Implementation and analysis of the AODVv2 Routing Protocol in ARM devices

Enrica Zola, Israel Martin-Escalona, Francisco Barceló-Arroyo, Sergio Machado
Network Engineering Departament
Universitat Politècnica de Catalunya
Barcelona, Spain
enrica.zola@upc.edu, (imartin, barcelo, smachado)@entel.upc.edu

Abstract*—AODVv2 is a well-known routing protocol used in MANETs (Mobile Ad hoc Networks). Formerly known as DYMO (DYnamic MANET On-demand), it is frequently used as a reference for routing protocols assessment. However, implementations of these protocols are scarce and the few ones available are currently outdated, no longer maintained and hardly upgradeable. This paper provides the details of a new AODVv2 implementation to be used in embedded devices working with the ARM microprocessor architecture. A user-space approach has been followed so both the upgradability and platform-independence are favored. A WiFi ad hoc network, modeling representative real scenarios, has been deployed to verify the correctness of the developed AODVv2 code and assess the performance of the protocol under realistic traffic conditions. A virtual machine has been used to perform a cross-compilation of a the code that implements the DYMO protocol in the Intel x86 computer architecture. Once compiled for being used in ARM-based devices, the code has been tested in Raspberry Pi devices to verify the proper behaviour. Simple scenarios and scenarios with high density of nodes have been deployed and data have been collected and analysed.

Keywords.- Manet, Routing Protocol, DYMO, AODVv2.

I. INTRODUCTION

A. The AODVv2 protocol

AODVv2 is a pure reactive protocol used in MANETs which was formerly known as DYMO (DYnamic MANET On-demand). It is based on AODV (Ad hoc On-demand Distance Vector) and shares with it some of its benefits while being simpler to implement [1, 2, 3]. The protocol works on a hop by hop basis and nodes do not send packets unless they are performing routing or transmitting tasks. The main benefit of this protocol is a low routing overhead that leads to an improved traffic capacity and lower energy consumption.

It uses basically two packet types to discover possible routes to the destination. On the one hand, Route Request packets (RREQ) are sent in broadcast mode through the MANET, each RREQ holding a list of all the nodes passed in its way to the destination. On the other hand, Routing Reply (RREP) packets are sent to the source from either the destination or from any forwarding node that has a valid route to destination, upon reception of the RREQ. RREP packets go back to the source by the same route traversed by the RREQ packet that trigger the reply, and inform that the route has been finally established. Hence the function of the RREP message is to set up a route between destination node and source node, and all the intermediate nodes between them. In addition to these two basic packet types related to the route discovery process, another message has been defined for route maintenance [3]. This the Route Error message (RERR), which used to indicate an invalid route from any intermediate node to the destination node.

The source sets a time-out after which a new RREQ will be sent if a RREP has not been received yet. This new RREQ has a larger time-out and, after repeating this process a given number of times without success, the destination node is considered unreachable (Route Discovery Fail). Routing tables are not maintained and kept after each topology change so control packets are not sent unless a transmission is needed.

B. Motivation and goals

In [4], an implementation of AODVv2 was developed to run over the Intel x86 architecture and tested with 3 laptop computers. The test proved that all functionalities worked as expected, without evaluating the performance. The AODVv2 protocol is targeted to MANET consisting of sensors or small devices that work with limited resources of CPU and energy and are better represented by the ARM architecture.

Several modifications of the AODVv2 protocol have been presented including possible upgrades, such as information about the nodes' position proposed in the DYMOselfwd [5] to guarantee loop-free routes [7]. It has been shown through simulation [6] that AODVv2 reduces the protocol overhead, hence the energy and consumption and traffic volume, which allows to slightly speed-up the route discovery and data transfer. A benchmark is necessary in order to assess that the performance of such protocols in real scenarios agrees with the forecast obtained through theoretical analysis and simulation.

The first goal of this work is to build an implementation of AODVv2 to run over ARM devices with limited resources of energy and CPU. This will be programmed in C and must work for WiFi WiPi (Wireless Internet Platform for Interoperability) adapters. Once built the protocol and checked its functionalities, the second goal is to obtain a performance evaluation of its behaviour in real scenarios.

