Let healthy links bloom: Scalable Link Checks in Low-Power Wireless Networks for Smart Health

Cenk Gündogan, Cédric Adjih, Oliver Hahm, Emmanuel Baccelli

Let Healthy Links Bloom: Scalable Link Checks in Low-Power Wireless Networks for Smart Health

C. Gündogan
FU Berlin, Germany
cenk.guendogan@fu-berlin.de

C. Adjih, O. Hahm, E. Baccelli
INRIA, France
firstname.lastname@inria.fr

ABSTRACT

Low-power, memory-efficient networking mechanisms are an essential part of the Internet of Things (IoT) and thus, fundamental to Smart Health applications leveraging IoT. This paper presents Bloom-RPL, an optimization of RPL, the standard low-power routing protocol for IoT. This paper evaluates Bloom-RPL both on an emulator, and on an IoT testbed using real hardware, to conclude that Bloom-RPL dramatically improves RPL's link check scalability with respect to both IoT device density and convergence time needed to detect a link break.

Keywords

RPL; Bloom-RPL; Bloom filter; scalable unreachability detection

1. INTRODUCTION

Sensor networks are a substantial part of the devices expected in the Internet of Things (IoT). Sensors are IoT devices used for distributed and automated monitoring of various environmental parameters such as temperature, movement, noise or radioactivity levels etc. Sensors are also expected to be leveraged in various smart health applications leveraging IoT, such as quantified-self applications [1], and medical applications such as ambient assisted living applications [2]. While some of these sensors will connect to the network via wire or power line communication, most sensors will use radio communications.

Typically, a number of sensors are scattered in the zone to be monitored e.g. a body, a room or a house (for on-premises smart health applications) or a small building (e.g. for smart health applications at a hospital). Each sensor then monitors the parameters to be measured in its vicinity and communicates through its single radio interface with its peers, spontaneously creating a wireless network. Using this network, sensors self-organize distributed computations, or convergecast, *i.e.* information gathering at a central control point – which is generally called the *sink*, in this context.

As shown in Fig. 1, sensors can directly communicate with the sink (single-hop case), or may require peers to forward information towards the sink (multi-hop case), because the sink is outside of its radio range e.g. due to deliberately lower power transmissions to save energy or limit interferences, or due to heterogeneous wireless technologies. Appropriate network protocols are thus required to enable each sensor to, on one hand, establish direct communication with peers that are within its radio range, and on the other hand, establish communication with other devices, reachable through peers with which the sensor can directly communicate.

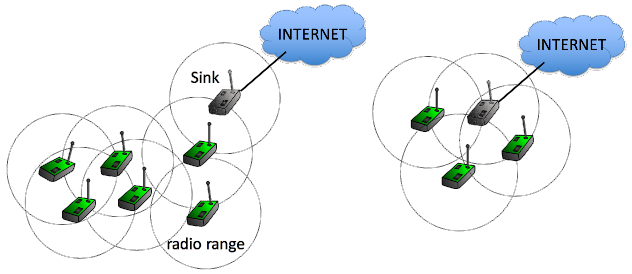

Figure 1: Low-power wireless network with sink, single hop case (right), multi-hop case (left).

Several technologies are used for direct, single-hop wireless sensor communication at the link layer. Prominent low-power radio technologies include for instance IEEE 802.15.4, Bluetooth, DASH7, IEEE 802.11ah. The network obtained using such radio technologies is thus traditionally called a Low-power Lossy Network (LLN). Complementary mechanisms are necessary at the network layer to enable multi-hop communication in

LLNs. For that matter, recent work enables the use of IP protocols [3] in LLNs, allowing IoT applications that are natively interoperable, end-to-end, with the rest of the Internet . In this paper, we will focus on such approaches, using an IPv6 protocol suite on top of LLNs.

The contributions of this paper are as follows. First we overview the interplay of standard IPv6 protocols for link check and routing in LLNs. Then we analyze available implementations of these protocols: we show via testbed measurements that link check mechanisms used in these implementations offer poor scalability w.r.t. node density and convergence time. We then introduce Bloom-RPL, an alternative link check mechanism for RPL, which we also evaluate via testbed measurements showing significantly improved scalability.

