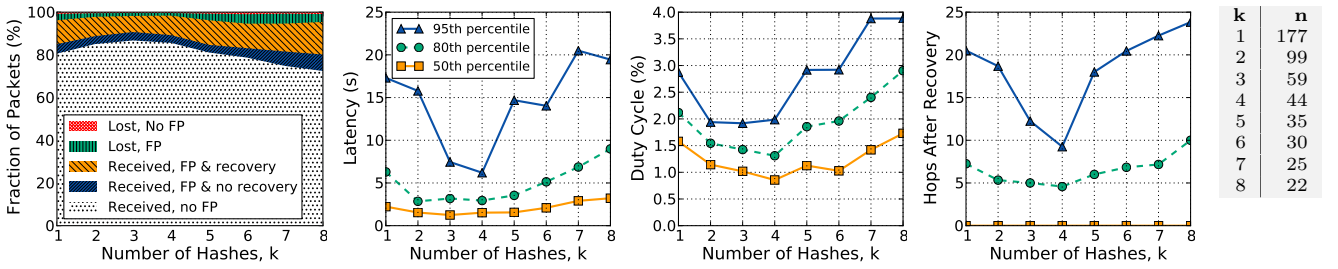
We have assessed ORPL in upwards and downwards routing while representing routing sets as bitmaps. We now use Bloom filters instead of bitmaps which makes ORPL suitable in scenarios where all nodes are not known at deployment time. We perform downwards routing in a scenario similar to the previous subsections, with the difference that we now address all nodes in the testbed.

During our experiments, we log every insert and merge operation on Bloom filters. This way, we compute offline the exact routing set of every node at any point in time. We distinguish true positives from false positives by comparing this set against the routing decision made by the nodes.

4.5.1 *Effect of the Bloom Filter Size m*

We first focus in Figure 10 on the effect of the Bloom filter size m on routing performance. We vary the size between 8 bytes and 80 bytes, and use the corresponding number of hashes k given in Figure 10e. As the false positive rate of a Bloom filter depends on the number of bits per element, not on the size nor number of elements alone, reducing the filter size emulates scenarios with larger scale. With an 8-byte filter, the root works with as little as 0.47 bit per element. We also run experiments with a bitmap (marked as "bm") as a baseline, without false positive recovery.

Figure 10a shows the fraction of packets that either are (1) received without FP on the path, (2) received after a FP that did not require recovery, (3) received after a FP that triggered recovery, (4) lost due to FP or (5) lost with-

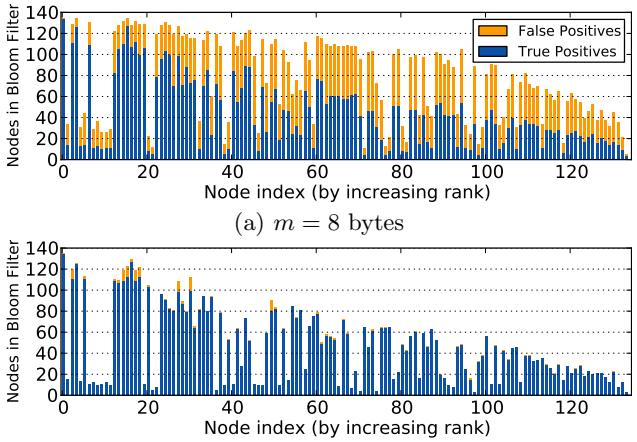


(a) Reliability (b) Latency (c) Energy (d) Hop Count After Recovery

k	n
1	177
2	99
3	59
4	44
5	35
6	30
7	25
8	22

(e) Optimal n

Figure 12: Effect of the Number of Hashes ($m = 32$ bytes). The number of hashes k affects the cost of recovery, and the resulting latency and energy. In this experiment, the optimal k is 3 or 4 (depending on the target metric).



