Depth-First Forwarding for Unreliable Networks: Extensions and Applications

Jiazi Yi, Thomas Heide Clausen, Ulrich Herberg

Depth First Forwarding for Unreliable Networks: Extensions and Applications

Jiazi Yi, Thomas Clausen
Laboratoire d'Informatique (LIX) – Ecole Polytechnique, France
Jiazi@JiaziYi.com, Thomas@ThomasClausen.org

Ulrich Herberg
Fujitsu Laboratories of America
ulrich@herberg.name

***Abstract*—This paper introduces extensions and applications of Depth-First Forwarding (DFF) – a data forwarding mechanism for use in unreliable networks such as sensor networks and mobile ad hoc networks with limited computational power and storage, low-capacity channels, device mobility, etc. Routing protocols for these networks try to balance conflicting requirements of being reactive to topology and channel variation while also being frugal in resource requirements – but when the underlying topology changes, routing protocols require time to re-converge, during which data delivery failure may occur. DFF was developed to alleviate this situation: it reacts rapidly to local data delivery failures and attempts to successfully deliver data while giving a routing protocol time to recover from such a failure. An extension of DFF, denoted DFF++, is proposed in this paper, in order to optimise the performance of DFF by way of introducing a more efficient search ordering. This paper also studies the applicability, of DFF to three major routing protocols for the "Internet of Things", including the Lightweight On-demand Ad hoc Distance-vector Routing Protocol - Next Generation (LOADng), the Optimized Link State Routing protocol version 2 (OLSRv2), and the IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL), and presents the performance of these protocols, with and without DFF, in lossy and unreliable networks.**

I. Introduction

With the advances in micro-controller and wireless technology, the concept of "being online" is no longer exclusively reserved for computers, but expected also for phones, vehicles, televisions, refrigerators, utility meters, etc. *"The Internet of Things"* (IoT) assumes objects in our environment to be part of the Internet, communicating with users and with each other – and that these objects have communication as a commodity, rather than as their reason for existence. Communication in "The Internet of Things" is thus abound with challenges, subject to resource constraints, fragile and low-capacity links, dynamic and arbitrary topologies. Routing is among these challenges, requiring efficient protocols, able to converge rapidly even in very large networks, while exchanging limited control traffic and requiring limited memory and processing power.

Routing protocol is a crucial component of "The Internet of Things". Various proposals exist in the literature, including protocols designed for low-power and lossy networks, sensor networks, MESH and Mobile Ad hoc NETworks (MANETs). While making different design trade-offs, all are designed to limit the routing overhead imposed to networks as much as possible, and to be adapted to the varying nature of the communication media, mobility, interference, etc. However, even once routes to a destination have been found, they might become unusable, in a non-predictable time-varying fashion: dynamic topology, presence of noise or interferences, low power supply in certain devices, uni-directional links, etc. From a routing protocol point of view, when a link failure is detected, signalling and/or time is required to recover and discover new, valid routes, if such are available in the network. During this recovery phase, data packets being sent over the broken link must either be buffered and wait for the route recovery, or be dropped because of lack of memory in constrained devices. Some preliminary study has been conducted in [1], mainly focusing on low-lower and lossy networks. Compared to the previous short conference paper, more detailed discussion of DFF are presented in this paper. Its applications to major routing protocols for "Internet of Things", including MANET routing protocol OLSR [8] and LOADng [15], and routing protocol for Low-power and Lossy networks RPL [10] are studied .

"Depth-First Forwarding in Unreliable Networks" (DFF) [2] is an experimental data forwarding standard that proposes a mechanism for rapid and localised recovery in case of link failure. Colloquially speaking, if a device fails in its attempt to forward a packet to its intended next-hop, then DFF suggests a heuristics for "trying another of that devices' neighbours", while keeping track of (and preventing) packet loops. While DFF can operate independently, *i.e.,* without a routing protocol (which amounts to simply doing a depth-first exploration of the network), it can also be used conjointly with a routing protocol: the routing protocol can provide an "order of priority" of the neighbours of a device, in which data delivery should be attempted – and DFF can also signal to a routing protocol when data delivery to a destination has (possibly repeatedly) failed via a neighbour but (possibly repeatedly) succeeded via another neighbour. [3] compares the performance of DFF with that of other data forwarding, showing that DFF can significantly improve the reliability of the data transmission in networks with unstable topology.

A. Background and History: A Tale of Three Protocols

Since the late 90s, the Internet Engineering Task Force (IETF)[1] has embarked upon a path of developing routing

[1] http://www.ietf.org/

protocols for networks with increasingly more fragile and low-capacity links, with less pre-determined connectivity properties and with increasingly constrained router resources. In '97, by chartering the MANET (Mobile Ad hoc Networks) working group, then subsequently in 2006 and 2008 by chartering the 6LowPAN (IPv6 over Low power WPAN) and ROLL (Routing Over Low power and Lossy networks) working groups, and more recently by creating the 6lo (IPv6 over Networks of Resource-constrained Nodes), 6TiSCH (IPv6 over the Time-slotted Channel Hopping mode of IEEE 802.15.4e), and CORE (Constrained RESTful Environments) working groups.

1) MANET Protocol Developments: The MANET working group converged on the development of two protocol families: reactive protocols, including the "Ad hoc On-Demand Distance Vector" (AODV) protocol [4], and proactive protocols, including the Optimized Link State Routing (OLSR) protocol [5]. A distance vector protocol, AODV operates in an *on-demand* fashion, acquiring and maintaining routes only while needed for carrying data, by way of a *Route Request–Route Reply* exchange. A link state protocol, OLSR is based on periodic control messages exchanges, and each router proactively maintaining a routing table with entries for all destinations in the network, which provides low delays but constant control overhead. A sizeable body of work exists, including [6], studying the performance of these protocols in different scenarios, and justifying their complementarity [7]. OLSR provides low delays and predictable, constant control overhead – at expense of requiring memory in each router for maintaining complete network topology. AODV limits the memory required for routing state to that for actively used routes – at the expense of delays for the *Route Request–Route Reply* exchange to take place, and control overhead dependent on data flows.

After acquiring operational experiences, the MANET working group commenced developing successors to OLSR and AODV, denoted OLSRv2 and DYMO. Whereas a relatively large and active community around OLSR thus standardised OLSRv2 [8], the momentum behind DYMO withered in the MANET working group[2].

2) 6LowPAN, ROLL and G3-PLC Protocol Developments: The 6LowPAN working group was chartered for adapting IPv6 for operation over IEEE 802.15.4, accommodating characteristics of that MAC layer, and with a careful eye on resource constrained devices (memory, CPU, energy, ...). Part of the original charter for this working group was to develop protocols for routing in multi-hop topologies under such constrained conditions, and over this particular MAC. Two initial philosophies to such routing were explored: *mesh-under* and *route-over*. The former, mesh-under, would, as part of an adaptation layer between 802.15.4 and IP, provide layer 2.5 multi-hop routing, presenting an underlying mesh-routed multi-hop topology as a single IP link. The latter, route-over, would expose the underlying multi-hop topology to the IP layer, whereupon IP routing would build multi-hop connectivity. Several proposals for routing were presented in 6LowPAN, for each of these philosophies, including LOAD [9]. LOAD was a derivative of AODV, but adapted for layer 2-addresses and mesh-under routing, and with some simplifications over AODV (*e.g.,* removal of intermediate node replies and sequence numbers). However, 6LowPAN was addressing other issues regarding adapting IPv6 for IEEE 802.15.4, such as IP packet header compression, and solving the routing issues was suspended, delegated to a working group ROLL, created in 2008 for this purpose. ROLL produced a routing protocol denoted "Routing Protocol for Low-power lossy networks" (RPL) [10] in 2011.