2. LLN ROUTING AND LINK-CHECKING

In LLNs, mechanisms are needed at the network layer to (i) check link viability between devices, and (ii) establish paths across multiple intermediate (viable) links, i.e. routing.

2.1 Standard Link Checking in LLNs

The most basic form of link check verifies that a neighbor node is *reachable* through an interface connecting to a given link. Reachability of node B over a link is generally defined for node A as such: B can receive link layer transmissions from A and vice versa, i.e. the link between A and B is *bidirectional*.

2.1.1 Neighbor Discovery Protocol

The standard link-check protocol in the IPv6 suite is NDP [4]. NDP performs a number of tasks, among others: Neighor Unreachability Detection (*NUD*), Router and Prefix Discovery.

To perform its tasks, NDP uses ICMP messages. Router Advertisements (*RA*) and Router Solicitations (*RS*) respectively announce and request default router information. Neighbor Advertisements (*NA*) and Neighbor Solicitations (*NS*) respectively announce and request nodes' addresses for purposes including duplicate address detection, link-layer address resolution, and reachability verification. NDP maintains on each node a *Neighbor Cache*, a set of entries tracking known neighbor's link-layer address, role (router or host), and reachability (determined by *NUD*).

6LoWPAN-ND [5] is an optimization of NDP for LLNs to reduce multicast signaling and support sleeping hosts. 6LoWPAN-ND leverages the central role of the sink (called 6LoWPAN Border Router (*6LBR*)) to perform (multi-hop) distribution of prefixes and context information, as well as duplicate address detection.

2.1.2 Other Techniques

Various mechanisms have been proposed to reduce bits-over-the-air for link checks. For example in [6], authors propose to group several suffixes in a Bloom filter [7] sent in a "compact neighbor discovery" message (CND) instead of several IPv6 ND messages. In [8] authors propose to represent neighbors in hello messages with a Bloom filter, to efficiently check link bidirectionality. More elaborate link checks evaluate links beyond mere reachability: a number of *metrics* have been designed, such as for example ETX [9] or DAT [10], which associate to a link a value (in $\mathbb{N}$ or $\mathbb{R}$) representing the quality of the link. However, in this paper we focus on basic link checks testing bidirectionality, associating to a link a value in $\{0, 1\}$.

2.2 Standard Routing in LLNs

Routing in LLNs is typically performed with RPL [11], which organizes routers along a Destination Oriented Directed Acyclic Graph (DODAG), rooted at the sink (router S in Fig. 2). The sink initiates the formation of the DODAG by periodically originating (DIO) messages via link-local multicast. DIO messages carry information that include the root's identity, the routing metric, and the originating router's depth in the DODAG, called the *rank*. A router joins the DODAG based on DIOs from its neighbors, and can thereby determine its own rank, choosing the parent that would yield the smallest rank. Once a router has joined the DODAG, the router has a path to the sink through its parent, and the router can in turn originate its own DIO messages. RPL thus provides paths from a router to the sink while requiring minimal forwarding/routing information storage.

In order to decrease control traffic overhead, the transmission rate of DIO messages follow a Trickle [12] policy which aims at pruning unnecessary transmissions. When a node's data differs with its neighbors', that node communicates quicker to resolve the inconsistency, otherwise it slows down the propagation of DIO messages.

2.2.1 RPL extensions & other approaches

RPL defines complementary mechanisms for routing from sink to sensors and extensions have been proposed to route from sensor to sensor without going through the sink. For example, ORPL [13] uses a technique based on Bloom filters [14] to detect shorter paths from sensor to sensor. P2P-RPL [15] uses a reactive routing technique to discover and establish paths from sensor to sensor without going through the sink. However, such mechanisms are optionally, and the focus of this paper is paths from sensors to sink (i.e. convergecast), a purpose for which RPL is currently the dominant approach.

