RNFD: Routing-Layer Detection of DODAG (Root) Node Failures in Low-Power Wireless Networks

Konrad Iwanicki
Faculty of Mathematics, Informatics and Mechanics
University of Warsaw, Poland
E-mail: iwanicki@mimuw.edu.pl

***Abstract*—While routing protocols for low-power wireless networks, such as CTP or RPL, handle link failures relatively well, node failures have received considerably less research attention. This paper thus studies crash-failures of destination nodes in distance-vector routing, that is, failures of so-called DODAG root nodes. First, it demonstrates empirically that handling root node crashes in existing state-of-the-art routing protocols leaves room for improvement or even fails completely in some cases. Second, based on an analysis of this behavior, the paper proposes RNFD, a new algorithm that explicitly tracks node failures at the routing layer. The algorithm is designed as a framework that complements rather than replaces regular route maintenance algorithms, which facilitates its integration with the existing protocols. Third, the paper evaluates a prototype implementation of RNFD through simulations and testbed experiments. In particular, it demonstrates that, with little information overhead, RNFD can speed up node failure detection by an order of magnitude and considerably reduce the traffic during the process.**

I. Introduction

The Internet of Things vision and the related standards have promoted routing to become important functionality for low-power wireless networks [1]. The goal of routing is to ensure that data from a source node can reach a destination node even if these nodes are outside each other's radio range, which is achieved by forwarding the data through intermediate nodes. A routing protocol thus provisions paths in a network along which data packets are forwarded. This entails not only discovering inter-node links and combining them into paths but also adapting or even tearing down these paths upon failures.

To this end, both link and node failures must be handled by a routing protocol. Due to the peculiarities of low-power wireless communication, managing links and their quality has been a major research topic, as elaborated in the next section. In effect, failures of individual links are handled well by state-of-the-art routing protocols. In contrast, node failures have received considerably less attention. This is likely because they can be handled eventually by the mechanisms for managing links: all links to a node that is down simply fail.

However, while this reasoning is correct, handling node failures explicitly—rather than just relying on link-oriented mechanisms—may improve the performance of a routing protocol. More specifically, let us consider destination crash-failures in protocols employing distance-vector routing techniques. These techniques are common in low-power wireless networks. In particular, in the state-of-the-art IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL) [2], distance-vector is utilized not only for routing from low-power devices via a border router to the Internet but also for collecting topology information by the router to enable source routing in the opposite direction. The reason for the focus on failures of destination nodes is in turn that such nodes are often important and their crashes affect many other nodes. For example, a power outage of a border router—a destination in RPL—is not uncommon and may preclude communication in the entire network. In general, detecting a failure of a destination node is a distributed process involving many nodes. It should be performed as quickly and with as little traffic as possible; otherwise, the nodes will waste resources by forwarding packets over broken paths to the failed destination and continuously trying to rebuild these paths.

This paper thus studies destination node crash-failure handling in distance-vector routing for low-power wireless networks. We start by identifying three main existing methods for maintaining routing paths that influence failure handling. From experiments with protocols implementing these methods in TinyOS and Contiki, we learn that they all leave room for improvement, in some cases even being incapable of detecting destination crashes. We analyze the failure detection process in the most advanced of the methods, adopted in RPL, and enumerate its drawbacks. The analysis leads us to RNFD, a new algorithm that addresses these drawbacks by explicitly tracking node failures. RNFD operates alongside existing routing path maintenance algorithms, thereby complementing rather than replacing them. It is also a framework algorithm, leaving several issues open to implementations, so that it can be integrated into different routing protocols. Based on simulations and testbed experiments, we conclude that, with a fixed information overhead, RNFD can outperform existing failure detection methods in both latency and traffic: the presented implementation handles destination crashes an order of magnitude faster and with a fraction of the traffic of the state-of-the-art path maintenance method, adopted in RPL.

The paper is organized as follows. Sect. II discusses related work. Sect. III shows the drawbacks of existing solutions. Sect. IV introduces RNFD. Sect. V–VII present the experiments conducted with RNFD. Finally, Sect. VIII concludes.

II. Background and Related Work

The peculiarities of low-power wireless networks have led to custom solutions for the many aspects of fault-tolerant routing.

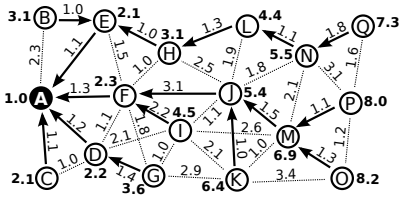
A. Link and Path Selection: DODAGs

To start with, since a node's communication range is hard to predict and varies in time, each node has to discover wireless links to other nodes in its range, so-called *neighbors*, and continuously assess the quality of the links (e.g., the packet reception rate, PRR) [3]. This is typically done by counting packets broadcast by a neighbor [3], counting link-layer acknowledgments for unicast packets forwarded to the neighbor [4], or by combining these methods, possibly also with physical signal metrics [5]. In general, link quality dynamics have received a lot of research attention, notably when it comes to correlating PRR with physical signal quality [6], link stability [7], influence of noise [8] and other environmental factors [9], and the impact of these properties on routing [10].

The links maintained by the nodes are used to build routing paths. To this end, in distance-vector techniques, each destination advertises itself to its neighbors, which forward the advertisement to their neighbors, and so on. A node receiving an advertisement from a neighbor records a routing entry for the destination, with the neighbor acting as a next hop on a path from the node to the destination. For fault tolerance, usually multiple next hops are recorded in an entry. Therefore, globally, the next-hop links of all nodes form a directed acyclic graph (DAG) with edges oriented toward the destination (cf. Fig. 1). Such a graph is commonly referred to as a *destination-oriented DAG* [2], or *DODAG* for short; the *parents* of a node are the node's next-hop neighbors toward the destination; the destination is the *root* of the graph; "*up*" refers to the direction toward the destination; "*down*" refers to the opposite direction.

To select parents among the neighbors and to decide to which parent a packet should be forwarded, each node is given a *rank*, which describes a cost of reaching the root from this node. Common rank metrics include hop count and estimated transmission count, ETX (i.e., $\frac{1}{PRR}$) [11] but others are possible [12]. To prevent cycles in the graph, the nodes' ranks should increase with the distance from the root. This can be achieved by computing a node's rank with an additive function over local metrics of links and/or nodes on the paths from the node. A popular function is sum of link ETX values [11], [13], [2]: a node sums for each parent the parent's rank and the ETX of the link to the parent; the minimal sum is the node's new rank and the parent corresponding to the sum is the node's *preferred parent*, to which packets are normally forwarded (cf. Fig. 1). More elaborate ranking functions, allowing, among others, for metric combining and flexibility in the selection of the preferred parent, include OF0 [14] and MRHOF [15].

Finally, whether and to which nodes a node advertises itself as destination depends on a particular routing technique: the more advertisers and the greater their scope, the less dilated the routing paths compared to optimal ones but also the more routing entries at each node. There exists an entire spectrum of techniques that trade off routing state for path dilation [16]. What is important here, however, is that they all require a DODAG for each destination. It is how a DODAG is built and maintained that affects failure handling.



A circle represents a node, with node A *being the root of the DODAG (the routing destination). A line corresponds to a wireless link between two nodes: an arrow points from a node to its preferred parent in the DODAG; a dashed line just connects two DODAG neighbors without indicating which of them, if any, is an alternative parent for the other. A number next to a link denotes the quality of this link in ETX. A (bold) number next to a node equals the node's rank in the DODAG.*

Fig. 1. A sample destination-oriented DAG (DODAG).

B. DODAG Maintenance

A DODAG can be formed on demand or maintained proactively. Let us focus on the proactive approach as it is more prevalent and has been standardized [2]. There are essentially three main methods for maintaining a DODAG.

Versioning: The first was adopted from wired networks and is based on advertisement versioning [17], [16]. In essence, the DODAG root periodically, every T_V time units, generates an advertisement with a new version number, which is disseminated among other nodes by flooding or gossiping and forces them to reselect parents and recompute their ranks. In other words, the DODAG is periodically rebuilt.

If the root fails, no new advertisement versions are generated. Therefore, when a node does not observe a new version for a sufficient period, it concludes that the root is down, empties its parent set, and adopts an infinite rank, thereby tearing down all its paths to the root.