(a) $m = 8$ bytes

(b) $m = 64$ bytes

Figure 11: False Positives per Node. Increasing the filter size from 8 to 64 bytes reduces the rate of false positives from 74.1 to 1.2%.

out FP. False positives that do not require recovery occur when a node does not have the destination in its routing set but finds opportunistically a next hop that can route to it. ORPL is reliable, with a packet delivery ratio above 98% for $m \geq 32$ bytes. The number of false positives increases as m decreases, but, in all cases, a large portion of false positives is successfully recovered. In the case of $m = 32$ bytes for example, of the 15.2% of packets that experience false positives, 3.9 pp (percentage points) were received with no need for recovery, 9.9 pp received thanks to recovery, and 1.4 pp were dropped.

Figure 10b and Figure 10c show the penalty imposed by false positive recovery on duty cycle and latency. At a filter size of 64 bytes and above, latency and duty cycle stabilize. Smaller filter sizes decrease performance, as recoveries become both more frequent and costly (longer exploration). Figure 10d shows the number of hops that are needed to reach the destination after recovery. With $m = 8$ bytes, this number exceeds 30 for the 95^{th} percentile, leading to latencies in the range of a minute.

To get a complete understanding of how false positives are distributed across the nodes, Figure 11 shows, for a filter size of 8 bytes and 64 bytes, the number of elements actually in the routing set (true positives) and the number of testbed nodes that appear to be in the set (false positives). Note that because we test the Bloom filter only against the nodes actually in the network, the sum of true positives and false positive never exceeds 135. With $m = 8$ bytes, the routing sets have on average 74.1% false positives. The resulting end-to-end PDR with this same extreme setting was 85.3%. With $m = 64$ bytes, the false positive rate drops to 1.2%, leading to a PDR of 99.76%.

4.5.2 *Effect of the Number of Hashes k*

When dimensioning Bloom filters for a given ORPL deployment, one has to choose k, the number of hashes (number of bits set at each insertion). There exists an optimal k for a given expected number of elements (see Figure 12e) and bloom filter size, yet choosing k is difficult because nodes have heterogeneous routing sets. We investigate the tradeoffs of k in a given network setup: 135 nodes and a Bloom filter size $m = 32$ bytes.

Figure 12 shows the performance obtained with $k \in [1; 8]$. The metrics that are most impacted by k are the number of hops after recovery (Figure 12d), and consequently the latency (Figure 12b). For instance, at $k = 4$, the 95^{th} percentile of nodes have a latency of 6.20 s against 20.49 s for $k = 7$. The overall duty cycle (Figure 12c) and reliability (Figure 12a) are also affected by k, to a lesser extent.

We find that k should not necessarily be dimensioned for a routing set that contains all nodes in the network. In the above experiment, the best k values are 3 or 4 (depending on the target metric), which optimizes 32-byte filters for respectively 59 and 44 elements – well below 135, the number of addressable nodes. This is explained by two factors: (1) Although the false positive *rate* increases with the number of nodes in the routing set, the *absolute number* of FP does not necessarily. Nodes with intermediately full filters have more nodes that are actually not in their set, leading to more instances of false positives. Optimizing Bloom filters for intermediate nodes alleviates this problem. (2) False positives occurring farthest from the root are the most costly to recover, as they require more hops. Optimizing k for nodes with lightly-loaded Bloom filters eliminates the most costly recoveries.

Figure 13 shows the true positives and false positives obtained with $m = 32$ bytes and $k = 1$ or $k = 8$. These two configurations have false positive rates that are very

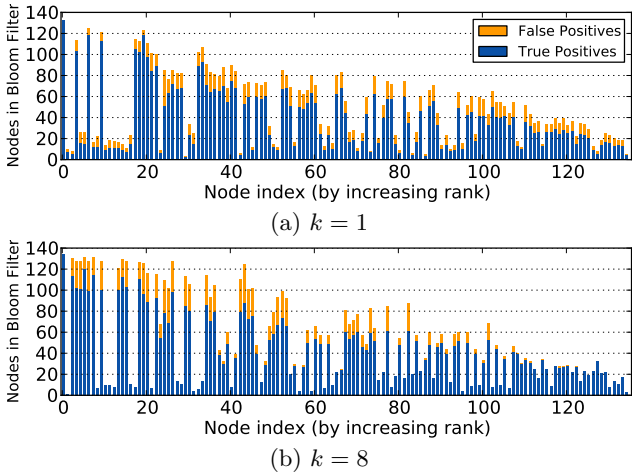


Figure 13: False Positives per Node ($m = 32$ bytes). k affects the distribution of false positives, either mostly close to the root ($k = 8$) or more evenly spread over nodes ($k = 1$). This impacts both the number of false positives and the cost of recovery.

similar (respectively 23.9% and 24.3%), but distributed differently among nodes. With $k = 1$, all nodes have false positives. With $k = 8$, nodes with less than 20 elements have no false positive, but more densely populated nodes experience higher false positive rate.