While LOAD [9] development was suspended by the 6LowPAN working group, pending the results from ROLL and experiences with RPL, AODV derivatives live on: IEEE 802.11s [11] is partly based on AODV, and the G3-PLC standard [12], published in 2011, specifies the use of LOAD at the MAC layer, for providing mesh-under routing for utility (electricity) metering networks. Justifications for using an AODV derivative in preference to RPL include that the former better supports bi-directional data flows such as a request/reply of a meter reading [13], as well as algorithmic and code complexity reasons [14]. The emergence of Low-power and Lossy Networks (LLN) thus triggered a renewed interest in AODV-derived protocols for specific scenarios, resulting in work within the IETF [15], [16] for the purpose of standardisation of LOADng, incorporating the experiences from deploying AODV – including, but not only, in LLNs.

During the winter 2012-2013, LOAD was adopted by the G3-alliance, and integrated into the ITU standard G.9956, ratified as a normative appendix specifying the layer 2 routing mechanism for power-line communication for AMI networks. LOADng was adopted by the G3-alliance in G.9903 Amendment 1 [17], [18].

B. Statement of Purpose

Depth-first search (DFS) and position based routing algorithms are abound in literature. In [19], a localised algorithm that using GPS information for QoS routing decisions is proposed. When the hop-count metric is used, the length of the produced QoS path is close to the shortest path algorithm. In [20], the DFS is further optimised, and integrated with power metrics minimising total power for routing packets. OWL (Ordered Walk with Learning protocol) [21] is a distributed approximation of DFS with the known topology information to reduce the search tree. These algorithms run within routing protocols, *i.e.,* result in routing table entries being installed, and most do require additional information (topology, location of source or destination) for making the routing decisions.

This paper explores extensions to and applications of depth-first search, referred as forwarding mechanisms (DFF) in networks for Internet of Things, such as MANETs and sensor networks. DFF can operate independently, or can operate conjointly with a routing protocols. One key issue of DFF – selecting appropriate next-hop candidates for forwarding data packets, is especially studied and discussed. Based on

[2]http://tools.ietf.org/wg/manet/minutes?item=minutes81.html

experiments and observations, an optimisation to the DFF standard [2], denoted DFF++, is proposed. It offers a procedure for choosing next hops based on previously forwarded packets, and reduces unnecessary explorations of dead-end branches in a depth-first search. The extension is of very low impact on DFF: it does not introduce additional protocol signalling or data sets, and remains completely interoperable with DFF as specified in [2]. The use of DFF, and its extension, conjointly with routing protocols such as LOADng, OLSR and RPL are also studied.

C. Paper Outline

The remainder of this paper is organised as follows. Section II provides a brief overview of DFF, as defined in [2], and section III studies, by way of an example, one of the cardinal points of the performance of DFF: the ordering of the elements of the so-called "Candidate Next Hop List". Section III also identifies a set of inconveniences in a "naive" (but, perfectly valid, according to [2]) ordering. [2] stipulates some basic constraints on how the Candidate Next Hop List is to be ordered, but otherwise leaves the exact approach and order unspecified. This paper, therefore, proceeds by proposing a simple and overhead-free optimisation to DFF, denoted DFF++, in section IV. This optimisation remains fully interoperable with DFF. The use of DFF conjointly with the routing protocols LOADng, OLSRv2, and RPL, is studied in section V, section VI, and section VII respectively, and performance results by way of network simulations are presented. Finally, this paper is concluded in section VIII.

II. Depth First Forwarding

DFF [2] is a forwarding mechanism for improving the data delivery success ratio across unreliable multi-hop networks. It operates solely on the forwarding plane, *i.e.,* does not assume any specific routing protocol to be in operation (or, indeed, that any routing protocol is in operation) – but can, as appropriate and as indicated in section I of this paper, interact with a routing protocol. DFF relies on an external mechanism providing each router with a list of its neighbours.

DFF specifies mechanisms for detecting looping data packets, encoded as flags and sequence numbers in IPv6 hop-by-hop header options, carried in each data packet. This additional header incurs a small, but fixed, per-data-packet overhead of 8 octets. This paper does not discuss this signalling and processing in further details.

In order to support the loop detection and duplicate detection, each router running DFF maintains a "Processed Set", which lists sequence numbers of previously received packets, as well as a list of next hops to which the packet has been sent successively as part of the depth-first forwarding mechanism.

Schematically, the basic operation of DFF is as follows, when a data packet for a destination arrives at the forwarding plane of a router:

1) The router temporarily creates an ordered Candidate Next Hop List for that packet from among the neighbours in the router's neighbour list. The list does not contain the neighbour from which the data packet was received (if any).
2) The router attempts to forward the data packet to the first neighbour in the resulting Candidate Next Hop List.
3) There are five possible outcomes from this attempt:
   a) The Candidate Next Hop List is empty, in which case the data packet is returned to the neighbour from which it was initially received, and the process for this router stops.
   b) Delivery to that neighbour succeeds (*e.g.,* as confirmed by a layer 2 acknowledgement), and that neighbour is the destination for the data packet. The layer 2 acknowledgement indicates successful data packet delivery to the destination. The process for this router stops.
   c) Delivery to that neighbour fails (*e.g.,* detected by lack of a layer 2 acknowledgement), in which case that neighbour is removed from the Candidate Next Hop List, and the process resumes at step 2 above.
   d) Delivery to that neighbour succeeds (*e.g.,* as confirmed by a layer 2 acknowledgement), but the data packet is returned from the neighbour as "undeliverable", in which case that neighbour is removed from the Candidate Next Hop List, and the process resumes at step 2 above, with the resulting Candidate Next Hop List.
   e) Delivery to that neighbour succeeds (*e.g.,* as confirmed by a layer 2 acknowledgement), the neighbour is not the destination for the data packet. That neighbour will, now, execute this very same procedure (create its own Candidate Next Hop List, and execute this process starting at step 1).

The initial Candidate Next Hop List for a data packet, by default, contains all the neighbours of a router, except for the neighbour from which the data packet was received, but may be smaller. Section 11 in [2] suggests several criteria to take into account when ordering that list, including that if a routing protocol is in operation, then the neighbour on the shortest path (as indicated by that routing protocol) must be part of the initial Candidate Next Hop List – and is recommended to be first in that initial Candidate Next Hop List. Link quality, historical information on "good and bad neighbours as next hop" is suggested to be used for ordering remaining neighbours.

The order of the Candidate Next Hop List has crucial influence to the performance of the search. The average complexity is *O(N)* (*N* is the number of nodes), linear in the size of the network. In the worst case, the packet has to travel all the nodes before reaching the destination. Due to the importance of the Candidate Next Hop List, it is further studied in the next section.

III. Ordering the Candidate Next Hop List

Section II has introduced the basic operations of DFF, indicating that a key operational parameter is the ordering of elements in the Candidate Next Hop List for a data packet.

To elaborate on this parameter, this section will consider the example in figure 1, where device *A* sends a data packet to device *D*, and which arrives at *B* – the sole neighbour of *A*.