II. IMPLEMENTATION OF THE AODVV2 PROTOCOL

The code has been managed and developed using GNU Autotools [8] a suite of programming tools that assists to write cross-platform software and makes it easier to export the code to many Unix-like systems. Moreover, it also provides tools for cross compilation. The code has been developed in Linux and it has been compiled for both Intel x86 and 32-bit ARM architectures. The implementation of the routing protocol

978-0-7381-1316-6/21/$31.00 ©2021 IEEE

follows the AODVv2 Internet Draft version 16 [9] and it has been coded in the user space in order to simplify the development, testing, deployment and portability of the source code. The user space refers to all the code in an operating system that lives outside the kernel.

The code is executed as a system daemon and can be setup through a configuration file, which defines the values of the AODVv2 parameters. These parameters are grouped into three main categories: timers, protocol constants, and administrative parameters and controls. When the daemon starts, it launches the two main threads along with the main blocks of the code, as depicted in Figure 1. These two threads are responsible for the AODV2 operation and they remain running as long the daemon is alive.

Figure 1 shows the data-flow diagram of the AODVv2 implementation. The process is triggered when a data packet is received (Figure 1 on the right). Then, the `aodv_2_decisor_thread` handles the packed and proceeds checking if a valid route to the target prefix is available. In the case it is not, the data packet is queued using NFQUEUE as the target of the iptables route. NFQUEUE is an iptables/ip6tables target which delegates the decision on packets to a user space software. Once the data packet has been queued, the focus moves to the `aodv_v2_thread`, which is responsible for the AODVv2 message processing, and the the route discovery process begins. Thus, a RREQ message is firstly sent in order to learn a route to the destination target. The node remains then waiting for a RREP message. When it arrives (Figure 1 on the left), the `aodv_v2_thread` will forward it unless the node is the destination of the RREP; in the latter case, a valid route to the target prefix has just been discovered and it must be installed in the Kernel forwarding table (i.e. the Forwarding Information Base, FIB). As a final step, this event is signaled to the `aodv_2_decisor_thread` so that the packet, which was queued while the route discovery process was seeking a valid route, is re-injected in the *netfilter* POSTROUTING hook and eventually transmitted via the outbound network interface. The same would happen in case a valid route is already available when a data packet is received.

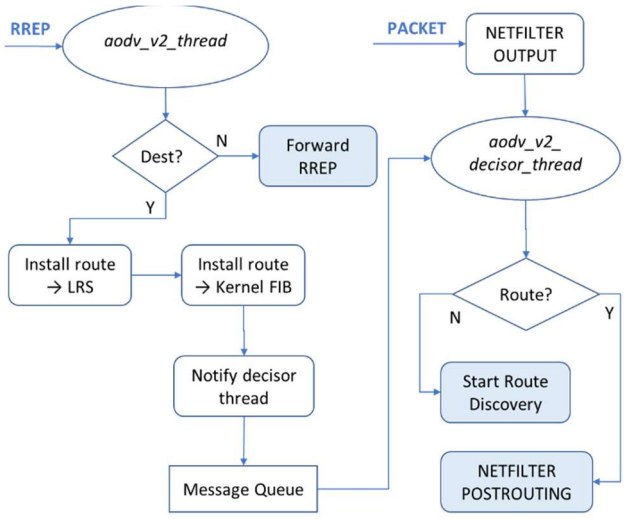

Figure 1. Main blocks of the code for ARM devices

Along with the code, a suite of unit tests have been run to ensure the correctness of the software. Furthermore, a set of use cases has been developed in order to cross validate the implementation under real conditions. This validation has been focused on three points: 1) AODVv2 message processing, 2) routing table update, and 3) timeout management. A specific procedure may involve more than one of these use cases. For instance, when a node receives a RREP as a reply to a given RREQ that was previously issued, one needs to cross check that such message is correctly processed (i.e., use case 1) and that the forwarding table is updated accordingly (i.e., use case 3).