Adaptation: The second DODAG maintenance method was introduced specifically for low-power wireless networks by the Collection Tree Protocol (CTP) [13]. It employs the Trickle algorithm [18] to minimize traffic when the DODAG is stable while ensuring fast adaptation to changes. More specifically, normally, once within every T_{max} time units (minutes or hours), each node broadcasts to its neighbors a so-called *beacon* containing its rank and preferred parent. The parent is selected based on received beacons, as the neighbor offering the lowest sum-ETX rank. However, whenever the node observes an event that affects the DODAG, such as a significant change of its rank, a routing cycle, or a so-called pulling beacon (i.e., a beacon from a neighbor lacking a parent), it resets its Trickle timer interval to T_{min} (milliseconds) to speed up the network's reaction to the event. From that moment, the node's subsequent intervals double up to T_{max}, unless another event is observed.

For detecting failures, this method relies solely on the underlying link quality estimator: when the quality of a link to a node's neighbor drops below a certain threshold, the estimator triggers DODAG maintenance, so that if the neighbor is the node's preferred parent, the node can change it for another one, potentially recomputing its rank. In contrast, no dedicated mechanisms are used for detecting failures of the root node.

Hybrid: The third method—the current state of the art—is standardized in RPL [2]. RPL is actually a framework with

many issues open to implementations. However, its core is a hybrid of Versioning and Adaptation. Like CTP, RPL employs beaconing but it separates pulling beacons (so-called DIS messages), broadcast with a fixed period, from normal beacons (DIO messages), broadcast following Trickle. DODAG version management is in turn left entirely to implementations; examples given in the standard include changing versions periodically, upon administrative decision, or on application-level detection of problems. Finally, the combination of Versioning and Adaptation also itself enables another mechanism that, as shown shortly, improves failure handling: bounding the maximal increase of a node's rank within a single DODAG version, so if the rank were to grow more due to major changes in the DODAG, the node would conclude that the DODAG is broken, empty its parent set, and adopt an infinite rank.

Apart from this, RPL does not define *any* mechanisms for detecting failures, leaving this entirely to implementations. The standard only suggests reactive approaches, for example, via IPv6 neighbor unreachability detection [19] or link-layer triggers [20]. These approaches originate from wired networks and focus on detecting failures of a single link. In contrast, for handling node failures, no special solutions are suggested.

In general, node failures have been studied mainly in the context of eliminating sensor outliers [21], [22], [23] or monitoring global network health [24], [25]. I am not aware of any work on explicitly detecting node crashes at the network layer. This seems consistent with a broader reflection by Birman [26] that node failure detection in distributed systems, in particular by employing network-layer observations, is largely unexplored. This paper thus constitutes a step to bridge the gap. Although due to space constraints it focuses on crashes of DODAG roots, generalizing the presented solutions to arbitrary DODAG nodes, for instance, those bridging densely connected parts of a network, is fairly straightforward.

III. Effects of DODAG Root Failures

To substantiate the claim that node failure detection in the existing DODAG maintenance methods leaves room for improvements, let us empirically study three representative routing protocols under crashes of DODAG roots. As an instance of the Versioning method, we take the hierarchical routing framework for TinyOS [16]. As an implementation of the Adaptation method, we adopt CTP from the main TinyOS repository [13]. Finally, as a representative of the Hybrid method, we use a fresh version of ContikiRPL [27].

A. Experimental Settings

To abstract out failure handling from a particular routing technique, we study the behavior of the protocols for a single DODAG spanning the entire network. Since the selected implementations differ in this respect, they have been adapted for consistency. More specifically, in the hierarchical routing framework, leader election and label assignment were disabled, and one node was designated to act as the sole root. CTP was left unmodified. In ContikiRPL, in turn, the RPL mode of operation 0 was activated, which implements pure distance-vector routing upward, without additional interfering DAO messages for enabling source routing downward.

To make the results reproducible, the protocols were deployed in simulators: TOSSIM and COOJA. Furthermore, unit-disk connectivity was used: a node had perfect links to the nodes in a unit radius and no links to other nodes. Finally, in all experiments 121 nodes were arranged in 11×11 grids with the inter-node spacing of $\frac{1}{\sqrt{2}}$ units, so that a node had up to 8 neighbors. The root node was in a corner.

In all experiments, the nodes were started randomly within one second and formed a DODAG. When the DODAG stabilized, the root node was killed and the node in the opposite corner was requested to initiate packet routing to the root.

B. Behavior of Versioning

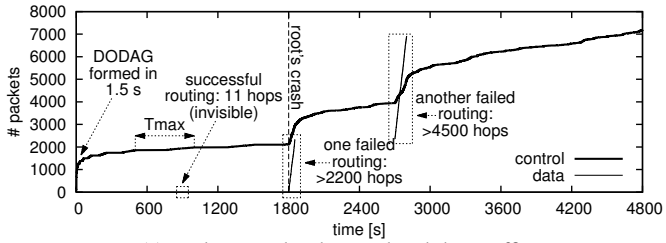
In Versioning, the root issues a new advertisement version every T_V time units (5 minutes in the implementation). Consequently, the minimal period a node has to wait without receiving a new version before it can conclude that the root has failed is $T_V+\varepsilon$, where ε accounts for delays of disseminating the advertisement in the network. In the implementation, $\varepsilon = 4\cdot T_V$ for flooding or $\varepsilon = (4+D)\cdot T_V$ for gossiping, where D is the hop-diameter of the network (the plots are omitted for brevity as they match these calculations). Even if ε were 0, $T_V = 5$ minutes to detect the failure would be a long time. Decreasing T_V would in turn proportionally increase control traffic—even without failures—which is highly undesirable.

This highlights another problem with Versioning: the time required by a node to detect the root's failure does not depend on the rate of data the node forwards. During the $T_V+\varepsilon$ period before concluding that the root is down, the node may be forwarding many data packets that will never reach the root but neither will they help the node detect that the root is down, thereby wasting network resources. The two remaining DODAG maintenance methods try to be more reactive.

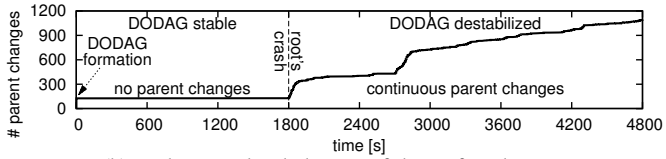
C. Behavior of Adaptation

Indeed, the behavior of Adaptation is different, as visible in a representative experimental run of CTP in Fig. 2. The nodes start with T_{min} Tickle timer intervals (125 ms in CTP), thereby generating lots of beacons. This allows them to rapidly select preferred parents, and hence form a DODAG. As the DODAG stabilizes, their intervals gradually double to up to T_{max} (500 s in CTP), so the control traffic levels off. A packet routed in the stable state needs 11 transmissions (hops) to reach the root.

In contrast, when the root crashes, CTP collapses. A routed packet reaches a root's neighbor, which repeatedly tries to forward it over the dead link to the root. The unacknowledged retransmissions ultimately cause the forwarding node to evict the failed link, select another preferred parent, and increase its rank accordingly. However, the parent may be one of the node's DODAG descendants, which may yet be unaware that the ancestor has lost its path to the root. As a result, a cycle appears in the DODAG. This is aggravated by the fact that the same eventually takes place at the root's all former neighbors:



(a) total accumulated control and data traffic



(b) total accumulated changes of the preferred parent

Fig. 2. A sample CTP run with a crash of the DODAG root.

since no link to the dead root exists, no parents the nodes select are able to provide any valid routing paths, only cycles.

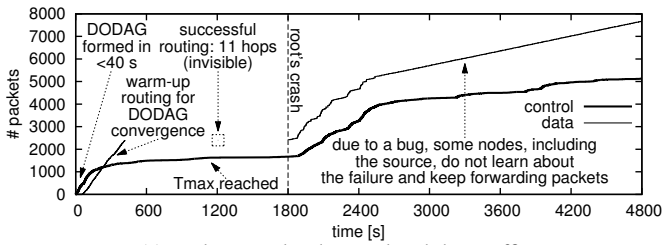
Significantly changing its rank, encountering a forwarding cycle, and receiving a pulling beacon from a neighbor with no parent, all cause a node to reset its Trickle timer to T_{min}. Since this often triggers a similar behavior at the node's DODAG descendants, the control traffic explodes whenever a packet is routed to the dead root, as can be seen in Fig. 2. Moreover, data packets are also forwarded excessively: in Fig. 2, one over 2200 and another over 4500 times. This is because CTP hardly ever drops a packet, essentially only if it is considered a link-layer duplicate or a node's forwarding queue overflows. In effect, entering a cycle, a packet circulates indefinitely: in the run from Fig. 2, the two packets were dropped only when the same node received them for the second time with the same value of a time-has-lived counter, which is incremented by 1 at each hop but loops back to 0 every 256 hops.

