Poster Abstract: Opportunistic RPL

Simon Duquennoy[1] and Olaf Landsiedel[2]

[1] Swedish Institute of Computer Science (SICS), `simonduq@sics.se`
[2] Chalmers University of Technology, `olafl@chalmers.se`

1 Introduction

Sensor nodes constituting Wireless Sensor Networks (WSN) are often battery-operated and have limited resources. To save energy, nodes sleep most of the time, and wake up periodically to handle communication. Such radio duty cycling poses a basic trade-off between energy and latency.

In previous work, we have shown that opportunistic routing is an efficient way to achieve low-latency yet energy efficient data collection in WSN (ORW [3]). In this paper, we extend this approach to the context of low-power IP networks, where nodes need to be addressed individually and where traffic patterns are irregular. We present ORPL, an opportunistic extension of RPL [5], the standard, state-of-the-art routing protocol for low-power IP networks. We discuss our preliminary results obtained with Contiki in a 137-node testbed.

2 Overview of ORPL

ORPL routes data traffic following a DODAG (Destination-Oriented Directed Acyclic Graph) in an opportunistic fashion. Like ORW [3], it relies on anycast over a low-power listening MAC. 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) offers sufficient routing progress, acknowledges the packet and forwards it.

ORPL performs any-to-any routing in a similar fashion as RPL, following a two step process. The data is first routed up in the DODAG until finding a common ancestor, then it is routed down to the destination. Routing upwards is done exactly as in ORW, along a gradient. Routing downwards, however, cannot rely only on the routing metric. Merely routing away from the root is not enough to ensure the destination will be reached. Because ORPL relies on anycast and opportunistic routing, we opt for routing-table free solution. Instead, each node maintains a set of nodes that are below them in the DODAG – their sub-DODAG. This information allows to decide whether a node is on the way to the destination or not. We use Bloom filters to represent this set of reachable nodes in a compact way and with fixed size, which enables efficient propagation through the network.

3 Design

This section describes the basic mechanisms of ORPL.

Routing Decision In ORPL, nodes anycast packets instead of electing a next hop and unicasting to it. Nodes receiving a packet choose whether to forwards it or not, and send an acknowledgment only in case they act as next hop. The routing decision at node N for a packet sent by S with destination D is as follows: If the packet is marked as going up in the DODAG (this information is included in RPL's IPv6 extension headers), then N forwards it *iff* $Rank_N < Rank_S$. If the packet is marked as going down in the DODAG, then N forwards it *iff* $Rank_N > Rank_S \wedge D \in SubDODAG_N$

Bloom Filters In ORPL, nodes need to know the set of nodes composing their sub-DODAG. To enable storing this set in memory-constrained devices and propagating it within size-restricted network frames, we resort to Bloom filters [1]. Each node populates its Bloom filter with its direct children, and propagate the filters through the RPL DIO messages (Trickle beacons). A node receiving a DIO from a node already in its Bloom filter merges the newly received filter with its own, by simply bitwise-ORing the filters.

Bloom filters are subject to false positives, meaning that sometimes a node believes the destination is in its sub-DODAG while it is not. When such situation is suspected (non-acked anycast), the node stores the packet ID in a blacklist, and sends the packet back to the previous hop. The previous hop will anycast again, looking for another path. If repeating this process recursively is not enough to reach the destination, the fallback behavior is to perform a network-wide multicast. This is inspired by CBFR [4], which extends CTP with downwards routing through network-wide multicast.

Link Quality Estimation ORPL requires to maintain link quality estimates of every reachable neighbor to decide whether to include nodes and merge Bloom filters. As ORPL targets scenarios with any-to-any, irregular traffic, we need a to maintain link quality even in the absence of traffic. In our current design, we us acknowledged broadcast to achieve efficient link estimation. With low-power-listening, broadcast are sent repeatedly for exactly one wakeup period. Nodes receiving a broadcast send a 802.15.4 link-layer acknowledgement extended with the receiver's address. The sender counts the received ACKs to estimate link quality.

Networks Dynamics and Inconsistency Detection Low-power networks are dynamic, often resulting in varying logical topology. To enable forgetting that given nodes were in one's sub-DODAG in the past, we use aging Bloom filters. We use a simple approach where two filters are maintained, one *active*, and one *warmup*. The filters are swapped and emptied periodically. ORPL inherits RPL's *datapath validation* mechanism to detect and repair loops. We extend this mechanism with a systematic lookup of the packet source in the local Bloom filter. This