III. Scenarios and Data Collection

A. Network topologies

Two sets of scenarios have been tested in order to asses the performance of the protocol in very simple cases. The first set is based on chain topologies with different number of nodes while the second set is a ring without and with a mesh connection of the intermediate nodes.

The first set of scenarios includes three scenarios based on the chain layout. Chain layouts with five, six and seven nodes have been studied. Figure 2 shows the layout with five nodes, the other two with six and seven nodes are exactly the same by just adding one and two intermediate nodes, respectively. The distance between neighbour nodes is set to 4 meters, modeling what could be a simple wireless sensor network.

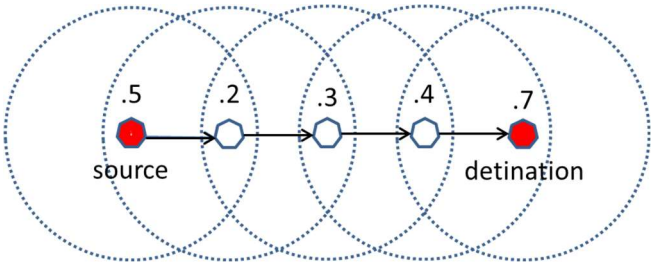

Figure 2. Chain topology with five nodes.

The data transfer is assumed to start at the source node (e.g., the node on the left) with destination to the other end (e.g., the node on the right). The IP addresses belong to the private network 10.0.3.0/24, being the source 10.0.3.5 and the destination 10.0.3.7. Notice that the coverage of each node reaches only its closest neighbour in order to avoid skipping nodes hence using actually a topology different to the one planned. To this end, the transmission power of each Wireless Network Interface Card (WNIC) is reduced to 1 mW instead of the default value of 25 mW, through the iwconfig command. Since this reduction is not enough to avoid a coverage larger than the next node in all cases (i.e., propagation conditions are variable in general due to temperature, reflexions, etc), the power is faded by enclosing each device into a Faraday cage that is built with a paperboard wrapped by metal paper. In this way, the proper attenuation is obtained in order to reach only the closest device.

The second set of scenarios consists of two diamond scenarios in order to show topologies more complex than the simple chain and to allow more than one path between source and destination. Figure 3 shows this layout with 6 nodes. The paths between the upper and lower chain in the figure have been blocked to ensure that the two paths are not communicated between them. Though according to the coverage map the upper and lower nodes do not reach among them, coverage and device sensitivity depend too much on temperature and air conditions to trust that they actually do not reach while being close to the planned coverage border. This

is why mechanical blocking (i.e., iron panels) has been used to avoid communication between nodes of the upper and lower chains.

The layout shown in Figure 3 represents in fact a ring with a maximum of two possible paths for each data transfer. In order to broaden the variety of possible paths, two new nodes have been added as shown in Figure 4. Notice that now the homogeneity of distances between neighbour nodes is no longer present while the number of hops to reach destination can change, leading to more realistic cases. The above-mentioned technique used to limit the power in order to reach the adequate coverage must be kept as in the chain topologies.

The topology in Figure 4 can be seen as the one displayed in Figure 3 after changing the iron blocking panels by two new nodes well under coverage of the upper and lower chains. In the following, we will refer to topology in Figure 3 as D1, and the one in Figure 4 as D2. Notice that in D1 the number of hops is 3 and no transmission with a different number of hops is possible while in D2 the number of hops is variable between 3 and 5.

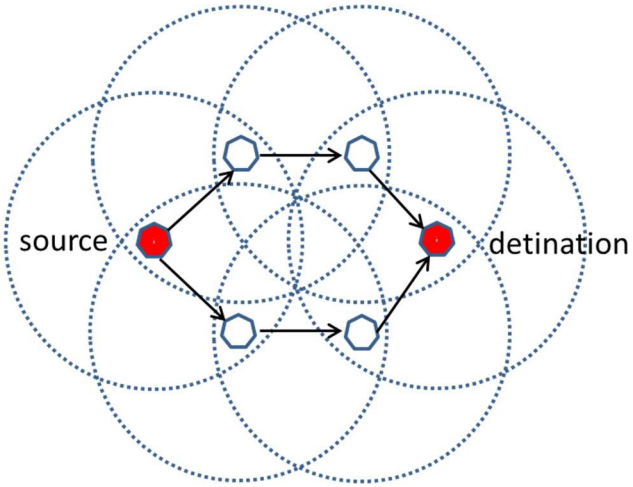

Figure 3. Diamond topology 1 (D1).