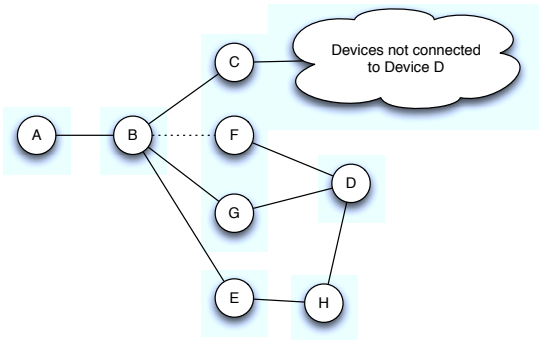

Figure 1. An example of DFF: device *A* sends packets to device *D*. The dotted line represents a broken link.

A. Ideal Ordering and Default Ordering

The ideal ordering of the elements in the Candidate Next Hop List for that data packet in *B* would list *F* and *G* first, followed by *E*. The ideal list would not include *C* at all. Absent topological information (maintained by some external process), however, *B* would by default populate the initial Candidate Next Hop List with all of its bi-directional neighbours, except for *A*. *B* would not be able to determine that *E* should be listed after *F* and *G*, nor that *C* should be excluded from the list. This, would lead to a "blind" search for all the DFF forwarded data packets. Indeed, in this example constructing the Candidate Next Hop List lexicographically – $\{C, E, F, G\}$ – would lead to the worst possible search order.

B. Ordering with Unicast Routing Protocol

If a unicast routing protocol is in place, as suggested by section 11 in [2], that routing protocol would provide *B* with information that *F* (or *G*) is the next hop identified on the shortest path to *D*, and therefore allow *B* to ensure that the first element in the Candidate Next Hop List for a data packet for *D* would be *F*. Unless if the routing protocol provides multiple paths or a complete topology in each device – unlikely, given that the application space is constrained devices with limited memory – the remainder of the list would have to be constructed without any additional guidance from the routing protocol as to a specific order of elements. A simple lexicographical order of the remaining elements, as in the above, would result in $\{F, C, E, G\}$.

C. When Links Break

The Candidate Next Hop List order, obtained when using a unicast routing protocol as illustrated above, is better than without – but is still not ideal. Consider that with the Candidate Next Hop List being $\{F, C, E, G\}$, and data packets being successfully transmitted from *A* to *D* along the path *A-B-F-D*. Now, the link between device *B* and *F* breaks – which would be detected by *B* when trying to forward a data packet to *F*. *B* would then remove *F* from the Candidate Next Hop List, and forward it to the next entry – *C*. As *C* is not on any route to device *D*, the packet would eventually be returned to device *B*, after having traversed the network indicated in the "cloud" in figure 1, and *B* would be able to remove *C* from the Candidate Next Hop List for that data packet.

D. Candidate Next Hop List Per Packet

[2] specifies that the Candidate Next Hop List is constructed per data packet. [2] also specifies that DFF may signal to the routing protocol when delivery to a next hop, indicated by the routing protocol, fails such that the routing protocol can take corrective action (*e.g.,* remove the entries from the routing table, corresponding to the failed next hop, and initiate recovery as specified by the routing protocol). In the example above, if DFF signals to the routing protocol that *F* has failed, and if the routing protocol then removes routing table entries indicating *F* as next hop, then for *all* subsequent data packets to *D* (until the routing protocol recovers and establishes a new entry in the routing table), the initial Candidate Next Hop List will be $\{C, E, G\}$ – back to the "worst case" default ordering.

IV. DFF++: The Destination Field Extension

As introduced in section II, when data delivery of a data packet fails, DFF removes the failed "next hop entry" from the Candidate Next Hop List for that data packet, and forwards the data packet to the next entry (if any) in that list. Section III illustrated, by way of an example, that while DFF thus may eventually succeed in delivering data packets to the intended destination, the efficiency of that operation – the path-length and number of forwards of a data packet – depends on the ordering of entries in that Candidate Next Hop List. As the Candidate Next Hop List is constructed *per-packet*, several subsequent data packets to the same destination may take the same "detour" through the network (or, in the example in figure 1, all explore the same "blind alley" in the network by way of *C*) – either persistently, or, if a unicast routing protocol is also operating in the network, until such time that that unicast routing protocol has recovered from the failure and provided a new entry for the destination in the routing table.

This section proposes a simple extension to DFF, henceforth denoted DFF++, for establishing "memory" across several data packets for the same destination. In the interest of being frugal with required state, this extension (i) piggy-bags off information already maintained by DFF, and (ii) maintains information only temporarily, for as long as DFF otherwise maintains information pertaining to forwarded packets.

A. State

In order to support loop and duplicate detection, each device running DFF maintains a Processed Set, which records sequence numbers of previously received data packets as well as a list of next hops to which each data packet has been successively sent, as part of the depth-first forwarding mechanism. Without going into the details of the loop and

duplicate detection mechanisms in DFF (refer to [2]), the "Processed Set" consists of "Processed Tuples", of the form:

```
(P_orig_address, P_seq_number,
P_prev_hop, P_next_hop_neighbor_list,
P_time)
```

where:

- `P_orig_address` is the originator address of the received packet;
- `P_seq_number` is the sequence number of the received packet;
- `P_prev_hop` is the address of the previous hop of the packet;
- `P_next_hop_neighbor_list` is a list of addresses of next hops to which the packet has been sent previously, as part of the depth-first forwarding mechanism;
- `P_time` specifies when this tuple expires and must be removed.

The proposed DFF++ extensions adds an element to each such tuple, thus:

```
(P_orig_address, P_seq_number,
P_prev_hop, P_next_hop_neighbor_list,
P_time, P_dest_address)
```

where:

- `P_dest_address` indicates the destination address of the received packet.

The proposed DFF++ extension also imposes an additional constraint on `P_next_hop_neighbor`, which is that:

- `P_next_hop_neighbor` must be ordered such that the last element (`P_next_hop_neighbor_list[LAST]`) of that list contains the last neighbour to which delivery to `P_dest_address` was attempted.

B. Processing

On receiving a data packet, not destined to itself, DFF++ defines the following process for selecting an ordered Candidate Next Hop List (CNHL), within the constraints and guidelines from section 11 in [2].

Find the (unique) Processed Tuple, where:

- `P_dest_address` == the destination address of the data packet; AND
- which has the greatest `P_time`.

Using that tuple, the CHNL is constructed thus (where $\oplus$ indicates list concatenation, $\setminus$ indicates list exclusion, RT(address) is the next hop on the shortest path to the destination from the routing table – if any, and NS indicates the set of neighbours of the device):

1) CNHL = RT(`P_dest_address`)
2) CHNL = CHNL $\oplus$ `P_next_hop_neighbor_list[LAST]`
3) CHNL = CHNL $\oplus$ {NS$\setminus$ {`P_prev_hop`} $\setminus$ `P_next_hop_neighbor_list`}
4) CHNL = CHNL $\oplus$ `P_next_hop_neighbor_list`

Where 1) satisfies the requirement from [2] that the first element in the CNHL is the next hop, indicated by a routing table (if present). Items 2) and 3) capture "pick up where the most recent data packet delivery to the same destination left off". Specifically, 2) is the neighbour, last tried for the most recent packet to the same destination, and which is not yet confirmed as having failed (in which case there would be a subsequent entry in the list, except if all neighbours had been tried and failed), 3) includes all other so far untried (by the most recent data packet delivery for this destination) neighbours. Finally, 4) – which is an optional step in DFF++ – includes all previously (by the most recent data packet delivery) tried neighbours, capturing the fact that a previous failure may have been due to transient losses. This excludes, of course, the neighbour from which the data packet was received.