Finding the best k for a given network is a difficult task, as it depends on the nodes connectivity. In practice, we recommend to proceed to a network calibration, such as presented in this section, before the final deployment.

4.6 Robustness

We assess the robustness of RPL and ORPL when experiencing a network outage, to (1) measure the benefits of ORPL in a challenging environment, and (2) evaluate our recovery mechanism when facing topology changes. We run ORPL under two different settings: either using a 64-byte Bloom filter, or using a bitmap. We run the downwards traffic experiment as in §4.3 where only nodes with an even node ID are addressable. Nodes with an odd node ID participate in routing but are not the target of traffic. Among those, half (randomly chosen set that is kept identical for all runs) experience an outage during which they cannot receive nor send any data. This reflects for example scenarios where battery maintenance requires to disconnect a part of the network, or where external interference (*e.g.*, WiFi or Bluetooth) affects communication. The outage lasts for 10 min, and is repeated every 25 min during a 2-hour experiment.

Figure 14 shows that RPL experiences a sharp drop in reliability during the first outage, consequence of failed MAC transmissions. Nodes react by switching parent, which heals the topology and slowly improves reliability. The next outages result in less churn.

In contrast, ORPL does not need to adjust its topology during network outages. Anycast transmissions use any available forwarder as a next hop, leaving many node failures unnoticed ("MAC Tx Attemps" remains almost constant). When a transmission fails, the recovery mechanism is triggered and keeps reliability high (98.8% on average throughout this experiment). This mechanism is primarily designed

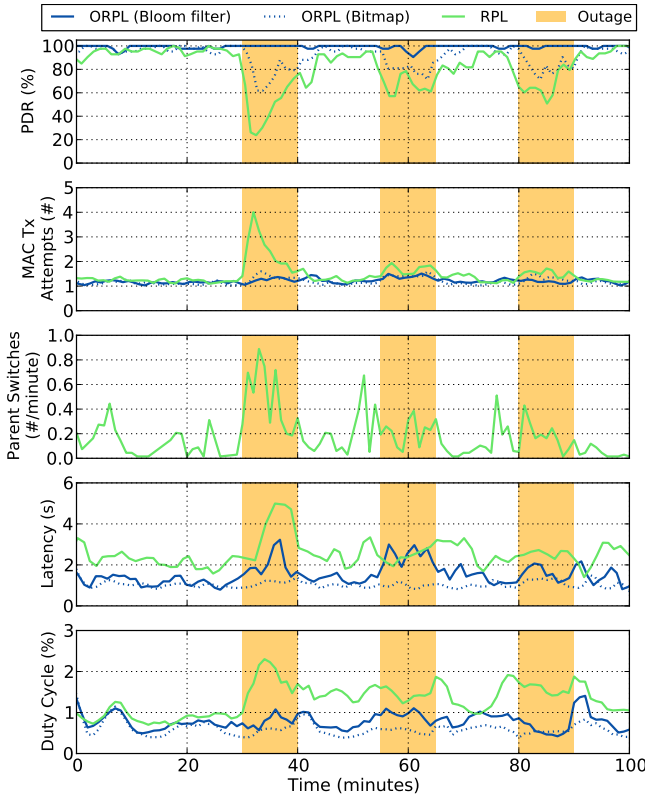


Figure 14: Robustness. ORPL maintains a reliability close to 100% during outages, partially through its recovery mechanism. RPL adjusts to topology changes, but loses data in the process.

for Bloom filter false positives, but handles and recovers from any network inconsistency in the same way. This results in increased latency during outages. The dotted line in Figure 14 shows the results obtained with a bitmap (no recovery mechanism): less reliability but more stable latency.