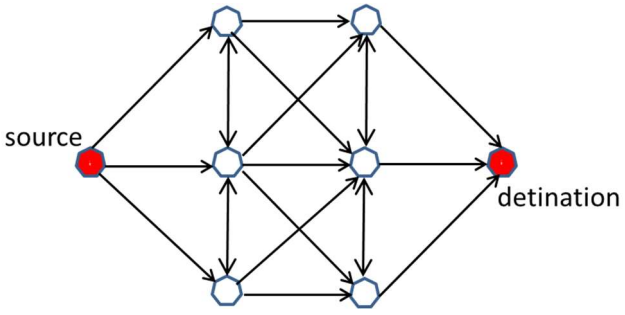

Figure 4. Diamond topology 2 (D2).

B. Metrics

In order to analyse the performance of the protocol in each specific scenario, the following metrics have been considered: latency, number of needed retries for the route discovery, and overall number of RREQ sent.

Latency. This is the time needed to discover a route between the source and destination. Specifically, it is the time involved since the source sends its first RREQ until the source receives the corresponding RREP from the destination

$$latency = t_{RX}(RREP) - t_{TX}(RREQ) \quad (1)$$

Number of needed retries for the route discovery. This is the number of times that a source starts the discovery procedure for a given destination by sending a new RREQ with increased time-out

$$num\_retries\_RREQ = \sum RREQ(timeout_i) \quad (2)$$

Overall number of RREQ sent. This metric allows to evaluate the whole overhead traffic injected into the network by the protocol and is obtained by accounting every RREQ sent by every node in the network.

C. Procedure to collect the data

In order to collect the metrics of interest, each minute the source node sends a PING to the destination, hence activating the route discovery protocol for the given destination node. Then the packets in the network must be captured with their corresponding timestamps and filtered in order to process only the packets that are needed in order to obtain the considered metrics. Wireshark [10] has been used to carry out this task. In order to obtain a sample large enough so that the confidence intervals remain within 5%, a number of PING between 300 and 500 are needed for each scenario.

As an example of the procedure, Figure 5 shows the capture obtained with Wireshark for the chain topology with 7 nodes. The IP addresses in the chain from source to destination are: 10.0.3.5 (source) → .2 → .3 → .4 → .6 → .20 → .7 (destination). Packets marked with protocol *packetbb* and with a broadcast IP as destination address (10.0.3.255) correspond to the RREQs, while the packets also belonging to the *packetbb* protocol but with an IP address different to broadcast are RREPs addressed to the next node in the return route, as explained in section 1. ICMP packets are also displayed. The timestamps associated to each packet are collected together with the packet class in order to get average values for the three metrics of interest. These timestamps are used to obtain the metrics. As an example, the timestamp corresponding to the $t_{TX}(RREQ)$ (267.692) and the one corresponding to the $t_{RX}(RREP)$ (268.184) are highlighted in red in Figure 5 and are used to obtain a measure of the latency.

The setup consists of one only source sending pings to the destination with a time interval between pings of 1 minute and the maximum number of retries set to 3.