C. Impact

Adding and using `P_dest_address`, as described above, allows construction of the Candidate Next Hop List to make use of information on previous data packet forwards to the same destination.

Returning to the example in figure 1, one of the issues raised in section III-C, and detailed in section III-D, is alleviated:

1) The initial Candidate Next Hop List for the first data packet arriving at *B* for destination *D* will – using the same ordering (routing table entry first, then the the "worst-case" lexicographical order) be {*F*, *C*, *E*, *G*}.
2) Initial delivery is attempted via *F* (which is added to the end of `P_next_hop_neighbor_list`) and fails, and delivery via *C* is attempted (which is added to the end P_next_hop_neighbor_list).
3) Delivery via *C* also fails (no path via *C* to *D*), and delivery is now attempted via *E* (which is added to the end of `P_next_hop_neighbor_list`) – as there is a valid path to *D* via *E*, delivery succeeds, and the P_next_hop_neighbor[LAST] for that processing tuple now contains *E*.
4) Other data packets for *D*, arriving at *B*, *before* the routing protocol (if any) has recovered and provided an entry in the routing table for *D*, will, using the DFF++ Candidate Next Hop List construction rules given in section IV-B, result in a Candidate Next Hop List of:
   - If they arrive after step 3), {*E*, *G*} – thus avoiding the "broken link" to *F*, as well as the "blind alley" that would be attempting delivery via *C*.
   - If they arrive after step 2) but before step 3), {*C*, *E*, *G*} – thus avoiding the "broken link" to *F*, but not the "blind alley" that would be attempting delivery via *C*
   - If they arrive before step 2), {*F*, *C*, *E*, *G*} – thus offering no improvement over DFF, but also no additional penalty.

Given a network with n routers, the worst case for DFF is that the entire network needs to be traversed before reaching the destination, which takes $O(n)$ tries. When DFF++ is applied, it will not change the behaviour of the packets to the destination that the router has not forwarded before. However, if the destination is already known to the router, DFF++ can reduce the retry time to $O(1)$. Note that DFF++ avoids the problem of repeatedly attempting delivery to a given destination via "blind alleys" and over "recently detected broken links", but does not attempt at offering "shortest paths" – that remains under the auspices of a routing protocol (if any) in the network. Also, DFF++ does not affect interoperability: the extension does not introduce any new signals or any new external behaviours, but simply offers guidance for how to order the Candidate Next Hop List for a data packet. The specification of DFF [2] specifically encourages an intelligent ordering, and DFF++ does just that. As that ordering of the Candidate Next Hop List for a data packet concerns only internal processing of a device, DFF and DFF++ remain interoperable. DFF++ can furthermore be deployed with exactly the same (or no) unicast routing protocols as DFF.

V. Application of DFF to LOADng

This section studies the application of DFF to LOADng. An overview of LOADng is firstly provided and followed by the interaction between LOADng and DFF to alleviate possible packet loss. Simulations are then presented, studying the performance of different protocol settings.

A. LOADng Overview

As a reactive protocol, LOADng only keeps the routing information to desired destinations. If a data packet is to be sent to an unknown destination, a route discovery is triggered "on-demand". The basic operations of LOADng [15] include generation of Route Requests (RREQs) by a LOADng router (originator) and flooded through the network when discovering a route to a destination, and Route Replies (RREPs) generated by the sought destination ad transmitted to the originator by way of unicasts. When an intermediate router forwards a RREQ, it installs temporary routing table information towards the originator of the RREQs – the "reverse route" from the destination to the originator. When the sought destination receives a RREQ, it will respond by a unicast RREP, which is forwarded along this installed reverse route – and the forwarding of which will serve to install a "forward route" from the originator to the destination. Thus, for each bidirectional path through a LOADng router, four entries are maintained in the routing table: for directions, an entry is recorded for the "next hop" and for the "destination" via that "next hop".

If a route is detected broken, *i.e.,* if forwarding of a data packet to the recorded next hop on the route towards the intended destination is detected to fail, a Route Error (RERR) message is returned to the originator of that data packet. The LOADng specification stipulates that when the transmission of a data packet fails, that data packet is dropped and a RERR is sent back to its source – which can, then, trigger a new route discovery.

B. LOADng with DFF

DFF requires that a router has a list of all its neighbours available for constructing the Candidate Next Hop List for a data packet. [2] specifies that an external mechanism is to be in place to provide that list, and suggests the use of NHDP (Neighborhood Discovery Protocol) [22] – which is implemented and used for the purpose of the performance studies in this paper.

The routing protocol LOADng provides, at most, one entry in the routing table for each destination, thus the integration of the requirements for ordering the entries in the Candidate Next Hop List for a data packet is met simply by, if a routing table entry for the destination is present, inserting this first in that list. The remainder of the entries in the Candidate Next Hop List are, simply, all the other neighbours discovered by NHDP (and with status SYMMETRIC), excluding of course the neighbour from which the data packet was received.

Additionally, the two following rules govern the application of DFF to LOADng, for the purpose of the studies in this paper, specifically when the protocol operations for each are activated:

- When a router receives a data packet from another router, for which it does not have a corresponding entry in the routing table:
  - Send the data packet according to the DFF forwarding rules, as described in section II.
  - Send a RERR message to the originator of that data packet, as described in section V-A.

  A RERR message is sent since while DFF will ensure data delivery, this may be by way of an excessively long path; by sending a RERR message, the routing protocol is instructed to "try to find a better path" whilst DFF concurrently attempts delivery of data in transit (thus reducing delays, retransmissions and/or buffer of data traffic).
- If forwarding of a data packet to the next hop, indicated by LOADng (*i.e.,* the first entry in the Candidate Next Hop List) fails (either by way of the packet being returned by DFF, or by a layer 2 acknowledgement being absent):
  - Send the data packet according to the DFF forwarding rules, as described in section II.
  - Send a RERR message to the originator of that data packet, as described in section V-A.

  In this case, a RERR message is sent since, in addition to the reasons listed above, this is indicative of the routing information being inconsistent with the network topology, and therefore needs to be updated.

With the example given in figure 1, the route originally found by LOADng protocol was *A-B-F-D* (one of the shortest paths). However, when a data packet arrived at node *B*, the link *B-F* was detected broken. By using DFF, a neighbour node from the Candidate Next Hop List, node *C*, for example, is chosen as next hop. The data packet is thus forwarded to node *C*, which will handle the packet according its routing table information or DFF, and forward it to the destination node *D*. In the meantime, node *B* will send a RERR message to node *A*, to notify the route failure.

C. Simulation and Analyses

1) Simulation Settings: In order to evaluate the performance of DFF++, and compare its performance to that of DFF, network simulations by way of NS2 are employed. While network simulations are, at best, an approximation of real-world performance (particularly due to the fidelity of their lower layers to reality), they do provide a baseline for comparison and, generally, best-case results, *i.e.,* real-world performance is expected to be no better than that which is obtained through simulations. The reason for using network simulations is that it allows running experiments with different protocols under identical conditions and parameters (MAC layer, distribution, number of nodes, etc.).

Simulations were conducted using the TwoRayGround propagation model and the IEEE 802.11 MAC. Although there are various low-layer technologies more commonly (and, perhaps, more viably) used for LLNs (power line communication, 802.15.4, low-power wifi, bluetooth low energy, etc.), general behaviours of a protocol can be inferred from simulations using 802.11.