2.3 RPL and NDP in Practice

RPL specification mandates that parent-child link checks are performed before being used, but does not specify how to perform these checks. RPL instead relies on external mechanisms to perform link checks, either (i) network layer signaling, e.g. NDP/NUD such as described in section 2.1, or (ii) lower layer signaling e.g. absence of link-layer acknowledgements, or (iii) upper

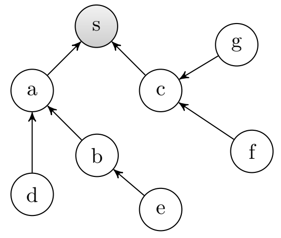

Figure 2: DODAG rooted at router S.

layer signaling e.g. absence of end-to-end acknowledgement of application data.

2.3.1 Reactive vs Proactive Link Checks

Parent-child link checks can either be performed *reactively* or *proactively.* In a reactive scheme, link checks are performed only when applications need to send data over the link. In case the link to the parent turns out to be defunct, the child starts the procedure to rejoin the DODAG through another parent. A reactive approach works fine in a single-hop case, while requiring minimal amounts of control traffic and small convergence time in the order of link-layer RTT if link-layer signaling is usable, or else, end-to-end application layer RTT. However, in multi-hop cases, reactive approaches yield terrible worst-case performance as its mechanism may have to be performed *recursively* along the path towards the sink, potentially leading to unmanageable convergence time. For this reason, proactive approaches are favored, whereby parent-child link checks are performed regularly, independently of application data triggers.

2.3.2 NDP vs RPL Functionalities Overlap

In practice, NDP and RPL functionalities overlap substantially. Router and prefix discovery, next-hop determination and address autoconfiguration are functionalities provided by NDP that can be substituted entirely by RPL. The multi-hop propagation of context information used for header compression is not supported with basic RPL specification [11]. However, compression is not mandatory, and if desired, could easily be piggy-backed with the prefix information dissemination mechanism of RPL. In minimal network setups, nodes can use stateless address autoconfiguration [16] to derive IPv6 addresses from their (assumed) unique link-layer addresses, which makes the address resolution and duplicate detection functionalities of NDP superfluous. Under these circumstances, unreachability detection is the only mandatory mechanism not supported by RPL.

We evaluated ROM/RAM usage due to NDP in a minimal IoT application implemented on top of RIOT [17] that sends sensor readings from a sensor to a sink using IPv6 and 6LoWPAN protocols. By removing NDP and the neighbor cache from RIOT's network stack, the same functionality can be achieved with $\approx$ 5,5 kB less ROM and $\approx$ 2 kB less RAM. For perspective, this requires roughly 25% ROM for the binary on popular IoT hardware based on 32-bit ARM Cortex M, thus freeing memory (usually in high-demand on low-end IoT devices) that can be used for high layer applications.

However, as analyzed above, an approach eliding NDP requires to provide mechanism that perform link checks. In the following, we thus study link check mechanisms directly built into RPL.

2.3.3 RPL Link Checks: Available Implementations

To check links between a child and its parent, the most popular RPL implementations available (e.g. in Contiki [18] and RIOT [17]), use a mix of dedicated, proactive unicast traffic (DIS/DIO message handshakes mimicking NDP's NS/NA exchanges) and an ETX metric [9] computed on both user traffic and control traffic. For simplicity, we consider an approximation of this technique which uses only DIS/DIO message handshakes, which results in the same number of data and control transmissions and the same amount of bytes over-the-air. The scalability of such link check mechanisms is questionable with greater node density, quicker convergence time. In the following we measure the performance of the available RPL implementations using DIS/DIO link checks and we propose an alternative link check mechanism which, as our comparative measurements show, provides much better scalability.

3. DIS/DIO LINK CHECK PERFORMANCE

In the following, we will measure the performance of DIS/DIO link checks. We first use a simple topology of three nodes, emulated on RIOT native [19]: One DODAG root (a) and two children (b, c), initially connected via bidirectional links $\overline{ab}, \overline{ac}$ and $\overline{bc}$. The links $\overline{ab}$ and $\overline{ac}$ alternate every 60 seconds between *up* and *down* (when one link is up, the other is down). We then measure in Fig. 3 the average time during which connectivity between child and parent remains down, when connectivity is lost, and link check period is L_p. Connectivity was checked by sending sensor readings from b and c to a, and we measured the average time until which an acknowledgement from a was received respectively. Unsurprisingly, we observe that average duration of connectivity loss grows approximately in $\frac{L_p}{2}$. In order to minimize down-time, more frequent link checks should be used.