What is more, the DODAG may fail to stabilize at all, as in Fig. 2. The reason is that node rank inconsistencies after the root's failure—the cause of the cycles—remain even when no packets are routed. When exchanging beacons, the nodes thus continuously discover these inconsistencies, switch parents, increase their ranks, and reset their Trickle timers, which is particularly visible in the bottom plot of Fig. 2. To sum up, the Adaptation method does not handle a DODAG root's crashes.

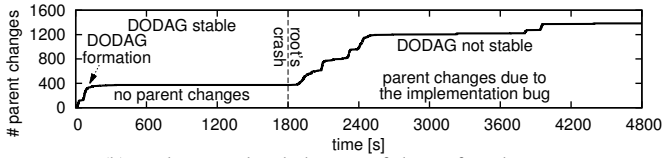
D. Behavior of Hybrid

The Hybrid method alleviates the problem of cycles. It limits the maximal growth of a node's rank within a DODAG version. In effect, some time after the root's crash, a node has no candidate that can be made the node's preferred parent without violating the node's rank growth limit. Consequently, the node empties its parent set and adopts an infinite rank, thereby tearing down its paths to the dead root. Note that using the same approach in Adaptation would prevent the algorithm from repairing the DODAG after major connectivity changes, if they required some nodes to increase their ranks beyond the limit. In contrast, in Hybrid, the DODAG can always be rebuilt by forcing the root to generate a new version. ContikiRPL should thus handle failures of the root better than CTP.

To verify this, however, the previous experimental scenario had to be modified. First, since ContikiRPL requires actually routing some packets for a freshly formed DODAG to converge, each node generated warm-up packets from the moment it first selected its parent to the moment its link estimates stabilized. Second, as RPL mandates dropping packets upon routing loop detection and allows for doing this in other situations, instead of a single packet, the sender node generated packets continuously in 10-s intervals after the root's crash; otherwise, it would have been impossible to detect the failure. Such a sample experimental run is shown in Fig. 3.



(a) total accumulated control and data traffic



(b) total accumulated changes of the preferred parent

Fig. 3. A sample ContikiRPL run with a crash of the DODAG root.

As expected, before the root's failure, ContikiRPL behaves largely the same as CTP. The small differences in the convergence speed and control traffic can be attributed mainly to different settings of Trickle: $T_{min} = 4$ s and $T_{max} \approx 20$ min.

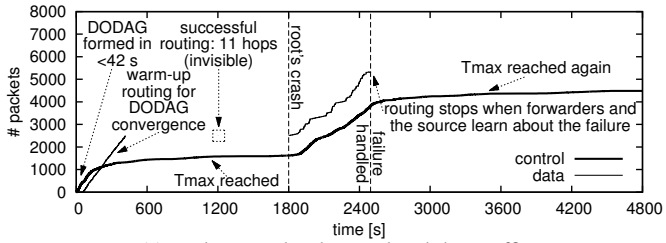
What is in turn unexpected is the behavior after the root's crash. The source node and some of its DODAG ancestors do not detect the failure. In effect, before being dropped, packets generated every 10 s are forwarded over a few hops, which is the reason for the continuous growth in data traffic in Fig. 3.

This behavior can be tracked to a subtle bug in the implementation of Hybrid in ContikiRPL. Under some circumstances, a node with an infinite rank—one that has torn down all its paths to the root—"forgets" the minimal rank it has advertised in the current DODAG version. It then chooses a parent and adopts a rank that causes a violation of its rank growth limit (cf. the occasional node parent changes in Fig. 3). As a result, cycles reappear the in the DODAG.

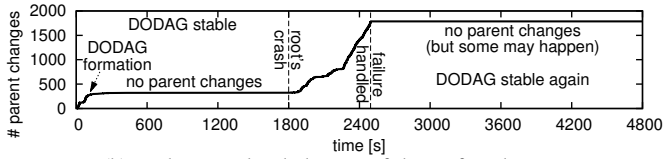
The bug was introduced in an attempt to fix another issue with failure detection. Without that code (results not plotted), the rank growth limit is respected, but information on the root's failure does not propagate at all in the DODAG.

The actual fix, developed during the presented research to accurately reflect Hybrid in ContikiRPL, is more intricate and involves changes in multiple places in the protocol code. A sample run of ContikiRPL with the fix is presented in Fig. 4.

In the correctly implemented Hybrid method, the root's death is eventually handled: after 2495 s in the run from Fig. 4, the routing layer at the source node stopped forwarding packets and started returning an error to the application layer.



(a) total accumulated control and data traffic



(b) total accumulated changes of the preferred parent

Fig. 4. A sample run, involving a crash of the DODAG root, of a version of ContikiRPL patched to correctly implement the Hybrid method.

However, the process is relatively slow and traffic-intensive. In the run from Fig. 4, the source node did not tear down its paths to the root until it had issued 70 packets. With the inter-packet interval as short as 10 s, and hence unlikely in low-power deployments, this still translates to 12 minutes, which is a lot compared to just 42 s for DODAG formation. What is more, those packets were forwarded 2826 times in total, which corresponds to a lot of data traffic. Finally, in the process, the nodes frequently switched parents and reset their Trickle timers, which in turn generated considerable control traffic.

Nevertheless, since Hybrid—if implemented correctly—does handle DODAG root failures and is the most advanced of the three DODAG maintenance methods, the following sections focus on improving its performance under such failures.

IV. The RNFD Algorithm

One approach that may potentially yield such improvements is to concentrate on individual issues that affect the handling of DODAG root crashes. Although some issues, such as node density and connectivity or the number of packet sources and their data generation rates, are beyond the control of a routing protocol, trade-offs in others can be exploited. For example, an estimator that aggressively marks links as dead can speed failure detection up, but will also cause false positives during normal operation, thereby increasing control traffic. Likewise, a rank growth limit smaller than the default can make detecting a root's crash faster, but will also limit the capabilities of a DODAG to reorganize after significant network changes. Above all, a general drawback of this trade-off exploration approach is that different deployments may require different configurations. Therefore, while studying such trade-offs is an interesting avenue for future work, here we attempt to identify and address the fundamental reasons for the low performance of DODAG root failure handling.

A. Observations

To this end, let us observe that detecting a DODAG root's crash depends largely on the root's neighbors. While forwarding packets, they gradually degrade the quality estimates of their links to the root, eventually evicting the links from the DODAG. Only then does information on the failure propagate down the DODAG, when subsequent nodes start lacking parent candidates that do not violate their rank growth limits, and hence adopt infinite ranks and broadcast beacons.

For a given data source, however, *the link removal process is sequential*. A node having the root as its preferred parent attempts to forward the source's packets—one after another—over the link to the root. Only when enough packets have failed to be forwarded over that link, does the link estimate degrade so much that the node evicts the link as dead and switches its preferred parent. At this point, another node takes over forwarding subsequent packets from the source on the last hop to the root, thereby gradually degrading the estimate for its link to the root, and so on. In other words, the dead links are degraded and removed one after another with a pace depending on the traffic from the source. Such sequential link removal is inherently slow. Although with multiple data sources, it may seem more parallel, traffic in low-power wireless networks is normally expected to be low; otherwise, a different networking technology would be more suitable. Therefore, the actual amount of parallelism still makes it hard to handle the root's failures with latencies as low as those for DODAG formation.

Furthermore, only when sufficiently many links to the root have been removed globally from the DODAG, can a node adopt an infinite rank. This is because if any of the root's former neighbors still "believes" to have a link to the dead root, it advertises a path over that nonexistent link in its beacons (i.e., it advertises a finite rank). Consequently, the node can select the neighbor or one of its DODAG descendants as the preferred parent, provided that doing so, it will not violate its rank growth limit. In practice, this often means that *all dead links to the root have to be removed from the DODAG* before a node has no candidate parents to choose from, and hence sets its rank to infinite. There may be many such links, and thus lots of data packets may need to be forwarded before those links are all degraded and removed from the DODAG.

Finally, *the link removal process is uncoordinated*. Each neighbor of the dead root degrades the quality estimate for its link to the root on its own, depending only on the traffic it forwards. In addition, while degrading and evicting links, the neighbors (and other nodes) switch parents, change ranks, and reset their Trickle timers. This generates lots of beacons, some of which could potentially be avoided if the link removal process were somewhat coordinated between the nodes.