The general network topology of a scenario is as follows: n (from 63 to 500) devices are placed randomly (while ensuring that the network is still connected) in a square field, such that to maintain a constant device density. There are $n-1$ Constant Bit Rate (CBR) streams in the network. The original node of the CBR stream (chosen randomly) sends one packet of 512 octets every 5 seconds to a destination (chosen randomly). As DFF is supposed to be particularly beneficial in lossy networks the simulations enforce that a packet is lost with a probability of 20%. Simulations were run for 100 seconds each, and for each datapoint 20 different and randomly generated scenarios – all corresponding to the same abstract parameters – were simulated, with the results shown below representing averages from among these.

2) Results and Discussions: DFF and DFF++ were evaluated both in isolation (without a concurrently operating unicast routing protocol), when used in conjunction with LOADng, as well as compared with LOADng operating alone in the same networks, so as to evaluate the benefits of DFF and DFF++, respectively. In total, five different protocol combinations were evaluated:

- *DFFonly*: DFF according to RFC6971 [2], with the Neighbourhood Discovery Protocol (NHDP) [22] used for bi-directional neighbour discovery.
- *DFF++only*: The DFF++ extension as described in section IV, with RFC 6971 [22].
- *LOADng*: LOADng, according to [15].
- *LOADngDFF*: LOADng with DFF ([2], [22] and [15]).
- *LOADngDFF++*: LOADng with DFF++ ([15] plus the process described in section IV and with [22]).

For NHDP [22], a HELLO message interval must be chosen. The shorter the HELLO message interval, the more accurate a list of neighbours can be acquired (and so, the better can DFF and DFF++ do their jobs) – but at the expense of increased control traffic

overhead. For the purpose of these simulations, a HELLO interval of 1s was (arbitrarily) chosen as it represents a "very frequent" HELLO message exchange and therefore a good "worst case" example. In a deployment, the HELLO interval should be selected so as to correspond to the expected local network topology change rate.

Figure 2 depicts the packet delivery ratio of the different protocols combination. When DFF (DFFonly and DFF++only) is running without an external routing protocol, DFF++ offers a significant improvement of the packet delivery ratio over DFF. The lower performance, experienced when running without an external routing protocol is due to depth first searching being inefficient, worst case causing a complete transversal of the network graph for each data packet. DFF, used with LOADng, yields about 20 percentage points improvement of the delivery ratio, as compared to LOADng alone, and DFF++ used with LOADng further improves the data delivery ratio – albeit marginally so.

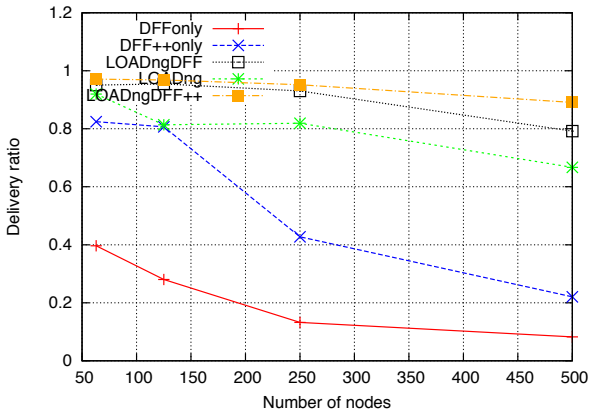

Figure 2. Average data packet delivery ratio

The average end-to-end delay is depicted in Figure 3. It should be noted, particularly for this figure, that only data packets that successfully arrive the destination are accounted in the statistics. DFF and DFF++ alone exhibit significantly greater delays than when running with LOADng. Balance with the data delivery ratio in figure 2, this is noteworthy: combining DFF/DFF++ yields lower delays and better data delivery. LOADng alone exhibits a slightly lower delay than when compared with DFF and DFF++ – which is compensated by the fact that inclusion of DFF/DFF++ increases the data delivery ratios obtained by approximately 20 percentage points.

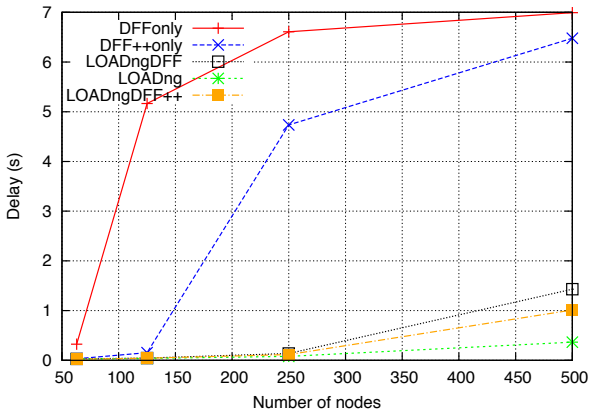

Figure 3. Average end-to-end delay

Figure 4 depicts the average path length (of successfully delivered packets), *i.e.,* the number of hops required for a packet to reach its destination. When running without an external routing protocol, the "blind" depth-first search of DFF causes 6-7 times as long paths as LOADng – with DFF++ offering shorter path lengths than DFF. When combined with LOADng, both DFF and DFF++ yield significantly shorter path lengths, as compared to DFF/DFF++ alone – and slightly longer path-lengths than when running LOADng alone. This, again, is explained by the fact that LOADng with DFF/DFF++ increases the data delivery ratio.

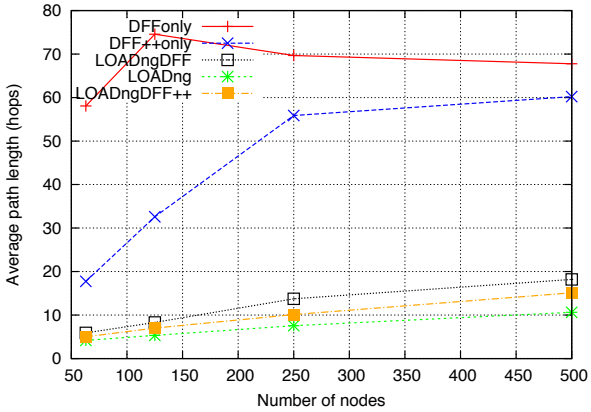

Figure 4. Average path length

Figure 5 illustrates the control packet overhead. For DFF and DFF++ without an external routing protocol, the overhead is constituted from locally exchanged HELLO messages, generated by NHDP to discover the bi-directional neighbours. When introducing LOADng (either alone, or in conjunction with DFF/DFF++), the protocol overhead of that routing protocol for route discovery (see [23] for details) is also imposed on the network, and causes additional MAC layer collisions.

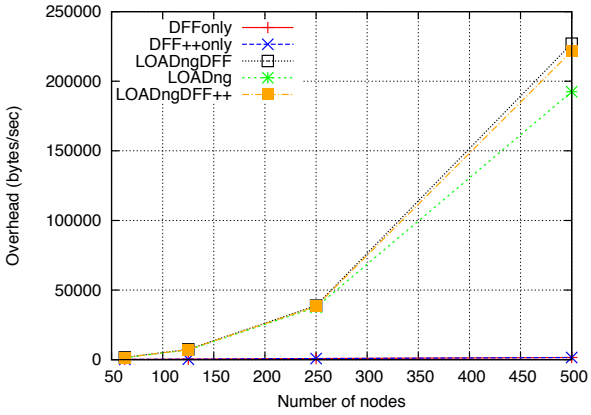

Figure 5. Control packet overhead

VI. Application of DFF to OLSR

This section studies the application of DFF to OLSRv2. An overview of OLSRv2 is firstly provided and followed by the interaction between OLSRv2 and DFF to alleviate possible packet loss. Simulations are then presented, studying the performance of different protocol settings. Note that while the results of this section are based on OLSRv2, expected results for OLSR(v1) are very similar due to the same core algorithm that is used in both OLSR and OLSRv2.