However, more frequent link checks lead to significantly more control traffic, as shown in Fig. 4. In this experiment, we measured the amount of DIS and DIO packets received by the DODAG root, with varying number of children and link check periods. All nodes were in hearing range of the root node, thus forming a star topology and the experiments were carried out on real hardware in the IoT-LAB testbed [20]. We observe in Fig. 4 that RPL control traffic explodes when the number of children and/or the link check frequency increases. We can therefore conclude that this approach is not scalable w.r.t. node density and undetected downtime reduction.

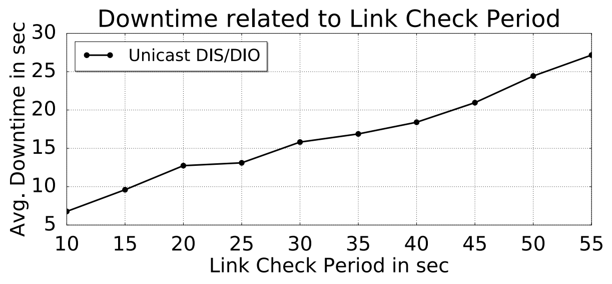

Figure 3: Average duration of connectivity loss w.r.t. link check interval L_p.

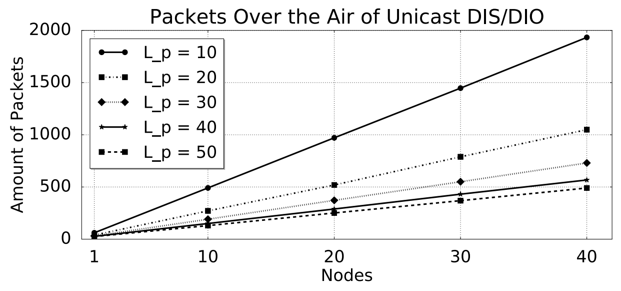

Figure 4: RPL control traffic transmitted with varying link check interval L_p and node density.

4. BLOOM-RPL

In this section, we present an optimization of RPL that substantially improves the scalability of link-checks between parent and child in the DAG, based on an approach using Bloom filters, loosely inspired from [8].

4.1 Overview of Bloom-RPL

The parent-child relationship that is formed with RPL's DODAG tree structure usually resembles an asymmetric relationship, where few nodes serve as parents for multiple children. Checking and caching bidirectionality information requires a parent to store an amount of states equal to the number of its children and thus is directly related to the node density.

Contrary to the unicast DIS/DIO approach described in section 3, Bloom-RPL reuses RPL's multicast control messages, so that bidirectionality of links between parent and children are verified with less traffic.

However, using multicast DIO messages to verify link bidirectionality requires children information in each of such messages, leading to message sizes that grow proportionally to the number of children instead of being constant. To mitigate this fact, Bloom-RPL utilizes Bloom filters to compress the children information, thusly making it more suitable for a multicast dissemination. The main drawback of Bloom-RPL is that Bloom filters introduce loss of information and potential false positives during link checking. Nevertheless, as demonstrated experimentally below, Bloom-RPL provides a functional and advantageous solution.

4.1.1 Additions to RPL

The following represents a list of conceptual additions that Bloom-RPL introduces to RPL.

- *NBF:* Neighborhood Bloom filter used to compress and store the children information.
- NBF_a/NBF_i*:* Active/Inactive bitmap of the double buffering technique.
- NBF_w/NBF_r*:* Warm-up and reset time of the double buffering technique.
- *BBF:* Bloom filter to blacklist link-local IPv6 addresses of parents with unidirectional links.
- NAO/NAO_d*:* Neighborhood Announcement Option in DIO to propagate *NBF* and time in seconds to delay it.
- *PAO:* Parent Announcement Option in multicast DIS/DIO to propagate parent information.
- L_p*:* Period in seconds to check link bidirectionality initiated by a child.
- L_{cr}/L_{cri}*:* Number of retries until link is marked unidirectional and interval in ms between retries.