B. Principal Ideas

Considering these observations, the proposed new approach to node failure handling is based upon the following ideas:

Coordinate failure detection: Whereas a dead link can be detected by a single node, a crash of a node is detected by multiple other nodes. It is thus reasonable to coordinate their activities: nodes suspecting that a DODAG root may be down should collaborate in verifying whether this is the case.

Remove links in parallel: To speed up failure detection, multiple links to a suspected node should be removed in

parallel. This can be achieved by concurrently probing links to a suspected root by the root's neighbors, even when there is temporarily no data traffic through some of these neighbors.

Do not probe all links: Due to the coordination and link probing, nodes no longer have to remove from the DODAG every link to a suspected node. Instead, after probing just sufficiently many links to a suspected DODAG root, they may consent that the root is down with a high probability.

What is also important is that rather than replacing existing solutions, this approach assumes complementing them. The reason is twofold. First, since coordination entails overheads, a user should have a possibility to deploy the presented solutions for only selected DODAGs, for which the overheads are justified, for example, DODAGs rooted at border routers, data sinks, or crucial actuators. Second, the Hybrid method is standardized and guarantees that a DODAG root's failure is eventually handled. It is thus reasonable not to abandon it for the sake of this guarantee and compatibility with the standards.

Finally, in line with the previous argument, the goal is to propose not a monolithic solution for handling DODAG root failures but an *open framework* algorithm that, on the one hand, organizes failure handling and, on the other hand, enables replacing its various mechanisms. Among others, this should facilitate standardization and allow for benefiting from future research developments, notably the aforementioned trade-off exploration. I have dubbed the algorithm *RNFD*, an acronym for *Routing-layer Node Failure Detector*.

C. Algorithm Overview

To implement the three core ideas, RNFD requires each node to explicitly track the current condition of the DODAG root, referred to as the node's *observed root state*, and to synchronize this knowledge with other nodes. The possible observed root states and their transitions are depicted in Fig. 5.

Initially, a node is not aware of the DODAG, and hence maintains no information on it, which is represented in Fig. 5 by an artificial state, *NONE*. If the node joins the DODAG, the root must be alive, so the node changes its observed root state to *UP* (transition 1). For a properly working root, the node's observed root state remains *UP*. In contrast, when the node suspects that the root may have failed, it changes the state to *SUSPECTED DOWN*. The transition from *UP* to *SUSPECTED DOWN* can happen based on the node's observations from either the data plane (transition 2a), for instance, missing link-layer acknowledgments for packets forwarded over the node's link to the root, or the control plane (transition 2b), for example, a significant growth in the number of nodes suspecting the root. In the *SUSPECTED DOWN* state, the node can verify its suspicion and/or inform other nodes about the suspicion. After this has been done, it changes its observed root state to *LOCALLY DOWN* (transitions 3a and 3b). If sufficiently many nodes are in this state, all nodes consent globally that the root must be down and set their observed root states to *GLOBALLY DOWN* (transition 4).

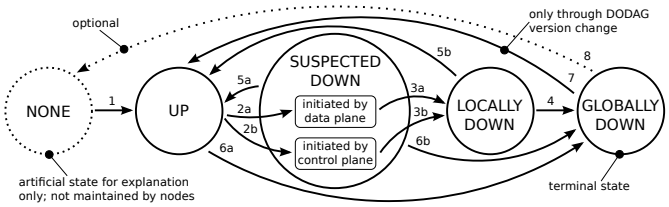


Fig. 5. The possible observed root states and transitions between them.

In state *UP*, a node has a parent and can thus forward packets. In states *SUSPECTED DOWN* and *LOCALLY DOWN*, it forwards packets only if it has a parent. While in any of the two states, it may observe that its suspicion is a mistake, in which case it returns to *UP* (transitions 5a and 5b). Likewise, in states *UP* or *SUSPECTED DOWN*, it may learn that other nodes have already agreed globally that the root is down, in which case it transits directly to *GLOBALLY DOWN* (transitions 6a and 6b).

The *GLOBALLY DOWN* state, in contrast, represents a situation where the nodes have reached consensus that the root is down. Therefore, no node in this state forwards any packets to the root. Furthermore, to ensure that the consensus outcome is indeed global, this state is terminal for the DODAG: the only transition from *GLOBALLY DOWN* to another state (transition 7 to *UP*) takes place when a new DODAG version is observed. Although Fig. 5 contains also a transition from *GLOBALLY DOWN* to *NONE* (transition 8), which corresponds to garbage-collecting information on a dead DODAG, this transition requires additional guarantees and in many cases is not required, as explained in Sect. IV-G.

D. Counting Neighbors of the Root

Several transitions in this scheme require each node to keep two counters: the total number of the root's neighbors and the number of the root's neighbors that consider the root as *LOCALLY DOWN*. The maintenance of these counters must be decoupled from the DODAG's shape, because when they are used, the DODAG may be broken. They are thus maintained through normal beaconing: each node embeds its counters in the beacons it broadcasts; when it receives a beacon from its neighbor, it merges the neighbor's counters with its local ones.

To avoid counting the same node many times—which would happen due to repeated merges if the counters were plain integers—the counters are implemented with order- and duplicate-insensitive synopses [28]. A synopsis is a probabilistic structure that uses little state, yet estimates the cardinality of a finite set with a small error. It can be set to zero or to a maximal value, which corresponds to respectively emptying or filling up the set, or incremented by one, which corresponds to adding an element to the set. Importantly, the result of merging two synopses is as if the two underlying sets were merged, as such being order- and duplicate-insensitive.

An example is linear counting synopsis [29]. It is a table of N bits of which only M bits (e.g., $0 \dots M-1$) must be stored physically. Resetting to zero/maximal value corresponds to clearing/setting all bits. Incrementing is done by setting a

uniformly random bit. Merging is OR-ing the two bit tables. Finally, the cardinality is estimated as $-N \cdot ln\frac{M_0}{M}$, where M_0 is the number of clear bits among the M stored bits (or 1 if all bits are set). Another synopsis example is hyper-log-log [30]. In general, different synopses offer different accuracy-state trade-offs. Therefore, RNFD does not require a particular synopsis, but allows for choosing the most suitable for a deployment.

Since a synopsis is an additive structure whereas our counters require also subtraction, each counter is implemented as two synopses. The counter value for the total number of the root's neighbors is a difference between the estimates of synopses s_A and s_R, which represent the number of nodes that, respectively, added the root to and removed it from their neighbor sets. Counter incrementation is done by incrementing s_A, decrementation by incrementing s_R. Similarly, the number of the root's neighbors that consider it as locally down is a difference between synopses: s_D, representing the root's neighbors that decided that the root is down, and s_M, representing the root's neighbors whose decisions were mistakes.

The four synopses are reset or incremented locally by a node based on its observations, as explained in subsequent sections. They are synchronized with other nodes through beaconing, as mentioned previously. Therefore, ideally, whenever a synopsis changes at a node, we would like the change to be propagated fast to other nodes. This need not be the case, though, as beacon broadcasting in Hybrid is governed by a Trickle timer, which is reset to the minimal interval, T_{min}, only upon major changes in the DODAG—not upon changes to the synopses.

As a remedy, RNFD introduces a dedicated Trickle timer for the synopses. This timer ticks independently of the main Trickle timer, and a beacon is broadcast whenever any of the two timers fires unless it has already been broadcast since the timer fired for the last time (due to the other's timer firing). An advantage of having two timers over simply resetting the main timer upon a synopsis change is that propagating synopsis changes may require different settings for Trickle or lower-level services, such as narrowcasting [31], than propagating DODAG information. At the same time, suppressing redundant beacons limits the growth in control traffic that would occur with two completely independent timers.

E. Detecting Problems with the Root

When joining a DODAG, a node sets its observed root state to *UP* and initializes all its synopses, s_A, s_R, s_D, and s_M, to zero. If it is not the DODAG root's neighbor, its role in the algorithm is passive: it just observes received beacons and acts only when the others have agreed that the root is down (see Sect. IV-G). If, in contrast, it is the root's neighbor (or ever becomes such a neighbor), it increases its s_A, effectively increasing the total count of the root's neighbors, optionally resets its synopsis Trickle timer to speed up propagation of the change, and starts taking an active part in the algorithm.

More specifically, at the data plane, it observes packet forwarding attempts over its link to the root. When such attempts fail (e.g., no link-layer acknowledgments are received), it may start suspecting the root to be down. RNFD does not rely on any particular failure detection heuristic. In the experiments, however, the following two were used. In the first, referred to as *NoAck-Oracle*, if a forwarding attempt fails, the node consults an oracle that has a perfect knowledge on the root's condition. The goal of this heuristic is to demonstrate the potential of RNFD with a perfect failure detector. In the second, dubbed *NoAck-K*, the node starts suspecting the root if K consecutive transmissions to the root fail. The heuristic thus allows for assessing RNFD with a common failure detector.