A. Optimized Link State Routing version 2

OLSRv2 (version 2) [8] periodically sends routing control messages to maintain the network topology inside each router. Compared to reactive protocol such as LOADng, OLSRv2 provides low delays and predictable, constant control overhead, at expense of requiring memory in each router for maintaining the network topology.

OLSRv2 contains three basic processes: Neighbourhood Discovery, MPR (Multi-Point Relay) flooding and link state advertisements, briefly described in the below.

1) Neighbourhood Discovery: Neighbourhood discovery is the process, whereby each router discovers the routers which are in direct communication range of itself (1-hop neighbours), and detects with which of these it can establish bi-directional communication. Each router sends HELLOs, listing the identifiers of all the routers from which it has recently received a HELLO, as well as the "status" of the link (heard, verified bi-directional).

2) MPR Flooding: MPR flooding is the process whereby each router is able to, efficiently, conduct network-wide broadcasts. Each router designates, from among its bi-directional neighbours, a subset (MPR set) such that a message transmitted by the router and relayed by the MPR set is received by all its 2-hop neighbours. MPR selection is encoded in outgoing HELLOs.

3) Link State Advertisement: Link state advertisement is the process whereby routers are determining which link state information to advertise through the network periodically. Each router must advertise, at least, all links between itself and its MPR-selector-set, in order to allow all routers to calculate shortest paths. Such link state advertisements are carried in TCs, broadcast through the network using the MPR flooding process described above.

4) Routing and Forwarding: OLSRv2 is a routing protocol, which implies that it acquires a topology database describing the network, and then produces a routing table – typically, handed off the the underlying operating system for use when forwarding data. OLSRv2 does have the ability to use various "triggers" (for example, if an external signal is available indicating that a local link is broken) to use for updating the topology database, but absent such relies on pure periodic control message exchange. Thus, assuming default parameters for OLSRv2 control message intervals, from a link is broken and until this has been reflected through the entire network, around 8 seconds will elapse while the protocol converges – 8 seconds, during which IP datagrams potentially are dropped even if an alternative forwarding path might exist in the network. Integrating DFF with OLSR is expected to be beneficial for data delivered during that time, improving the data delivery rate in across a network wherein links are either lossy, rapidly changing, or both.

B. OLSRv2 with DFF

As indicated in section II, DFF relies on an external mechanism for providing neighbourhood information. When used in conjunction with OLSR, a neighbourhood discovery process [22] is already in place, providing the required information for constructing the Candidate Next Hop List. Additionally, OLSRv2 generates a routing table, providing next hop information for each destination that the routing protocol is aware of. The corresponding entry from this routing table is – as recommended by [2] – if present, is used as the first element in the Candidate Next Hop List.

The DFF and DFF++ forwarding mechanisms are applied only to unicast data traffic, not to broad/multicast traffic such as OLSRv2 control traffic, according to the steps indicated in section II.

When a router attempts forwarding an IP datagram to a next hop, according step 2 in section II, one of the five outcomes in step 3 can result. If the outcome is either of step 3b, 3d, or step 3e, then the link to that neighbour is "good" – and that information may be used for increasing the link quality for that link, as per section 14 of [22]. Conversely, if delivery to that neighbour fails, as per step 3c, this may be used for decreasing the link quality for that link. So increasing and decreasing the link quality provides a signal for OLSRv2 to determine if a given link should be admitted, or not, as part of the usable network topology. If a IP datagram is successfully delivered to a neighbour, which subsequently returns it as "undeliverable" (as per step 3a), the neighbour which returned the IP datagram is definitely not a valid next hop for the given destination. Any routing table entry indicating that neighbour as next-hop for the destination of the returned IP datagram is, therefore, removed.

C. Simulation and Analyses

This section describes the simulation settings and results.

1) Simulation Settings: The simulations are performed to evaluate the application of DFF to OLSR. The simulator and network interface settings are identical with section V-C1.

Because OLSRv2 is designed for MANETs, a dynamic network scenario is employed in the simulations. There are 50 mobile routers running OLSRv2, moving with different velocities (0 – 10 meters per second) in a 1000m $\times$ 1000m square. 50 CBR concurrent streams exist in the network, sending one packet of 512 octets every 5 seconds to a random destination. As the network is dynamic, the link breakages and routing failure would depend on the speed of the routers: the faster the routers move, the more routing failures will occur in the network.

2) Results and Discussions: DFF and DFF++ are applied to OLSRv2, and compared to the original OLSRv2. Three different protocol combinations were evaluated:

- *OLSRv2*: the original OLSRv2 implementation based on [8].
- *OLSRv2DFF*: OLSRv2 with DFF.
- *OLSRv2DFF++*: OLSRv2 with the DFF++ extension.

The packet delivery ratio is shown in figure 6. All three settings have similar performance in static scenarios (*i.e.,* routers do not move), because the network is stable and routing failure is very rare. However, the packet loss increases significantly for OLSRv2 as the network becomes more dynamic. The packets are dropped if the next hop in the routing set moves out of the transmission range. DFF can greatly improve the packet delivery ratio in such unreliable scenarios. When the link to the next ho in the routing set is detected broken, the packet is redirected by DFF, until it reaches the desired destination (or is dropped if TTL = 0).

The proactive nature makes OLSRv2 suitable for use with DFF: as the local routing set keeps the next hop information to all the possible destinations in the network, OLSRv2 provides already the first element in the Candidate Next Hop List for DFF. On receiving a data packet forwarded by DFF, the router can first search its routing set, rather than blindly in the neighbour set – which is the case for LOADng most of the time, because LOADng discoveries the route reactively, and thus normally does not possess the routing information outside the originally built path. For the same reason, the improvement brought by DFF++ is relatively moderate (but constant), because the purpose of DFF++ is to avoid "blind searches".

Figure 7 and figure 8 show the average end-to-end delay and overhead respectively. Because packets forwarded by DFF tend to travel on longer paths compared to original OLSRv2 until reaching the destination, they would take more time and result in higher overhead. However, it is important to mention that the delay is counted based on the packets that successfully reach the destinations. Considering the significant improvement in packet delivery ratio, the trivial cost in delay and overhead can be considered acceptable.

VII. Application of DFF to RPL

In this section, the application of DFF to RPL is studied. A brief overview of RPL is first given, followed by discussions on why RPL is not suited for DFF.