4.1.2 Link Checking Procedure

Bloom-RPL adds NAO and PAO options to classical DIO messages from RPL. They are sent either at expiration of Trickle intervals or as replies to DIS from children. The NAO advertises the current NBF and the PAO advertises parent(s).

The first part of the protocol consists of parents discovering their children. Every time a child is confirmed to the parent, the *NBF* is updated with the child's link-local IPv6 address. Confirmation occurs when an unicast DIS, or DAO (in Storing Mode) is sent from child to parent. Upward control messages are also used, if a PAO is included and matches the parent's link-local IPv6 address.

The second part of the protocol consists in a child checking if its parent actually discovered it: this is verified when the child link-local IPv6 address appears in the NAO sent by its parent. If so, then the link towards the parent will be marked as bidirectional. Otherwise: the child will retry the link checking with DIS messages (unicast or multicast) containing a matching PAO, for L_{cr} times. If this procedure fails, a parent's link-local IPv6 address will be blacklisted (stored in the BBF), its subsequent DIO messages will be ignored and the node will search for another parent. Ultimately, all nodes end up with a parent with a symmetric link.

Finally, to handle the case of disappearing links, if the last received NAO is older than L_p seconds, the parent will be marked as unidirectional and the link check procedure is repeated by soliciting a new NAO.

4.1.3 Bloom Filter Maintenance

While basic Bloom filters support insertion and set membership queries, it is not trivial to remove specific entries from the set. The simplest approach is to reset the NBF periodically to trigger the children to perform link checks. The disadvantage of this method is however, that a variety of children would start this procedure at the same time. To alleviate this case, we decided to use the double buffering technique [21]. Hence, our NBF consists of an active NBF_a and an inactive NBF_i bitmap of same size. Until a certain point in time NBF_w, entries will be inserted into NBF_a, and in both bitmaps thereafter (warm-up phase). Once time passes NBF_r, both bitmaps switch their roles and the former NBF_a will be reset (reset phase). For the dissemination of the children information inside NAOs, NBF_a is used and it is noteworthy, that due to warming up the Bloom filter, NBF_a consists of recent children information.

Piggy-backing optimization. Once a NAO is requested by DIS messages, a parent delays the solicited DIO for NAO_d seconds. Thus, subsequent solicitations falling into this interval can be aggregated and answered with only one multicast DIO in return. The time between link check retries (L_{cri}) should be greater than NAO_d.

4.2 Experimental Evaluation of Bloom-RPL

We present below experimental measurements evaluating Bloom-RPL, comparatively to the unicast DIS/DIO approach. Bloom-RPL was implemented on top of RIOT [17] and experiments were carried out on real hardware, on the IoT-Lab testbed [20]. In these experiments, we vary both the number of children of the root (i.e. the node density) and the link check frequency. We observe the network for periods of 10 minutes in row. Note that Trickle-induced multicast DIO messages decrease rapidly in the initial minute, and becomes negligible traffic thereafter. The size of NBF was set to 32 bytes in order to support node densities of up to 40 nodes with a false positive rate of less than 5% (derived from the well-known analytic formula, e.g. see [14]). For node densities of larger than 40 nodes and up to 100 nodes, a size of 64 bytes was chosen to provide a maximum false positive rate of $\approx 10\%$. The reset phase was entered after $NBF_r = 90$ seconds and the warm-up phase started after $NBF_w = \frac{NBF_r}{2}$ seconds.

Fig. 5 shows the ratio of link checks that were performed by Bloom-RPL in relation to unicast DIS/DIO. We note a drop in link checks performed by about 20% to up to 60%. This derives from the dissemination of compressed children information in each multicast DIO, which is in turn received by all children and triggers a refresh of certain link check periods. Additionally, the double buffering technique provides a Bloom filter that is pre-filled reasonably in each warm-up phase.