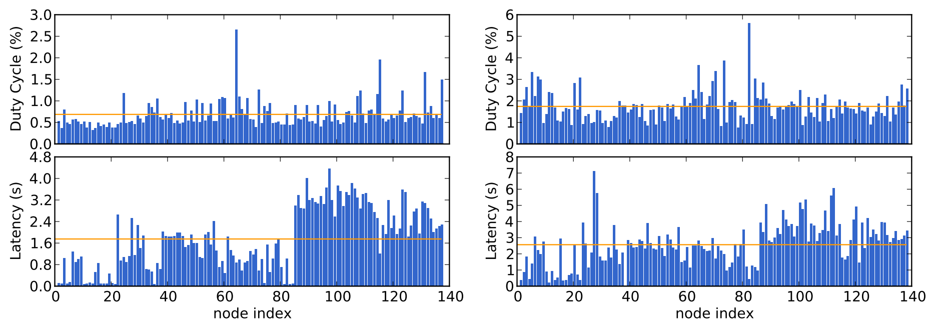

Fig. 1: In both *up-traffic* (left) and *down-traffic* (right) experiments in Indriya, the average duty cycle is kept below 2% and latency below 3 s. Downwards routing is not as efficient as upwards but remains in the same order of magnitude.

allows detecting, for instance, cases where a node with lower rank (a parent) is in one's filter. When this happens, the Bloom filters are switched and the Trickle reset in order to repair inconsistencies.

4 Preliminary results

We implement ORPL in Contiki, based on ContikiRPL and the ContikiMAC duty cycling protocol (used with a wakeup interval of 500 ms). We run our experiments (summarized in Figure 1) in the Indriya testbed [2] which has 137 Tmote sky spanning 3 floors of an office building. Our first experiment (*up-traffic*) is a collect-only scenario, with an inter-packet interval of 8 min, used as a baseline. Our second experiment (*down-traffic*) is a reverse-collect, where the sink sends periodically data to every node in the network. The inter-packet interval is 4 seconds, resulting in a load similar to that of the first experiment. To isolate the effects of filter propagation and the adaptation to network dynamics, we use a three steps scenario where (1) ranks are calculated in the network, (2) bloom filters are propagated and (3) sink-to-nodes traffic is started. Each experiment runs for 80 minutes.

In the *up-traffic* experiment, ORPL achieves a packet delivery ratio of 99%, a network duty cycle of 0.69%, and an average latency of 1.75 s. The *down-traffic* experiment obtains an average delivery ratio of 95%, a duty cycle of 1.74% and latency of 2.57 s. These results are promising in terms of energy-latency balance, keeping the average duty cycle below 2% and latency below 3 s in a testbed with >100 addressable nodes within a hop count of 1–9. Note that in both experiments, the energy expenditure is well balanced among nodes, which is an important property for network lifetime.

Downwards routing in RPL (or ORPL) is by nature more challenging than upwards routing, because the topology and link metrics are build in direction of the sink. In RPL, upwards traffic is used to build and heal the topology throughout the lifetime of the network. The other reason behind the performance difference between up and down traffic is the fact that the sink is not duty cycled. Sending to the sink is therefore more energy efficient and fast than sending to other nodes.

5 Conclusion and Future work

Our initial prototype implementation and testbed experiments demonstrate the feasibility of our approach. There are a number of key questions we need to address in our future work. First, we will look in more detail at Bloom filter propagation, maintenance and aging. There is a difficult trade-off between efficiency and correctness of the topology. We want to investigate and evaluate policies to recover from false positives when routing downwards. We may also consider resetting the network in case the filters contain more false positives, and changing the seed used for hash calculation, aiming at converging to a state with fewer false positives. Finally, we plan to diversify our experiments across scenarios with different node density, and compare ORPL to traditional any-to-any routing in RPL.

Acknowledgments

This work was partly funded by the European Commission through CALIPSO (contract number FP7-ICT-2011.1.3-288879). We'd like to thank the Communications and Internet Research Lab in Singapore for providing the Indriya testbed.

References

1. B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, July 1970.
2. M. Doddavenkatappa, M. C. Chan, and A. Ananda. Indriya: A Low-Cost, 3D Wireless Sensor Network Testbed. In *TridentCom: Proc. of the Int. ICST Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities*, 2011.
3. O. Landsiedel, E. Ghadimi, S. Duquennoy, and M. Johansson. Low Power, Low Delay: Opportunistic Routing meets Duty Cycling. In *Proceedings of the International Conference on Information Processing in Sensor Networks (ACM IPSN 2012)*, Beijing, China, Apr. 2012.
4. 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 IEEE International Symposium on a World of Wireless Mobile and Multimedia Networks (WoWMoM 2012)*, San Francisco, CA, USA, June 2012.
5. T. Winter (Ed.), P. Thubert (Ed.), and RPL Author Team. RPL: IPv6 Routing Protocol for Low power and Lossy Networks. RFC 6550.