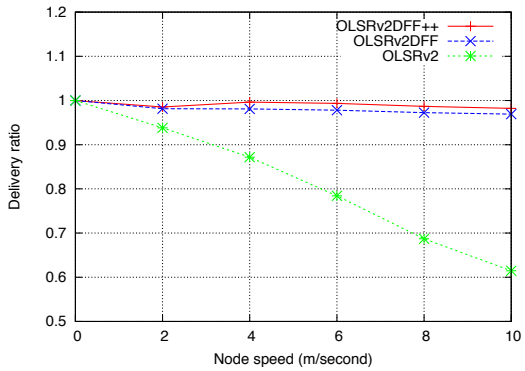

Figure 6. Average delivery ratio

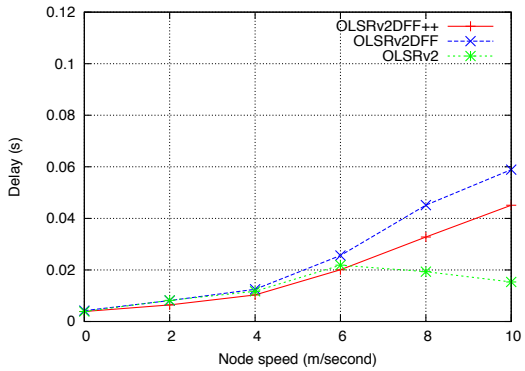

Figure 7. Average end-to-end delay

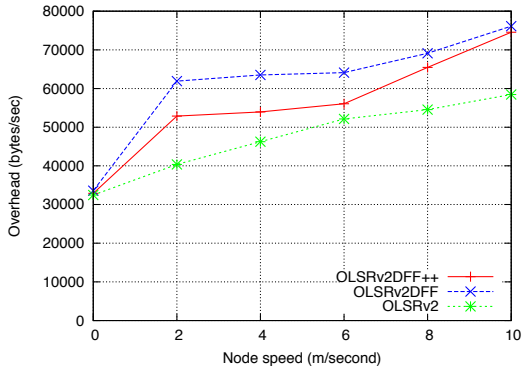

Figure 8. Average overhead

A. RPL Overview

The basic construct in RPL is a "Destination Oriented Directed Acyclic Graph" (DODAG), depicted in figure 9. In a converged LLN, each RPL router has identified a stable set of parents, each of which is a potential next-hop on a path towards the "root" of the DODAG, as well as a *preferred parent*. Each router, which is part of a DODAG (*i.e.* has selected parents) will emit *DODAG Information Object* (DIO) messages, using link-local multicast, indicating its respective *rank* in the DODAG (*i.e.* distance to the DODAG root according to some metric(s), in the simplest form hop-count). Upon having received a (number of such) DIO messages, a router will calculate its own rank such that it is greater than the rank of each of its parents, select a preferred parent and then itself start emitting DIO messages.

The DODAG formation thus starts at the DODAG root (initially, the only router which is part of a DODAG), and spreads gradually to cover the whole LLN as DIOs are received, parents and preferred parents are selected and further routers participate in the DODAG. The DODAG root also includes, in DIO messages, a *DODAG Configuration Object*, describing common configuration attributes for all RPL routers in that network – including their mode of operation, timer characteristics etc. RPL routers in a DODAG include a verbatim copy of the last received DODAG Configuration Object in their DIO messages, permitting also such configuration parameters propagating through the network.

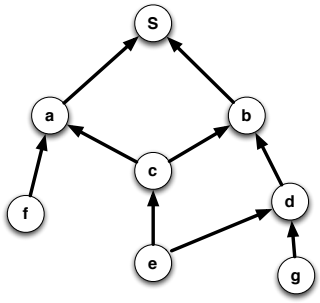

Figure 9. RPL Basic Construct: DODAGs

The DODAG so constructed is used for installing upward routes: the "preferred parent" of an RPL router can serve as a default route towards the root, and "parents" (if exist) are also kept as backup routes. Thus, RPL by way of DIO generation provides "upward routes" or "multipoint-to-point routes" from the sensors inside the LLN and towards the root.

"Downward routes" are enabled by having sensors issue *Destination Advertisement Object* (DAO) messages, propagating as unicast via parents towards the DODAG root. These describe which prefixes belong to, and can be reached via, which RPL router. In a network, all RPL routers must operate in either of storing-mode or non-storing-mode, specified by way of a "Mode of Operation" (MOP) flag in the DODAG Configuration Object from the root. Depending on the MOP, DAO messages are forwarded differently towards the root:

- In *non-storing-mode*, an RPL router originates DAO messages, advertising one or more of its parents, and unicast it to the DODAG root. Once the root has received DAOs from an RPL router, and from all routers on the path between it and the root, it can use source routing for reaching advertised destinations inside the LLN.
- In *storing-mode*, each RPL router on the path between the originator of a DAO and the root records a route to the prefixes advertised in the DAO, as well as the next-hop towards these (the router, from which the DAO was received), then forwards the DAO to its preferred parent.

"Point-to-point routes", for communication between devices inside the LLN and where neither of the communicating devices are the DODAG root, are as default supported by having the source sensor transmit via its default route to the DODAG root (*i.e.,* using the upward routes) which will then, depending on the "Mode of Operation" for the DODAG, either add a source-route to the received data for reaching the destination sensor (downward routes in non-storing-mode) or simply use hop-by-hop routing (downward routes in storing-mode). In the case of storing-mode, if the source and the destination for a point-to-point communication share a common ancestor other than the DODAG root, a downward route may be available (and used) before reaching the DODAG root.

B. RPL with DFF?

LOADng and OLSRv2 purely operate on the "control plane", *i.e.,* , they do not alter the flow of the data traffic nor the data packet headers

or content. However, RPL requires changes to the "data plane" by imposing different "traffic patterns" to data packets transmitted in the network. To perform loop detection in RPL, for each packet being forwarded, an IPv6 hop-by-hop option header is added that includes (amongst others) the following fields [24]:

- SenderRank, which records the rank of router that forwards the packets.
- Direction flag, which indicates whether the packet is to progress upward or downward.

For upward traffic, RPL provides a local route recovery mechanism: the "preferred parent" serves as default route to the root. If the link to the preferred parent is detected broken, RPL will try to forward the packet to other parents. However, RPL will never return the packet back in the opposite direction of the current packet flow. This is incompatible with DFF because DFF explicitly requires the packet to be able to be returned to the previous hops, in order to perform the depth-first search. An RPL implementation in storing mode could be modified to support this behavior by basically ignoring the direction flag and letting DFF handle forwarding and loop detection. However, this would break interoperability with existing, standard-compliant RPL implementations that do not use DFF. A standard-compliant RPL implementation would simply drop a received packet that was returned to it in case the direction flag is in the opposite direction of the current packet flow.

For non-storing downward traffic, the packet is forwarded using a source route constructed at the root. It is also impossible to perform any DFF function because only the root has enough topology information to calculate the source route. Moreover, the source route addresses in the source route header [25] are not modified per the RPL standard and per [25].

Therefore, as RPL imposes directions on the traffic flow and modifies IP headers in the "data plane", DFF cannot be applied RPL to in any of its mode of operations without breaking interoperability. This demonstrates one of the reasons why many routing protocols, such as OSPF, strictly operate on the control plane only and do not interfere with the data plane: it may severely limit the usability of extensions in the data plane.

VIII. Conclusion

DFF is a forwarding mechanism using depth-first searching for un-reliable networks. This paper studies the application of DFF in unreliable networks. The ordering issue of Candidate Next Hop List is discussed in detail, based on which, a minimal-impact optimisation to DFF, denoted DFF++ is presented. DFF++ does not impose additional signals in-the-air, and is fully interoperable with DFF. It alleviates some problems of DFF, such as repeatedly trying to forward traffic down "blind alleys" and across recently detected broken links. Performance studies comparing DFF and DFF++ alone (without a concurrently operating unicast routing protocol) have revealed the benefits of this optimisation to be significant: DFF++ attains a higher data delivery rate, shorter paths and lower data delivery delays than DFF. DFF++ attains these performance improvements without introducing new control signals, minimal additional state (a single IP address added to an existing data set) and low implementation complexity – and, remains completely interoperable with DFF, as specified in [2].