At the control plane, in turn, the node observes its synopses and starts suspecting the root is down whenever the fraction of neighbors that consider the root to be down, $\frac{s_D - s_M}{s_A - s_R}$, has grown significantly, at least by Δ_S, since the last time the node suspected the root. This mechanism aims to speed up failure detection when the node does not forward any packets, and hence is unable to detect anything at the data plane. The suspicion threshold, $\Delta_S \in (0 \ldots 1]$, controls how aggressive the node's suspicions are: the lower Δ_S, the more easily the node suspects the root; in contrast, $\Delta_S = 1$ disables the mechanism.

Finally, as RNFD runs along a regular routing protocol, the protocol may itself add/remove the root to/from the node's neighbor set. RNFD reacts to such events as follows. Whenever the root becomes the node's neighbor, the node increases its s_A, optionally resets its synopsis Trickle timer, and starts taking an active part in the algorithm, like upon joining the DODAG. Whenever, in turn, the root ceases to be the node's neighbor, which is allowed only if the node does not already consider it as down, the node increases its s_R, resets its synopsis timer, and changes its role in RNFD to passive.

F. Verifying if the Root Has Failed

When a node being the root's neighbor starts suspecting that the root may be down, it changes its observed root state to *SUSPECTED DOWN*. At this point, it can verify its suspicion before advancing the state further, to *LOCALLY DOWN*. Again, RNFD does not mandate any particular verification heuristic: users can define their own ones. In the experiments, however, the following heuristics were employed.

If the suspicion is raised based on the node's observations at the data plane, no verification is performed, and the node transits directly to *LOCALLY DOWN*. The rationale is that if multiple forwarding attempts from the node to the root have failed, it is unlikely that they will succeed during verification.

If, in contrast, the suspicion is raised based on observations at the control plane, the node performs additional verification at the data plane with probability P_V. The verification boils down to forwarding packets to the root and relying on the data-plane heuristics to assess whether the root is down. If the node has no data packets to forward, it generates synthetic ping packets. Since many of the root's neighbors may perform the verification simultaneously, in state *SUSPECTED DOWN*, forwarding a packet is always preceded by a random backoff proportional to the number of the root's neighbors: $s_A - s_R$. Probabilistically deciding whether to perform the verification, which is controlled by the P_V parameter, aims in turn to avoid verifying all links to the root in the DODAG.

If a node being the root's neighbor has decided—with or without verification—that from its perspective the root seems down, it transits to *LOCALLY DOWN*, as mentioned previously. In addition, however, it performs two actions that aim to stimulate other nodes to also assess the root's condition.

First, if the node has any data packets to forward, it forwards the first one, albeit not to the root but to k_F of its other parent candidates, in addition tagging the packet with a special flag. A node that is *not* the root's neighbor and that receives such a packet simply forwards it to its preferred parent. In contrast, when the root's neighbor receives such a packet, it changes its observed root state to *SUSPECTED DOWN* if the state is still *UP*. It then performs normal data-plane verification of the suspicion with the packet. Assuming that $k_F \geq 2$, this fan-out forwarding mechanism allows for disseminating the suspicion among the root's neighbors exponentially fast. This speeds up failure detection, provided that the network is not overloaded with tagged packets. To ensure this, each node limits the number of tagged packets it may forward in a time period, T_F, to c_F: tagged packets above this limit are dropped.

The second action that a node being the root's neighbor performs upon transiting to *LOCALLY DOWN* targets the control plane. More specifically, the node increases its local synopsis of the root's neighbors that consider the root as down, s_D, and resets its synopsis Trickle timer to T_{min}. In effect, other nodes are likely to quickly learn that yet another node believes the root to be down, which may trigger them to start suspecting the root themselves, as discussed in Sect. IV-E, or even agree that the root is globally down, as explained next. This control-plane mechanism aims to handle situations where the condition of the DODAG prevents fan-out forwarding from reaching all of the root's neighbors, for example, when the DODAG is broken: the synopses are embedded in beacons, and hence propagate independently of the DODAG's shape.

G. Reaching Consensus on a Failure

In state *SUSPECTED DOWN* or *LOCALLY DOWN*, a node may realize that its suspicion is a mistake. Also here, RNFD does not rely on any specific heuristic based on which this is done. In particular, in the experiments, a node considered its suspicion as false upon any sign that the root was up: a beacon or a link-layer acknowledgment received from the root.

Realizing that the root is up, the node sets its observed root state back to *UP*. In addition, if the state was *LOCALLY DOWN*, the node must have incremented s_D and may have notified others about its suspicion by broadcasting the increased s_D in a beacon. To inform about its mistake, it thus increments s_M and optionally resets its synopsis Trickle timer. In effect, it is no longer counted in $s_D - s_M$ (i.e., as suspecting the root).

If, however, the root is indeed down, more and more of its neighbors change their observed root states to *LOCALLY DOWN*, thereby also globally increasing s_D. When *any* node—the root's neighbor or not—observes that the fraction of the root's neighbors that consider the root as down, $\frac{s_D - s_M}{s_A - s_R}$, is above a threshold, θ_C, it changes its observed root state to *GLOBALLY DOWN*, sets s_D and s_A to maximal values, and resets its synopsis Trickle timer; the same is done if a node learns that any synopsis has a maximal value. As a result, eventually all nodes set their observed root states to *GLOBALLY DOWN*, thereby agreeing that the root has failed.

The correctness of this consensus algorithm follows from two observations. First, a node transiting to *GLOBALLY DOWN* sets its synopses to maximal values, so that after its next beacon, its neighbors, their neighbors, and so on, also set their synopses to the maximal values and thus transit to *GLOBALLY DOWN*. Second, no node leaves the terminal *GLOBALLY DOWN* state unless the root generates a new DODAG version, which in turn implies that the root is up.

Finally, RNFD optionally allows for garbage-collecting information on a dead DODAG that is unlikely to recover (transition 8 in Fig. 5 from *GLOBALLY DOWN* to *NONE*). However, implementing this transition requires additional guarantees, in particular, when any node does the transition, all nodes must be in state *GLOBALLY DOWN*; otherwise, information on the dead DODAG may reappear in the network. In practice, no garbage collection may be necessary, like in CTP and to some extent in ContikiRPL. Alternatively, probabilistic guarantees may be acceptable, such as having each node wait sufficiently long in *GLOBALLY DOWN* before transiting to *NONE*.

V. Experimental Setup

RNFD has been implemented in TinyOS 2.1 and evaluated in the TinyOS simulator, TOSSIM, and on two testbeds.

A. Implementation

The initial approach was to integrate RNFD into ContikiRPL. However, in the process, it turned out that the produced binaries exceeded the MCU program memory of the MSP430-based nodes I had access to: the code overhead of IPv6 is simply large. As the same was likely for TinyRPL, the integration of RNFD with RPL was postponed.

Instead, a lightweight prototype implementation was created as follows. The TinyOS version of CTP was modified, so that instead of Adaptation, it implemented Hybrid DODAG maintenance, as in RPL. This included adapting the routing and forwarding engines of CTP but the original configuration of timings, pool sizes, thresholds, and the like was preserved to facilitate comparisons. Extensive tests were then conducted to ensure that this implementation of Hybrid in CTP indeed worked, notably that it handled root crashes and recoveries in various scenarios. Only upon this Hybrid CTP, was RNFD implemented, using for the open issues the solutions proposed in the previous section. Table I lists the memory footprints of the evaluation application for the two implementations (plus the original CTP) on the popular TelosB platform.

TABLE I
The memory footprint of the experimental binaries for TelosB

Memory type	Original CTP	Sole Hybrid CTP	Hybrid+RNFD CTP
ROM	21,746	22,578 B	31,102 B
RAM	3,710	3,714 B	3,898 B

Although RNFD incurs overhead on the program memory (ROM), the overhead is not huge, in particular, as the implementation needs no code for handling ICMPv6 packets, dealing instead with much simpler Active Messages. Furthermore, compared to Table I, the actual RAM overhead is smaller. More specifically, what RNFD requires is just the state maintained at each node. In the implementation, it is four 64-bit linear counting synopses, a synopsis Trickle timer, and several integers and booleans. The rest of the RAM overhead in the table is in turn mostly due a dedicated buffer for a ping packet for verifying suspicions in the absence of data traffic (cf. Sect. IV-F). The buffer is used for simplicity and consistency with CTP. Normally, however, dynamic packet pooling would be utilized instead. Finally, the additional information in packets (not listed in Table I) comprises one bit in each forwarded packet for the fan-out forwarding flag (cf. Sect. IV-F) and the four synopses in each beacon.