4.7 Any-to-Any, Bidirectional Traffic

We now evaluate RPL and ORPL in an IP-based request-response scenario, representing applications such as RESTful resource querying with CoAP [25].

We select 9 nodes in the network, at left, center and right positions of each of the three floors of the testbed (as detailed in the upper part of Figure 15). Node 20, at the center of the third floor, is still used as the network root. Each node initiates communication with another randomly chosen node, at random time with an average interval of 2 minutes. Upon receiving a request, the target node replies to the originator. We set up ORPL to use bitmaps (Bloom filters had comparable results, because only 9 addressable nodes led to no false positives), and run the experiments for 2 hours. Note that in any-to-any routing, packets are routed upwards and then downwards to the destination. In this request-response scenario, the process is repeated twice.

Figure 15 shows the hop count and round-trip time between every pair of nodes. These results shown are from a single run, to reflect the heterogeneity of per-node latency obtained with a given DODAG. With RPL, the pair of nodes ⟨left of third floor, center of first floor⟩ was not able to

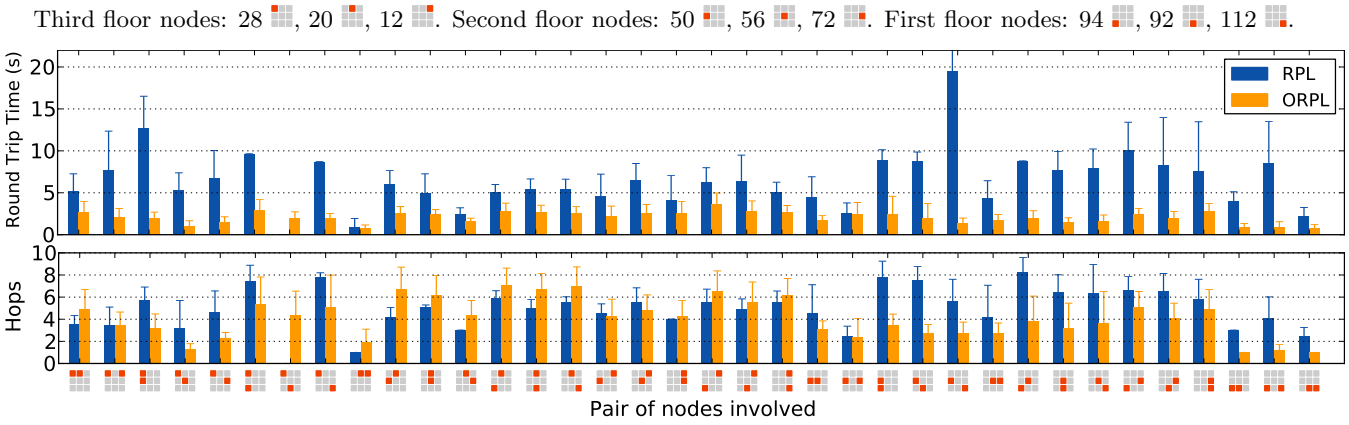


Figure 15: Any-to-any Round Trip. For every single pair of nodes, ORPL achieves substantially shorter and more constant round trip time than RPL. This is due to both the opportunistic nature of anycast, and the shortcuts ORPL takes in the DODAG during any-to-any routing.

communicate, due to topology inconsistencies. ORPL decreases latency substantially, with a median round trip for the median pair of nodes of 2.02 s vs. 6.16 s for RPL.

Looking at round-trip times and hop count as a whole, we observe two distinct cases:

Gains Due to Fast Forwarding In certain cases, ORPL has a higher hop count than RPL, but a lower round-trip time. This is similar to what we have already witnessed in the up- and down-traffic experiments: ORPL saves time at each hop through anycast, even if that implies longer paths.

Gains Due to Shortcuts In other cases, ORPL has a lower hop count than RPL, leading to even higher latency savings. This is explained by the shortcuts ORPL exploits during any-to-any routing (§3.2), and by the optimization that consists in including all direct neighbors the routing set (§3.3). For example, in this experiment, the pair of nodes ⟨left of second floor, right of first floor⟩ has an average latency of 19.52 s with RPL, against 1.34 s with ORPL.