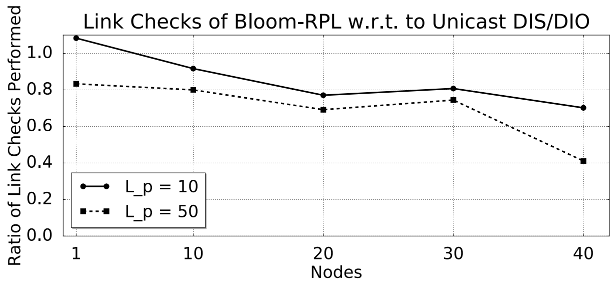

Figure 5: Comparison of link checks performed between unicast DIS/DIO and Bloom-RPL.

Fig. 6 depicts the amount of control traffic bytes over the air. We observe that Bloom-RPL generates up to 80% less control traffic in bytes compared to unicast DIS/DIO when the node density and/or the link check frequency is bigger, due to the fact that Bloom-RPL eliminates the unicast DIO traffic for link checks. Unsurprisingly, Bloom-RPL requires more control traffic for lowest densities (cases where parents have one child), but this represents a tolerable, small overhead in absolute value. The amount of control packet transmissions of Bloom-RPL compared to the DIS/DIO approach is shown in Fig. 7. We observe that Bloom-RPL requires 60-75% less control packets compared to DIS/DIO when node density grows to 40 nodes per radio range (savings would be even bigger in case of bigger densities).

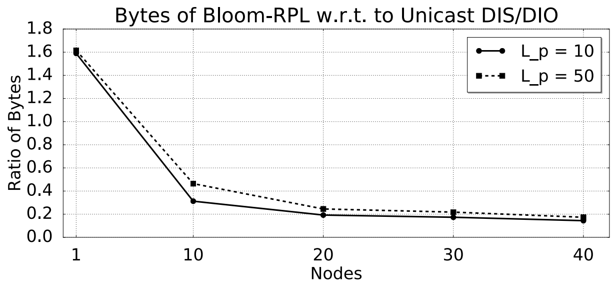

Figure 6: Ratio of control traffic in bytes between unicast DIS/DIO and Bloom-RPL.

In Fig. 8, we measure sensor data packet delivery ratio in a network with varying node density: each node sends to the sink a sensor reading periodically, with interval of 300 ms and 100 ms jitter. We observe that Bloom-RPL provides a functional network with an acceptable PDR (above 70%) when node density grows to 80 nodes per radio range, while the DIS/DIO approach requires so much control traffic that sensor reading traffic experiences a rather dysfunctional PDR around 30%.

5. CONCLUSION

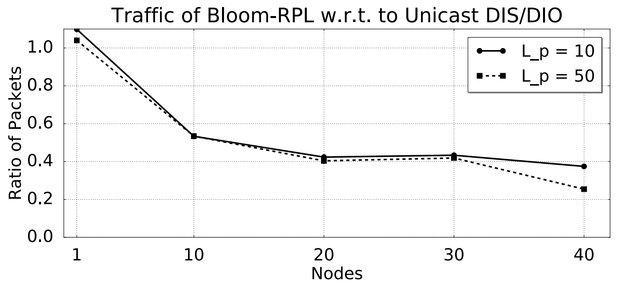

Figure 7: Ratio of control traffic between unicast DIS/DIO and Bloom-RPL.

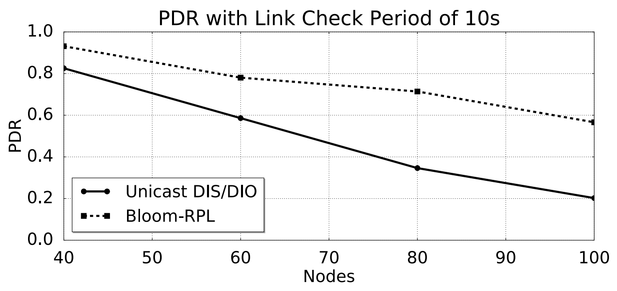

Figure 8: Avg. packet delivery ratio (PDR) between unicast DIS/DIO and for Bloom-RPL.