B. Experimental Settings

RNFD exposes six configuration parameters and leaves several issues open to implementations. This approach is similar to how most routing protocols provide user-configurable parameters that can tune their performance. In addition, the parameters control RNFD's behavior in specific or pathological topologies, under rare network events, or in situations resulting from implementation specific decisions and heuristics. For this reason, however, rather than exhaustively studying all parameter configurations across all such scenarios, this preliminary evaluation involves just a few parameter values and scenarios that are adequate to demonstrate RNFD's potential.

Therefore, unless noted otherwise, the following parameter values were used. The synopsis growth threshold that triggered a suspicion at the control plane, Δ_S, was $\frac{1}{8}$ (cf. Sect. IV-E), while the probability, P_V, of performing additional data-plane verification upon such an event was 1 (cf. Sect. IV-F). The number of parents to which a tagged packet was transmitted during fan-out forwarding, k_F, was 10, but in a period, T_F = 10 s, a node was allowed to forward at most $c_F = 4$ such packets (cf. Sect. IV-F). Finally, the consensus threshold for the synopses, θ_C, was set conservatively to $\frac{3}{4}$ (cf. Sect. IV-G).

The experimental application itself aimed to model a common traffic pattern and give as much advantage as possible to sole Hybrid. More specifically, at a random moment in every T time units, each node generated a packet that was forwarded by the network to the root. This resulted in relatively uniform all-to-one traffic that helps Hybrid to remove all dead links after the root's failure, thereby speeding up failure detection. Other tested traffic patterns showed wider performance gaps between RNFD and sole Hybrid, so they are omitted here. Furthermore, unless noted otherwise, no radio duty-cycling was used, so that low values of T could be tested without the results being influenced by a particular energy saving technique.

The first evaluation environment for the application was a low-level simulator with realistic communication models, TOSSIM. Its accompanying tools were used to generate network topologies with 121 nodes placed randomly in square areas of 5 various sizes and realistic radio gain values between the nodes (10 different network instances per area size). In addition, for simulating noise, a real-world trace (Casino Lab) was utilized. The noise was relatively heavy as it reduced the mean neighborhood size up to a factor of 4 compared to the same environment without it. A node's neighborhood size was measured in preliminary experiments as the sum of PRR to all other nodes, which implies that it overestimated the number of links actually suitable for routing. All in all, the simulations should predict the real-world behavior of RNFD well.

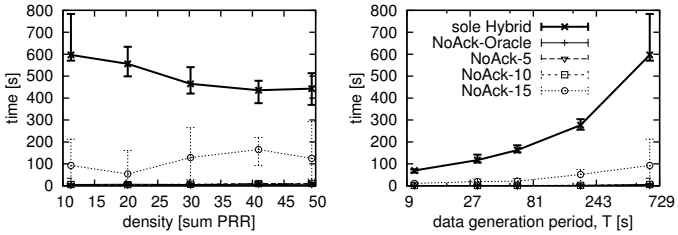
The experiments on the two testbeds aimed to verify this. Testbed A consisted of 32 TelosB nodes communicating in the 2.4GHz band and deployed in a student computer lab. The network diameter was 2 hops. The neighborhood sizes ranged from 7 to 31. There were some asymmetric links and noise from students' WiFi devices. Testbed B, in turn, comprised over 100 G-Node G301 nodes communicating in the 868MHz band and dispersed across an office building [32], of which at least 76 were active during the experiments. The network diameter was 6 hops and the density varied from a few to tens of neighbors per node, with many asymmetric links. Overall, communication between the testbed nodes exhibited many phenomena normally encountered in real-world deployments.

VI. TOSSIM Results

Numerous simulation runs have been conducted but, due to space constraints, just their fraction is presented here. In these runs, nodes started simultaneously and operated for 5 TOSSIM hours, apart from the root, which was killed after 2.5 hours. The nodes thus operated long enough in the stable states, both before and after the root's death, to study their performance.

A. Duration of Failure Detection

Figure 6 shows the time it takes 90% of nodes to handle a root's death, that is, to empty their parent sets and permanently adopt infinite ranks, depending on the network density (a) and packet generation interval (b). The 90% is chosen instead 100% because the duration of failure detection in sole Hybrid has a long tail: a few nodes often lag behind. In contrast, in RNFD a consensus outcome propagates rapidly. The choice of 90% thus aims to avoid bias. Each data point represents a median over the 10 instances of a network configuration (area size). The whiskers, in turn, mark the 10^{th} and 90^{th} percentiles.



(a) density = $\star$; T = 600 s (10 min.) (b) density = sparsest; $T = \star$

Fig. 6. The duration of failure handling by 90% of the nodes.

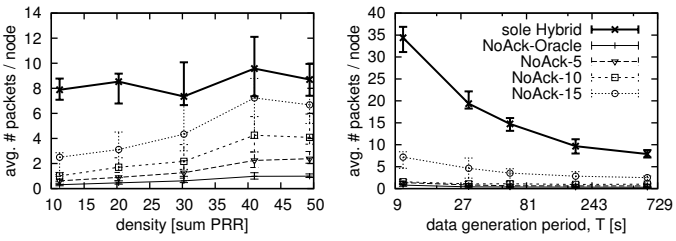
As can be observed, RNFD outperforms sole Hybrid. For example, in the sparsest network with each node generating a packet in every $T = 10$ minutes, the median failure handling time for sole Hybrid is 597 s whereas for *NoAck-Oracle*, that is, RNFD with a perfect failure detector, it is just 6 s: two orders of magnitude less. RNFD with a popular, practical failure detector, *NoAck-K*, is also much faster. In particular, even for the worst $K = 15$, the corresponding time is 93 s, while for the remaining two, less than 10 s. The growth in density reduces this difference (Fig. 6(a)), albeit only slightly, which is due to a smaller DODAG diameter. In contrast, decreasing the packet generation interval (i.e., increasing the data generation rate), has a more pronounced effect (Fig. 6(b)). Yet, even for the shortest interval of 10 s, arguably too short for low-power applications, RNFD is nearly an order of magnitude faster. This is also true with radio duty-cycling. For example, in the aforementioned settings but with 128-ms-wakeup-interval low-power listening [33], the median failure handling time for sole Hybrid and *NoAck-10* is respectively 689 s and 34 s.

Since RNFD operates along Hybrid, they cannot be slower than sole Hybrid. On the contrary, due to the design ideas behind RNFD, properly configured together, they can be much faster. An example of such a configuration option illustrated in Fig. 6 is K in the *NoAck-K* failure detector. With $K = 5$, just 5 missing link-layer acknowledgments from the root raise a node's suspicion, so RNFD is triggered fast. The higher K is, the longer the trigger latency, and hence the slower failure handling. Interestingly, however, $K = 15$ performs visibly slower than $K = 5$ or 10. This is because above 10 unacknowledged transmissions a node's underlying link estimator [5] degrades link quality estimates aggressively. Consequently, Hybrid at a node being the root's former neighbor may switch the node's preferred parent. In effect, for a while the node no longer forwards anything over its link to the root, which delays triggering RNFD. This is just one example of behaviors that may emerge depending on the solutions a particular routing protocol employs, notably for the issues RNFD leaves open to implementations. Nevertheless, in reasonable configurations, RNFD can indeed boost failure handling.

B. Traffic during Failure Detection

Figure 7 gives more insight into these results. It depicts the number of data packets a node forwards on average from the moment the root crashes or, put another way, the data traffic of an average node to handle the root's failure, because afterward the nodes do not forward any data packets.

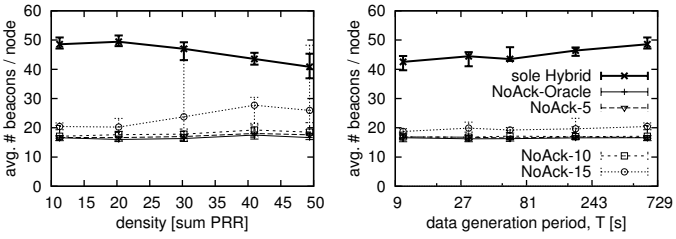
It can be seen that the difference between RNFD and sole Hybrid is lower than for the failure handling time. This is due to the fan-out forwarding employed by RNFD to quickly verify links to a suspected root, even for low data traffic (cf. Sect. IV-F), which is particularly visible for a growing network density (Fig. 7(a)), and hence the number of such links. The figure also highlights the differences between *NoAck-K* for various K. All in all, however, the total data traffic is also lower for RNFD than for sole Hybrid, which implies that RNFD utilizes the traffic more efficiently to detect the root's failure.