An interesting finding is that in spite of its opportunistic nature, ORPL obtains more stable round-trip times than RPL. The hop count of ORPL has a higher standard deviation than that of RPL, because the routing path varies across transmissions. However, because ORPL selects the first awoken neighbor that offers routing progress as the next hop, the resulting end-to-end latency is more stable than with RPL. In other words, the irregularities of link quality and wakeup schedule are compensated by ORPL through adaptive path selection.

5. DISCUSSION

With ORPL, we have demonstrated the applicability of opportunistic routing to low-power IP networks with arbitrary traffic patterns. We discuss our design and the impact and limitations of our approach.

Impact on Applications RPL targets a variety of applications involving low-power communication, such as building automation, smart cities, or healthcare. These applications typically have well-defined delay requirements. For instance, assume the latency must be in the order of 500 ms. In our data collection scenario (see §4.2), this would be achieved with RPL and a MAC wakeup interval of 125 ms, leading to a network duty cycle of 4.19%. The opportunistic nature of ORPL allows to reach a comparable latency at a MAC wakeup interval of 500 ms, with a network duty cycle of 0.48%. With this example application, ORPL reduces energy consumption by a factor of 8.7 times, thus significantly lowering the battery maintenance cost.

Scalability A main feature of ORPL is its scalability. In our experiments, we addressed 135 nodes with a PDR of 99.8% using a 64-byte Bloom filter. Larger networks, with thousands of nodes, are foreseen *e.g.*, in building automation systems that exploit sub-GHz frequency bands [23]. Such large networks can be covered by ORPL in two ways: (1) Using a Bloom filter that exceeds the size of a single datagram, broadcasted to the neighbors after fragmentation. Fragmented traffic can be supported by low-power-listening MAC with little overhead, by repeatedly sending the fragments as preamble, and having the receivers listen until they get the full datagram [16]. (2) Using a bitmap, and rebooting the network upon addition of new nodes to propagate the new network map. A 64-byte bitmap, which typically fits a single datagram, can accommodate up to 512 nodes. In both cases, ORPL improves RPL's scalability significantly, both in memory and control traffic, because it reduces the routing state to a set of nodes.

Interoperability The main advantage of IP and RPL in sensor networks is interoperability, both above IP (heterogeneous applications and networks) and below IP (heterogeneous link layers). With opportunistic routing in general, link-layer interoperability is an issue, because unlike unicast, anycast transmissions do not target a specific neighbor and its link layer. While RPL can handle heterogeneous link layers almost transparently, ORPL would need to run one instance of the protocol per link layer.

Generality ORPL is designed for low-power-listening 802.15.4 networks carrying IP traffic. Our design, however, is more generic. ORPL could target other physical and link layers as long as they can support any form of anycast. Even non duty-cycled MAC layers can be considered, following a more traditional anycast design, *e.g.*, similar to ExOR [4], which includes a list of potential forwarders in the frame header and uses a duplicate suppression mechanism. The design principles of ORPL are also largely independent of IP, and could be applied with no effort to any network layer.

6. RELATED WORK

The potentials of opportunistic routing in low-power wireless networks were first shown from a theoretical perspective [1, 12, 29, 35]. GeRaF [40] and RAW [31] were the first to propose routing protocols that exploit anycast to reduce latency in duty-cycled networks. A number of variants followed, based on synchronous [32] or asynchronous MAC layers, receiver-initiated [38] or sender-initiated [1, 2, 12, 26]. Pavković *et al.* [32] also extended RPL with multipath routing, but focusing on data collection only, and choosing a next hop opportunistically at the sender rather than leaving the forwarding decision to the receiver. Any-MAC [1] proposes a generic scheme to extend any duty-cycled MAC with anycast capabilities, aiming for opportunistic routing.

Most of the protocols above share the design principle of making the forwarding decision at the receiver, based on whether it is closer to the destination or not. There are, however, a number of different approaches in building a topology that enable routing towards a given destination:

Mesh Networks ExOr [4] is probably the most well-known opportunistic routing protocol. It targets WiFi mesh networks, has a link-state approach, and uses ETX as a metric to calculate distance from any node to another. Periodically, ExOr floods the network with topology information, so that every node has the full knowledge of the topology. Similarly, MORE [9] builds its topology using ETX, but uses network coding to exploit spacial reuse.