In this paper we have presented Bloom-RPL, an optimization of RPL's link check mechanism for low-power wireless networks using IPv6 in Smart Health applications. Both simulations and experiments on real hardware show that Bloom-RPL provides significantly improved scalability in terms of IoT device density and convergence time needed to detect a link break. Using an approach piggy-backing RPL's multicast traffic and compressing children information with Bloom filters, we achieved 75% reduction in control traffic packets and 80% less control traffic bytes when node density is large. Even more gains would be obtained with even larger densities. In effect Bloom-RPL does not only save energy, but also provides an acceptable PDR for larger node densities while detecting faster links that are down, compared to basic RPL implementations that are available to date. These properties allow to significantly decrease the amount of health sensor data that is lost, which can be critical in Smart Health scenarios.

6. REFERENCES

[1] M. Swan, "Sensor mania! the internet of things, wearable computing, objective metrics, and the quantified self 2.0," *Journal of Sensor and Actuator Networks*, 2012.

[2] A. Dohr *et al.*, "The internet of things for ambient assisted living," in *IEEE International Conference on Information Technology*, 2010.

[3] Z. Sheng *et al.*, "A survey on the IETF protocol suite for the internet of things: Standards, challenges, and opportunities," *IEEE Wireless Communications*, 2013.

[4] W. A. Simpson, D. T. Narten, E. Nordmark, and H. Soliman, "Neighbor Discovery for IP version 6 (IPv6)," RFC 4861, Oct. 2015. [Online]. Available: https://rfc-editor.org/rfc/rfc4861.txt

[5] Z. Shelby, S. Chakrabarti, and E. Nordmark, "Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)," RFC 6775, Oct. 2015. [Online]. Available: https://rfc-editor.org/rfc/rfc6775.txt

[6] P. Mutaf and C. Castelluccia, "Compact neighbor discovery: a bandwidth defense through bandwidth optimization," in *In proceedings of IEEE INFOCOM*, 2005.

[7] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, 1970.

[8] D. Ellard and D. Brown, "DTN IP Neighbor Discovery," IETF Draft draft-irtf-dtnrg-ipnd-01, Tech. Rep., 2010.

[9] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-efficient communication protocol for wireless microsensor networks," in *In Proceedings of IEEE HICSS*, 2000.

[10] H. Rogge and E. Baccelli, "Directional airtime metric based on packet sequence numbers for optimized link state routing version 2," Tech. Rep., 2016.

[11] T. Winter *et al.*, "RPL: Routing protocol for low power and lossy networks," 2012.

[12] P. Levis *et al.*, "The trickle algorithm," *Internet Engineering Task Force, RFC6206*, 2011.

[13] S. Duquennoy, O. Landsiedel, and T. Voigt, "Let the tree bloom: Scalable opportunistic routing with orpl," in *Proceedings of ACM SenSys*, 2013.

[14] A. Broder and M. Mitzenmacher, "Network applications of bloom filters: A survey," *In Internet Mathematics*, 2004.

[15] J. Martocci *et al.*, "Reactive discovery of point-to-point routes in low-power and lossy networks," *Internet Engineering Task Force, RFC6997*, 2013.

[16] D. T. Narten, T. Jinmei, and D. S. Thomson, "IPv6 Stateless Address Autoconfiguration," RFC 4862, Oct. 2015. [Online]. Available: https://rfc-editor.org/rfc/rfc4862.txt

[17] E. Baccelli *et al.*, "RIOT OS: Towards an OS for the Internet of Things," in *In proceedings of IEEE INFOCOM*, 2013.

[18] A. Dunkels, B. Grönvall, and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," in *IEEE LCN*, 2004.

[19] RIOT, "Native." [Online]. Available: https://github.com/RIOT-OS/RIOT/wiki/Family\%3A-native

[20] C. Adjih *et al.*, "FIT IoT-LAB: A Large Scale Open Experimental IoT Testbed," in *In Proceedings of IEEE WF-IoT*, 2015.

[21] S. Tarkoma *et al.*, "Theory and practice of bloom filters for distributed systems," *IEEE Communications Surveys and Tutorials*, 2012.