(a) density = ⋆; T = 600 s (10 min.) (b) density = sparsest; T = ⋆

Fig. 7. The average node data traffic during failure detection.

The same is true for control traffic, as can be seen in Fig. 8. The figure shows an average node's control traffic in the first 0.5 hour after the root's failure. This fixed period is chosen instead of the (variable) total failure handling time again not to favor RNFD, which handles the root's crash rapidly. The 0.5 hour is sufficient even for the nodes running sole Hybrid to handle the failure (cf. Fig. 6) and later gradually double their Trickle timers up to T_{max} (500 s), thereby bringing the network back to a stable state, albeit without the dead root.



(a) density = ⋆; T = 600 s (10 min.) (b) density = sparsest; T = ⋆

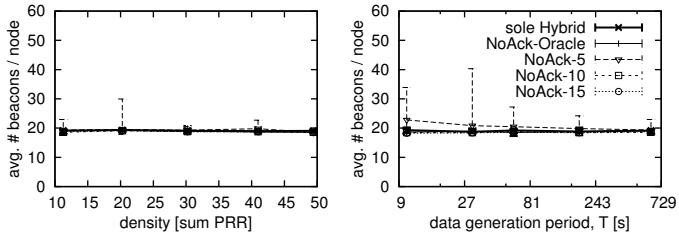
Fig. 8. The average node control traffic for 0.5 hour after the root's crash.

Figure 8 shows that, in most of the presented configurations, RNFD generates a factor of 2–3 less traffic than sole Hybrid. In fact, the traffic in these configurations is close to the optimal 16 beacons per node, resulting from a single reset of a node's Trickle timer to T_{min}, doubling the timer 12 times to T_{max}, and 4 triggers of the timer already in T_{max}. The only exception is again *NoAck-15*, for which the traffic in the 90^{th} percentile run is high in the densest and medium-density configurations. The reason is the aforementioned premature parent switching that not only delays triggering RNFD but also generates additional beacons. Such parent switching is a direct consequence of the way Hybrid handles failures, and hence precisely the reason for the heavy control traffic in sole Hybrid. In short, RNFD normally outperforms sole Hybrid also in control traffic.

C. Behavior in the Absence of Failures

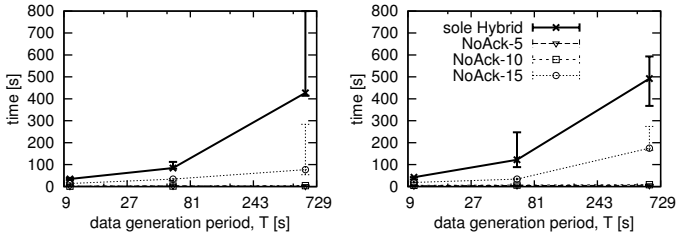
Figure 9 shows analogous plots but for the first 0.5 hour after the start. It thus compares control traffic of RNFD and sole Hybrid during DODAG formation and in the stable state.

It can be seen that RNFD performs largely the same as sole Hybrid. In both solutions, the number of beacons per node is slightly higher than that for RNFD in Fig. 8 because of the initial pulling beacons the nodes broadcast when they are unaware of any DODAG. The only worse RNFD configuration



(a) density = ⋆; T = 600 s (10 min.) (b) density = sparsest; T = ⋆

Fig. 9. The average node control traffic for 0.5 hour after the start.



(a) Testbed A (32 nodes @ 2.4GHz) (b) Testbed B (76 nodes @ 868MHz)

Fig. 10. The duration of failure detection by 90% of the nodes (cf. Fig. 6(b)).

this time is *NoAck-5*, which is particularly visible for shorter packet generation intervals (Fig. 9(b)). The occasional control traffic surge in *NoAck-5* happens mainly when the DODAG is not yet stable, node Trickle intervals are short, but the nodes already route packets. Since only 5 missing link-layer acknowledgments from the root raise a node's suspicion and since the network is relatively congested, it may happen that such a suspicion indeed occurs. This in turn causes a change to the nodes' synopses followed by a reset of their Trickle timers, which results in more beacons being broadcast. For the higher values of K in *NoAck-K*, this effect is not observable in Fig. 9.

By and large, properly configured RNFD hardly affects the operation of Hybrid in the absence of failures. In particular, in the presented simulations, there is no statistically-relevant difference between RNFD and sole Hybrid in terms of packet delivery rates (in all cases, above 99%) or the total transmissions necessary to deliver packets. Therefore, plots for these results are omitted. Likewise, by design, RNFD does not influence the DODAG's shape. Normally thus the main stable-state overhead to consider when deploying RNFD is only the aforementioned extra information in packets: one bit in every forwarded packet and four synopses in beacons.

VII. Testbed Results

A number of experiments have also been conducted on the two testbeds. Most of them were microbenchmarks according to the same scenario as the simulations. In addition, however, a few long-term experiments were conducted. In both cases, the main problem was that the simulated hours for the algorithms to stabilize and operate in either of the two stable states translated to real testbed time. The number of configurations and experimental runs per configuration thus had to be limited. Nevertheless, the results do serve our main goal for the testbed evaluation: assessing the potential of RNFD in the real world.

A. Microbenchmark Experiments

As an illustration, Fig. 10 shows the duration of failure detection for 90% of nodes generating packets at various rates. Each point represents a median over 3 microbenchmark runs. The whiskers, in turn, mark the minimum and maximum run.

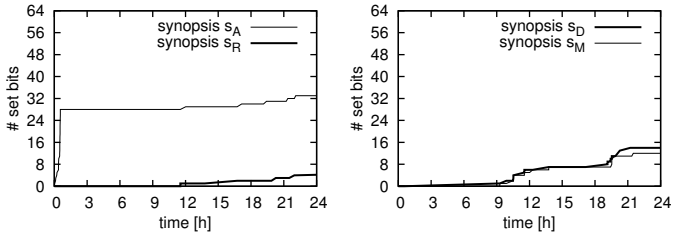
The presented values match the simulations well. For instance, on both testbeds, with the packet generation interval of 600 s, *NoAck-10* again detects the root's crash in under 10 s, whereas the median time for sole Hybrid is 427 s on Testbed A and 492 s on Testbed B. Likewise, with respect to the other analyzed metrics, the algorithms behave as in the simulations. In particular, *NoAck-15* is slower than the other two selected variants. *NoAck-5*, in turn, tends to generate additional control traffic in the absence of failures. Let us thus omit these results and instead focus on the long-term behavior of RNFD.

B. Long-Term Experiments

A major problem with failure detectors is that they can make mistakes, which trigger nodes running RNFD to unnecessarily suspect the root. The better a failure detector, the fewer false positives it gives. Let us thus study false positives in *NoAck-10*, as it performs the best among the presented *NoAck-K* variants.

However, quantitatively studying false positives is extremely challenging. On the one hand, in the presented simulations, no false positives were observed for *NoAck-10*. On the other hand, on the testbeds, the inter-node connectivity was influenced by many uncontrollable factors. Therefore, we cannot draw meaningful conclusions from simply comparing RNFD and sole Hybrid because external events may trigger any of the two algorithms to occasionally rebuild parts of the DODAG.

For these reasons, let us focus on the long-term behavior of just *NoAck-10* from the perspective of false positives. More specifically, RNFD with *NoAck-10* and the packet generation interval of 600 s was run for a full week on Testbed B. The goal was to observe the changes of bits in the synopses. Bits set in s_R reflect flapping links to the root. Bits in s_D correspond in turn to suspicions that trigger RNFD. In either case, if a synopsis becomes saturated with set bits, its estimates are less accurate [29]. Figure 11 presents those selected 24 hours from the run in which the number of bit changes was the greatest.



(a) root's all neighbors (s_A & s_R) (b) suspecting neighbors (s_D & s_M)

Fig. 11. The evolution of network synopses during 24 hours on Testbed B.