Geographic Opportunistic routing was applied early on geographic routing protocols [28, 31, 40]. In that case, the forwarding decision is based on physical node locations. This approach is often used analytically or in simulation, but is sometimes complex to put in practice, because real nodes do not always have location information and because there is no direct mapping between distance and radio connectivity.

Random Walk A more exotic category is random walk, investigated analytically by Basu *et al.* [3]. The authors suggest that random walk has good properties in load balancing and fault tolerance, but would be outperformed in latency by routing protocols that exploit topology information.

Single-Sink The majority of recent work in opportunistic routing for sensor networks focuses on data collection [20, 26, 32, 38]. This approach allows to reason on the pure anycast forwarding mechanism, and typically enables comparisons against CTP [21] as a benchmark. ORW [26] proved the practical feasibility of opportunistic routing in large scale duty-cycled networks. To support multiple destinations, this family of protocols would require to propagate not only distances to the sink, but distances to all nodes in the network; a solution with poor scalability properties.

ORPL addresses the challenge of any-to-any routing in low-power networks, and differs from existing opportunistic routing protocols in the following ways:

Topology ORPL uses a rooted multipath topology (a DODAG) rather than a full mesh, aiming for inexpensive topology maintenance. We show that opportunistic routing makes possible to take shortcuts between different branches of the topology, significantly improving any-to-any routing.

Forwarding Decision Traditionally, the forwarding decision in opportunistic routing is based on a forwarder list included in the packet header. In ORPL, no additional header is used, and the forwarding decision is left to the receiver based on the gradient and the nodes in the sub-DODAG.

Routing State ORPL replaces traditional routing tables by *routing sets* that are propagated and merged through the network in a scalable way. The routing set is represented either as a bitmap or a Bloom filter.

Bloom filters [5] find many other applications in networking [7], typically for multicast [22], loop detection, or queue management. A notable example is CBFR [34], a routing protocol for mobile sensor networks that uses Bloom filters. Targeting (always-on) mobile nodes, CBFR proposed the use of counting Bloom filters for gradual forgetting of routing information. This inspired us to use aging Bloom filters for ORPL's routing set.

7. CONCLUSION

We presented ORPL, an opportunistic routing protocol that provides any-to-any routing in low-power IPv6 networks. The opportunistic nature of ORPL brings low latency and robustness to duty-cycled networks. By adopting a rooted topology and moving the forwarding decision from the sender to the receiver, ORPL routes to any destination using simple, scalable routing sets. Our testbed-based evaluation demonstrates the benefits of ORPL over state-of-the-art solutions and for various traffic patterns, including when routing along probabilistic Bloom filters. We believe the concepts of ORPL – in particular opportunistic routing based on routing sets – to be also applicable and useful in other contexts than wireless sensor networks.

Acknowledgments

We would like to thank the CIR Lab in Singapore for providing the Indriya testbed, and Federico Ferrari for his help in setting up the LWB experiments. We also thank Sébastien Dawans for his feedback in early stages of this work. This work was partly funded by the European Commission through CALIPSO (contract number FP7-ICT-2011.1.3-288879) and by SSF, the Swedish Foundation for Strategic Research.

8. REFERENCES

[1] F. Ashraf, N. H. Vaidya, and R. Kravets. Any-MAC: Extending any Asynchronous MAC with Anycast to Improve Delay in WSN. In *Proceedings of the Conference on Sensor, Mesh and Ad Hoc Communications and Networks (IEEE SECON)*, 2011.

[2] F. Ashref, R. H. Kravets, and N. H. Vaidya. Exploiting Routing Redundancy using MAC Layer Anycast to Improve Delay in WSN. *SIGMOBILE Mob. Comput. Commun. Rev.*, 14, 2010.

[3] P. Basu and C.-K. Chau. Opportunistic Forwarding in Wireless Networks with Duty Cycling. In *Proceedings of the Workshop on Challenged Networks (ACM CHANTS)*, 2008.

[4] S. Biswas and R. Morris. ExOR: Opportunistic Multi-Hop Routing for Wireless Networks. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (ACM SIGCOMM)*, 2005.