Although DFF and DFF++ can work alone by performing purely depth-first search, neither of them attempt to offer "shortest paths" – that remains under the auspices of a routing protocol. Both DFF and DFF++ are intended to operate concurrently with a unicast routing protocol. For the purpose of this study, the application of DFF (and DFF++) to three major industrial routing protocol standards for MANET and sensor networks: LOADng, OLSRv2 and RPL are discussed.

DFF and DFF++ can be applied to LOADng with additional neighbourhood discovery mechanism. Both of them can significantly improve the delivery ratio of LOADng, especially in large scale scenarios. DFF++ can offer a moderate, but consistently better, performance when compared to LOADng with DFF. For OLSRv2, because it provides already neighbourhood discovery protocol, DFF and DFF++ can be integrated without additional control traffic. The simulation results show that the packet delivery ratio can be greatly improved in dynamic scenarios. The RPL protocol is also discussed. Because of the way RPL enforces directions of data packets and modifies IP headers in the data plane, DFF is not applicable to RPL.

Acknowledgement

Part of this work was carried out as part of the SOGRID project, funded by ADEME (French agency for Environment and Energy Management) and developed in collaboration between participating academic and industrial partners.

References

[1] J. Yi, T. Clausen, and U. Herberg, "Depth first forwarding for low power and lossy networks: Application and extension," in *Internet of Things (WF-IoT), 2014 IEEE World Forum on*, March 2014, pp. 462–467.

[2] U. Herberg, A. Cardenas, T. Iwao, M. Dow, and S. Cespedes, "Depth-First Forwarding (DFF) in Unreliable Networks," RFC 6971, IETF, June 2013.

[3] S. Cespedes, A. Cardenas, and T. Iwao, "Comparison of Data Forwarding Mechanisms for AMI Networks," in *Proceedings of 2012 IEEE Innovative Smart Grid Technologies Conference (ISGT)*, January 2012.

[4] C. Perkins, E. Belding-Royer, and S. Das, "Ad hoc On-Demand Distance Vector (AODV) Routing," RFC 3561, IETF, July 2003.

[5] T. Clausen and P. Jacquet, "Optimized Link State Routing Protocol (OLSR)," RFC 3626, IETF, October 2003.

[6] T. Clausen, P. Jacquet, and L. Viennot, "Comparative Study of Routing Protocols for Mobile Ad-hoc Networks," in *Proceedings of the IFIP MedHocNet, September, Sardinia, Italy*, 2002.

[7] ——, "Analyzing Control Traffic Overhead versus Mobility and Data Traffic Activity in Mobile Ad Hoc Network Protocols," *ACM Journal on Wireless Networks, vol. 10 no. 4*, 2004.

[8] T. Clausen, C. Dearlove, P. Jacquet, and U. Herberg, "The Optimized Link State Routing Protocol version 2," RFC 7181, IETF, April 2014.

[9] K. Kim, S. D. Park, G. Montenegro, S. Yoo, and N. Kushalnagar, "6LoWPAN Ad Hoc On-Demand Distance Vector Routing," Internet Draft, work in progress, draft-daniel-6lowpan-load-adhoc-routing-03, IETF, June 2007.

[10] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, and J. Vasseur, "RPL: IPv6 Routing Protocol for Low power and Lossy Networks," RFC 6550, IETF, March 2012.

[11] G. Hiertz, S. Max, R. Zhao, D. Denteneer, and L. Berlemann, "Principles of IEEE 802.11s," in *Proceedings of WiMAN in conjunction with the 16th ICCCN*, Honolulu, Hawaii, USA, Aug 2007, p. 6.

[12] "ITU-T G.9956: Narrow-Band OFDM power line communication transceivers - Data link layer specification," November 2011.

[13] U. Herberg and T. Clausen, "A Comparative Performance Study of the Routing Protocols LOAD and RPL with Bi-Directional Traffic in Low-power and Lossy Networks (LLN)," in *Proceedings of the 8th ACM International Symposium on Performance Evaluation of Wireless Ad Hoc, Sensor, and Ubiquitous Networks (PE-WASUN)*, October 2011.

[14] T. Clausen, U. Herberg, and M. Philipp, "A Critical Evaluation of the IPv6 Routing Protocol for Low Power and Lossy Networks," in *Proceedings of the 5th IEEE International Conference on Wireless & Mobile Computing, Networking & Communication (WiMob)*, October 2011.

[15] T. Clausen, A. C. de Verdiere, J. Yi, A. Niktash, Y. Igarashi, H. Satoh, U. Herberg, C. Lavenu, T. Lys, and J. Dean, "The Lightweight On-demand Ad hoc Distance-vector Routing Protocol – Next Generation (LOADng)," Internet Draft, work in progress, draft-clausen-lln-loadng, IETF, October 2013.

[16] T. Clausen, A. Camacho, J. Yi, A. C. de Verdiere, Y. Igarashi, H. Satoh, Y. Morii, U. Herberg, and C. Lavenu, "Interoperability Report for the Lightweight On-demand Ad hoc Distance-vector Routing Protocol - Next Generation (LOADng)," Internet Draft, work in progress, draft-lavenu-lln-loadng-interoperability-report, IETF, December 2012.

[17] ITU, "ITU-T G.9903: Narrow-band orthogonal frequency division multiplexing power line communication transceivers for G3-PLC networks: Amendment 1," May 2013.

[18] K. Razazian, M. Umari, A. Kamalizad, V. Loginov, and M. Navid, "G3-plc specification for powerline communication: Overview, system simulation and field trial results," in *Power Line Communications and Its Applications (ISPLC), 2010 IEEE International Symposium on*, March 2010, pp. 313–318.

[19] I. Stojmenovic, M. Russell, and B. Vukojevic, "Depth first search and location based localized routing and qos routing in wireless networks," in *Computers and Informatics*, 2000, pp. 21–24.

[20] B. Vukojevic, N. Goel, K. Kalaichevan, A. Nayak, and I. Stojmenovic, "Power-aware depth first search based georouting in ad hoc and sensor wireless networks," in *Mobile Wireless Communications Networks, 2007 9th IFIP International Conference on*, Sept 2007, pp. 141–145.

[21] S. Dabideen and J. Garcia-Luna-Aceves, "Owl: Towards scalable routing in manets using depth-first search on demand," in *Mobile Adhoc and Sensor Systems, 2009. MASS '09. IEEE 6th International Conference on*, Oct 2009, pp. 583–592.

[22] T. Clausen, C. Dearlove, and J. Dean, "Mobile Ad Hoc Network (MANET) Neighborhood Discovery Protocol (NHDP)," RFC 6130, IETF, April 2010.

[23] T. Clausen, J. Yi, and A. C. de Verdiere, "LOADng: Towards AODV Version 2," in *VTC Fall*. IEEE, 2012, pp. 1–5.

[24] J. Hui and J. Vasseur, "The Routing Protocol for Low-Power and Lossy Networks (RPL) Option for Carrying RPL Information in Data-Plane Datagrams," RFC 6553, IETF, March 2012.

[25] J. Hui, J. Vasseur, D. Culler, and V. Manral, "An IPv6 Routing Header for Source Routes with the Routing Protocol for Low-Power and Lossy Networks (RPL)," RFC 6554, IETF, March 2012.