The selected 24 hours actually include the warm-up period when the DODAG is formed, and hence there are numerous changes to the synopses, but we can also observe changes in

the stable period. However, these latter changes are fairly rare and can be effectively dealt with. For example, changes to s_A and s_R due to flapping links may be disabled at a node by (temporarily) pinning the root in the node's parent set, so that it does not get removed. Likewise, if a node tends to falsely suspect the root, thereby changing s_D and s_M, RNFD may be (temporarily) disabled at that node, as this does not influence the correctness of the algorithm: only a subset of nodes can be used to detect root failures. In addition, to avoid saturating the synopses, they can be periodically reset to zeroes. This can be realized, for instance, by introducing minor DODAG versions that instead of forcing nodes to rebuild the DODAG and reset their Trickle timers, only make them lazily reset their synopses to zeroes. Even rebuilding the DODAG via a (major) version change would probably be acceptable, as this would not be necessary frequently. By and large, RNFD, possibly augmented with these improvements, can likely provide stable long-term performance with only rare changes to the synopses. This also implies that one may consider not embedding the synopses in every beacon, which would minimize the information overhead incurred by RNFD.

VIII. Concluding Discussion

All in all, the results suggest that RNFD has the potential to significantly improve the handling of routing destination crash-failures in low-power wireless networks. By making nodes coordinate their actions, probe a suspected node in parallel even under low data traffic, and abandon laboriously evicting all dead links, RNFD handles failures in seconds, instead of minutes, and with much lower traffic than the state of the art.

At the same time, however, RNFD is not the solution for all conceivable failure scenarios. It handles only crashes of destination nodes under some assumptions and at best, after some adaptation, also failures of important nodes, for instance, bridge nodes between well-connected subnetworks. Moreover, it is a probabilistic solution, which implies that it does not guarantee detecting all failures, relying to this end on existing route maintenance algorithms. Finally, it introduces information overhead that may be problematic in some deployments.

Nevertheless, considering its potential and the fact that it is a framework specifically designed to facilitate integration with existing routing protocols, I hope that RNFD can ultimately be standardized, for instance, as an extension to RPL. In this view, one of the goals of this paper is thus to encourage adoption and further experiments with RNFD. From a broader perspective, in turn, the presented research evidences that the existing routing protocols for low-power wireless networks do leave room for improvement with respect to failure handling. Fault tolerance and reliability of routing protocols thus provide opportunities for innovation.

Acknowledgments

The author would like to thank the shepherd of this paper, Omprakash Gnawali, and the anonymous IPSN'16 reviewers, whose feedback has helped to improve the paper. The presented research was supported by the National Center for Research and Development (NCBR) in Poland under grant no. LIDER/434/L-6/14/NCBR/2015. The author was additionally supported by the Polish Ministry of Science and Higher Education with a scholarship for outstanding young scientists.

References

[1] J.-P. Vasseur and A. Dunkels, *Interconnecting Smart Objects with IP: The Next Internet*. Morgan Kaufmann Publishers Inc., 2010.
[2] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J.-P. Vasseur, and R. Alexander, "RPL: IPv6 routing protocol for low-power and lossy networks," RFC 6550, 2012.
[3] A. Woo, T. Tong, and D. Culler, "Taming the underlying challenges of reliable multihop routing in sensor networks," in *Proc. SenSys '03*, 2003.
[4] K.-H. Kim and K. G. Shin, "On accurate measurement of link quality in multi-hop wireless mesh networks," in *Proc. MobiCom '06*, 2006.
[5] R. Fonseca, O. Gnawali, K. Jamieson, and P. Levis, "Four-bit wireless link estimation," in *Proc. HotNets-VI*, 2007.
[6] K. Srinivasan and P. Levis, "RSSI is under appreciated," in *Proc. EmNets '06*, 2006.
[7] K. Srinivasan, M. A. Kazandjieva, S. Agarwal, and P. Levis, "The β-factor: Measuring wireless link burstiness," in *Proc. SenSys '08*, 2008.
[8] H. Lee, A. Cerpa, and P. Levis, "Improving wireless simulation through noise modeling," in *Proc. IPSN '07*, 2007.
[9] C. A. Boano, H. Wennerström, M. A. Zúñiga, J. Brown, C. Keppitiyagama, F. J. Oppermann, U. Roedig, L.-Å. Nordén, T. Voigt, and K. Römer, "Hot Packets: A systematic evaluation of the effect of temperature on low power wireless transceivers," in *Proc. ExtremeCom '13*, 2013.
[10] K. Srinivasan, P. Dutta, A. Tavakoli, and P. Levis, "Some implications of low-power wireless to IP routing," in *Proc. HotNets-V*, 2006.
[11] D. S. J. De Couto, D. Aguayo, J. Bicket, and R. Morris, "A high-throughput path metric for multi-hop wireless routing," in *Proc. MobiCom '03*, 2003.
[12] J.-P. Vasseur, M. Kim, K. Pister, N. Dejean, and D. Barthel, "Routing metrics used for path calculation in low-power and lossy networks," RFC 6551, 2012.
[13] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis, "Collection tree protocol," in *Proc. SenSys '09*, 2009.
[14] P. Thubert, "Objective function zero for the routing protocol for low-power and lossy networks (RPL)," RFC 6552, 2012.
[15] O. Gnawali and P. Levis, "The minimum rank with hysteresis objective function," RFC 6719, 2012.
[16] K. Iwanicki and M. van Steen, "A case for hierarchical routing in low-power wireless embedded networks," *ACM Transactions on Sensor Networks*, vol. 8, no. 3, pp. 25:1–25:34, 2012.
[17] C. E. Perkins and P. Bhagwat, "Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers," in *Proc. SIGCOMM '94*, 1994.
[18] P. Levis, N. Patel, D. Culler, and S. Shenker, "Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks," in *Proc. NSDI '04*, 2004.
[19] T. Narten, E. Nordmark, W. Simpson, and H. Soliman, "Neighbor discovery for IP version 6 (IPv6)," RFC 4861, 2007.
[20] F. Teraoka, K. Gogo, K. Mitsuya, R. Shibui, and K. Mitani, "Unified layer 2 (L2) abstractions for layer 3 (L3)-driven fast handover," RFC 5184, 2008.
[21] J. Chen, S. Kher, and A. Somani, "Distributed fault detection of wireless sensor networks," in *Proc. DIWANS '06*, 2006.
[22] P. Jiang, "A new method for node fault detection in wireless sensor networks," *Sensors*, vol. 9, no. 2, pp. 1282–1294, 2009.
[23] J. W. Branch, C. Giannella, B. Szymanski, R. Wolff, and H. Kargupta, "In-network outlier detection in wireless sensor networks," *Knowledge and Information Systems*, vol. 34, no. 1, pp. 23–54, 2013.
[24] L. Paradis and Q. Han, "A survey of fault management in wireless sensor networks," *Journal of Network and Systems Management*, vol. 15, no. 2, pp. 171–190, 2007.
[25] A. Pruteanu, V. Iyer, and S. Dulman, "FailDetect: Gossip-based failure estimator for large-scale dynamic networks," in *Proc. ICCCN '11*, 2011.
[26] K. Birman, *Guide to Reliable Distributed Systems*. Springer, 2012.
[27] J. Ko, J. Eriksson, N. Tsiftes, S. Dawson-Haggerty, A. Terzis, A. Dunkels, and D. Culler, "ContikiRPL and TinyRPL: Happy together," in *Proc. IP+SN '11*, 2011.
[28] S. Nath, P. B. Gibbons, S. Seshan, and Z. R. Anderson, "Synopsis diffusion for robust aggregation in sensor networks," in *Proc. SenSys '04*, 2004.
[29] K.-Y. Whang, B. T. Vander-Zanden, and H. M. Taylor, "A linear-time probabilistic counting algorithm for database applications," *ACM Transactions on Database Systems*, vol. 15, no. 2, pp. 208–229, 1990.
[30] P. Flajolet, E. Fusy, O. Gandouet, and F. Meunier, "HyperLogLog: The analysis of a near-optimal cardinality estimation algorithm," in *Proc. AofA '07*, 2007.
[31] T. Pazurkiewicz, M. Gregorczyk, and K. Iwanicki, "NarrowCast: A new link-layer primitive for gossip-based sensornet protocols," in *Proc. EWSN '14*, 2014.
[32] M. Michalowski, P. Horban, K. Strzelecki, J. Migdal, M. Klimek, P. Glazar, and K. Iwanicki, "A sensornet testbed at the University of Warsaw," University of Warsaw, Warsaw, Poland, Tech. Rep. TR-DS-01/12, 2012.
[33] J. Polastre, J. Hill, and D. Culler, "Versatile low power media access for wireless sensor networks," in *Proc. SenSys '04*, 2004.