IV. Results

A. Chain scenarios

In Figure 6 the probability density function for the latency (route discovery time) is presented for the three cases with five (orange), six (yellow) and seven (blue) nodes respectively. The peak value and the average of the latency are shorter for a smaller number of nodes, as could be expected since the number of hops is smaller, thus leading to shorter times spent in processing the packets (i.e., the impact of the shorter signal propagation time is negligible given the short distances involved). Since the access method is CSMA/CA, the amount of collisions must be also considered as another source of delay for larger amount of nodes. As the coverage reaches only the closest node, the collisions observed at each node are roughly the same independently of the number of nodes. However each node receives packets from its left and right

	Time	Source	Destination	Protocol	Length	Info
51458	267.692223	10.0.3.5	10.0.3.255	packetbb	137	
51460	267.697903	10.0.3.2	10.0.3.255	packetbb	131	[Malformed Packet]
51463	267.705379	10.0.3.3	10.0.3.255	packetbb	131	[Malformed Packet]
51464	267.709219	10.0.3.4	10.0.3.255	packetbb	131	[Malformed Packet]
51467	267.717262	10.0.3.4	10.0.3.20	packetbb	98	
51491	267.793667	10.0.3.4	10.0.3.3	packetbb	100	
51494	267.795580	10.0.3.3	10.0.3.4	packetbb	98	
51502	267.845397	10.0.3.4	10.0.3.3	packetbb	133	
51519	267.895010	10.0.3.3	10.0.3.6	packetbb	100	
51521	267.897493	10.0.3.6	10.0.3.3	packetbb	98	
51534	267.942895	10.0.3.3	10.0.3.6	packetbb	133	
51551	268.000688	10.0.3.6	10.0.3.2	packetbb	100	
51554	268.004009	10.0.3.2	10.0.3.6	packetbb	98	
51565	268.066551	10.0.3.6	10.0.3.2	packetbb	133	
51585	268.127076	10.0.3.2	10.0.3.5	packetbb	100	
51587	268.130554	10.0.3.5	10.0.3.2	packetbb	98	
51593	268.184266	10.0.3.2	10.0.3.5	packetbb	133	
51661	268.738980	10.0.3.5	10.0.3.20	ICMP	147	Echo (ping) request
51664	268.739834	10.0.3.5	10.0.3.20	ICMP	147	Echo (ping) request
51666	268.740909	10.0.3.5	10.0.3.20	ICMP	147	Echo (ping) request

Figure 5. Wireshark results for the chain scenario with 7 nodes.

neighbour, so collisions are a source of delay similar to the processing time at the node. The delays introduced by the collisions and processing time are both of random nature and its characterization is out of the scope of this work.

The three shapes are quite similar among them and close to a Gauss bell; also, the function is slightly more spread over time in the case of a higher number of nodes, which could also be predicted since more hops involved means more variability in the times spent for processing.

B. Diamond scenarios

In Figure 7 the probability density functions of the latency for the two cases corresponding to the diamond topology are displayed. The shorter latency displayed in D2 (i.e., after introducing two new nodes to D1) does not seem to match what was expected since now, contrary to what happened with the chain topology where more nodes connected involved more collisions and longer latency. Notice that with D2 topology every intermediate node is not only connected to the node on the left and on the right, but also to two other intermediate nodes. It is obvious that a noticeable increase of the collisions must take place when changing from D1 to D2 leading to expect a longer latency.

The explanation to the shorter latency in D2 must be associated to the actual implementation of the protocol that limits to three the number of RREQ looking for a route. In other words, if a route has not been discovered after three retries, no more tries are sent and the route is just considered not accessible. As shown later (see explanations to figures 8 and 9 below), in D1 the percentage of discovered routes is higher than in D2, where the higher number of collisions tend to reduce the performance. Since the latency provided in the figures is the latency of the routes discovered only and non-discovered routes are not involved in the statistics, the latency is slightly shorter while the number of routes discovered is clearly smaller in D2. Accordingly, it can be stated that AODVv2 performs better in D1 mainly due to the higher amount of collisions in D2.