[5] B. H. Bloom. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Commun. ACM*, 13(7):422–426, July 1970.

[6] F. Bonomi, M. Mitzenmacher, R. Panigrah, S. Singh, and G. Varghese. Beyond Bloom Filters: From Approximate Membership Checks to Approximate State Machines. *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (ACM SIGCOMM)*, 2006.

[7] A. Broder and M. Mitzenmacher. Network Applications of Bloom Filters: A Survey. In *Internet Mathematics*, pages 636–646, 2002.

[8] N. Burri, P. V. Rickenbach, and R. Wattenhofer. Dozer: Ultra-Low Power Data Gathering in Sensor Networks. In *Proceedings of the Conference on Information Processing in Sensor Networks (ACM/IEEE IPSN)*, 2007.

[9] S. Chachulski, M. Jennings, S. Katti, and D. Katabi. Trading Structure for Randomness in Wireless Opportunistic Routing. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (ACM SIGCOMM)*, 2007.

[10] D. S. J. De Couto, D. Aguayo, J. Bicket, and R. Morris. A High-Throughput Path Metric for Multi-Hop Wireless Routing. *Wirel. Netw.*, 11(4):419–434, July 2005.

[11] M. Doddavenkatappa, M. C. Chan, and A. Ananda. Indriya: A Low-Cost, 3D Wireless Sensor Network Testbed. In *Proceedings of the Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities (TridentCom)*, 2011.

[12] H. Dubois-Ferriè andre, M. Grossglauser, and M. Vetterli. Valuable Detours: Least-Cost Anypath Routing. *IEEE/ACM Trans. Netw.*, 19(2), 2011.

[13] A. Dunkels. The ContikiMAC Radio Duty Cycling Protocol. Technical Report T2011:13, Swedish Institute of Computer Science, 2011.

[14] A. Dunkels, B. Gronvall, and T. Voigt. Contiki - A Lightweight and Flexible Operating System for Tiny Networked Sensors. In *Proceedings of the Conference on Local Computer Networks (IEEE LCN)*, 2004.

[15] A. Dunkels, F. Österlind, N. Tsiftes, and Z. He. Software-based On-line Energy Estimation for Sensor Nodes. In *Proceedings of the Workshop on Embedded Networked Sensor Systems (IEEE Emnets)*, 2007.

[16] S. Duquennoy, F. Österlind, and A. Dunkels. Lossy Links, Low Power, High Throughput. In *Proceedings of the Conference on Embedded Networked Sensor Systems (ACM SenSys)*, 2011.

[17] P. Dutta, S. Dawson-Haggerty, Y. Chen, C.-J. M. Liang, and A. Terzis. Design and Evaluation of a Versatile and Efficient Receiver-Initiated Link Layer for Low-Power Wireless. In *Proceedings of the Conference on Embedded Networked Sensor Systems (ACM SenSys)*, 2010.

[18] F. Ferrari, M. Zimmerling, L. Mottola, and L. Thiele. Low-Power Wireless Bus. In *Proceedings of the Conference on Embedded Networked Sensor Systems (ACM SenSys)*, 2012.

[19] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh. Efficient Network Flooding and Time Synchronization with Glossy. In *Proceedings of the Conference on Information Processing in Sensor Networks (ACM/IEEE IPSN)*, 2011.

[20] J. Flathagen, E. Larsen, P. Engelstad, and O. Kure. O-CTP: Hybrid Opportunistic Collection Tree Protocol for Wireless Sensor Networks. In *Proceedings of the Workshop on Practical Issues in Building Sensor Network Applications (IEEE SenseApp)*, 2012.

[21] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis. Collection Tree Protocol. In *Proceedings of the Conference on Embedded Networked Sensor Systems (ACM SenSys)*, 2009.

[22] Z. Heszberger, J. Tapolcai, A. GulyÃąs, J. Biro, A. Zahemszky, and P.-H. Ho. Adaptive Bloom Filters for Multicast Addressing. In *Proceedings of the Workshop on High-Speed Networks (IEEE HSN)*, 2011.

[23] JP. Vasseur and J. Hui and S. Dasgupta and G. Yoon. RPL Deployment Experience in Large Scale Networks. IETF draft-hui-vasseur-roll-rpl-deployment-01, WiP.

[24] J. Kim, X. Lin, and N. B. Shroff. Optimal Anycast Technique for Delay-Sensitive Energy-Constrained Asynchronous Sensor Networks. *IEEE/ACM Trans. Netw.*, 19(2):484–497, Apr. 2011.

[25] M. Kovatsch, S. Duquennoy, and A. Dunkels. A Low-Power CoAP for Contiki. In *Proceedings of the Workshop on Internet of Things Technology and Architectures (IEEE IoTech)*, 2011.

[26] O. Landsiedel, E. Ghadimi, S. Duquennoy, and M. Johansson. Low Power, Low Delay: Opportunistic Routing meets Duty Cycling. In *Proceedings of the Conference on Information Processing in Sensor Networks (ACM/IEEE IPSN)*, 2012.

[27] P. Levis, N. Patel, D. Culler, and S. Shenker. Trickle: A Self-Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks. In *Proceedings of the Symposium on Networked Systems Design & Implementation (USENIX NSDI)*, 2004.

[28] S. Liu, K.-W. Fan, and P. Sinha. CMAC: An Energy-Efficient MAC Layer Protocol using Convergent Packet Forwarding for Wireless Sensor Networks. *ACM Trans. on Senor Networks*, 5, 2009.

[29] X. Mao, X.-Y. Li, W.-Z. Song, P. Xu, and K. Moaveni-Nejad. Energy Efficient Opportunistic Routing in Wireless Networks. In *Proceedings of the Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (ACM MSWiM)*, 2009.

[30] D. Moss and P. Levis. BoX-MACs: Exploiting Physical and Link Layer Boundaries in Low-Power Networking. Technical Report SING-08-00, Stanford, 2008.

[31] V. Paruchuri, S. Basavaraju, A. Durresi, R. Kannan, and S. S. Iyengar. Random Asynchronous Wakeup Protocol for Sensor Networks. In *Proceedings of the Conference on Broadband Communications, Networks, and Systems (IEEE BROADNETS)*, 2004.

[32] B. Pavković, F. Theoleyre, and A. Duda. Multipath Opportunistic RPL Routing over IEEE 802.15.4. In *Proceedings of the Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (ACM MSWiM)*, 2011.

[33] M. V. Ramakrishna and J. Zobel. Performance in Practice of String Hashing Functions. In *Proceedings of the Conference on Database Systems for Advanced Applications (DASFAA)*, 1997.

[34] A. Reinhardt, O. Morar, S. Santini, S. Zöller, and R. Steinmetz. CBFR: Bloom Filter Routing with Gradual Forgetting for Tree-structured Wireless Sensor Networks with Mobile Nodes. In *Proceedings of the Symposium on a World of Wireless Mobile and Multimedia Networks (IEEE WoWMoM)*, 2012.

[35] G. Schaefer, F. Ingelrest, and M. Vetterli. Potentials of Opportunistic Routing in Energy-Constrained Wireless Sensor Networks. In *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*, 2009.

[36] K. Srinivasan, M. Jain, J. I. Choi, T. Azim, E. S. Kim, P. Levis, and B. Krishnamachari. The κ Factor: Inferring Protocol Performance using Inter-Link Reception Correlation. In *Proceedings of the Conference on Mobile Computing and Networking (ACM MobiCom)*, 2010.

[37] K. Srinivasan, M. A. Kazandjieva, S. Agarwal, and P. Levis. The β Factor: Measuring Wireless Link Burstiness. In *Proceedings of the Conference on Embedded Networked Sensor Systems (ACM SenSys)*, 2008.

[38] S. Unterschütz, C. Renner, and V. Turau. Opportunistic, Receiver-Initiated Data-Collection Protocol. In *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*, 2012.

[39] T. Winter (Ed.), P. Thubert (Ed.), and RPL Author Team. RPL: IPv6 Routing Protocol for Low power and Lossy Networks, Mar. 2012. RFC 6550.

[40] M. Zorzi and R. R. Rao. Geographic Random Forwarding (GeRaF) for Ad Hoc and Sensor Networks: Energy and Latency Performance. *IEEE Trans. on Mobile Computing*, 2(4):349–348, 2003.