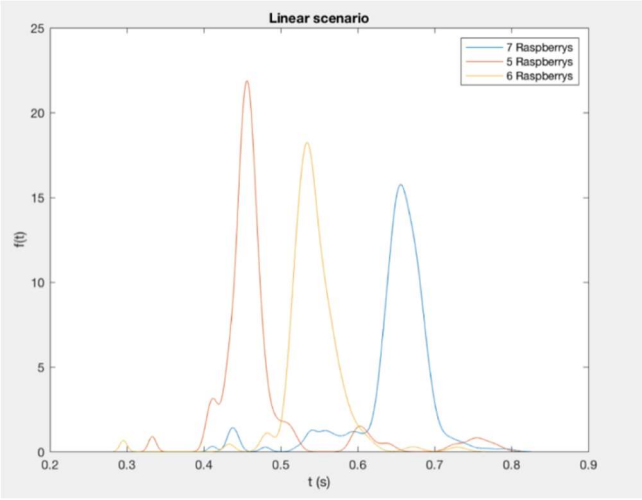

Figure 6. Latency for the chain topology with 5, 6 and 7 nodes.

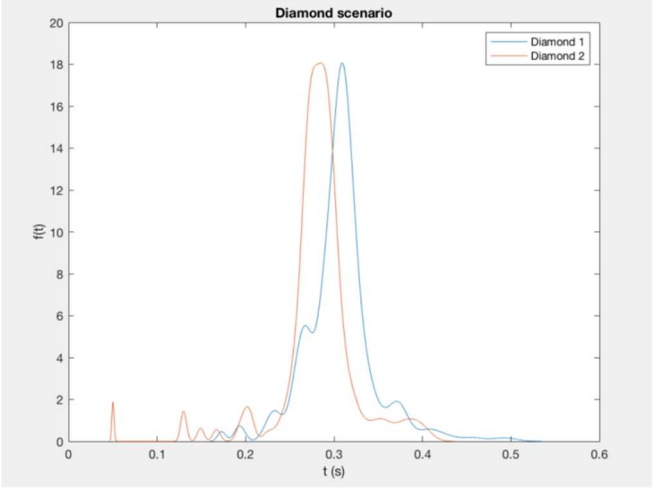

Figure 7. Latency pdf for diamond topologies 1 and 2.

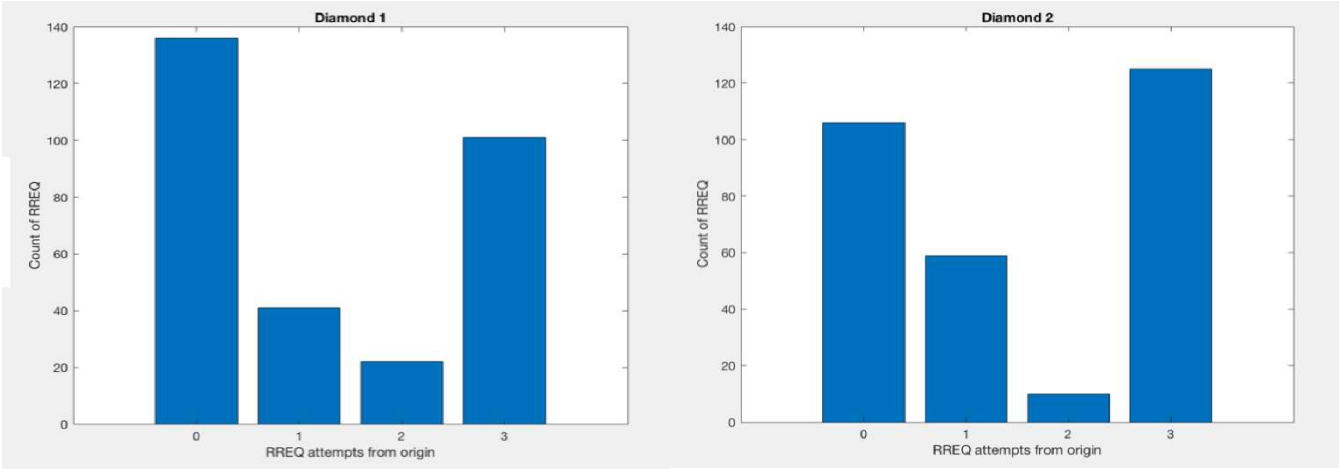

Figure 8. Histogram of the number of retries a) D1, b) D2.

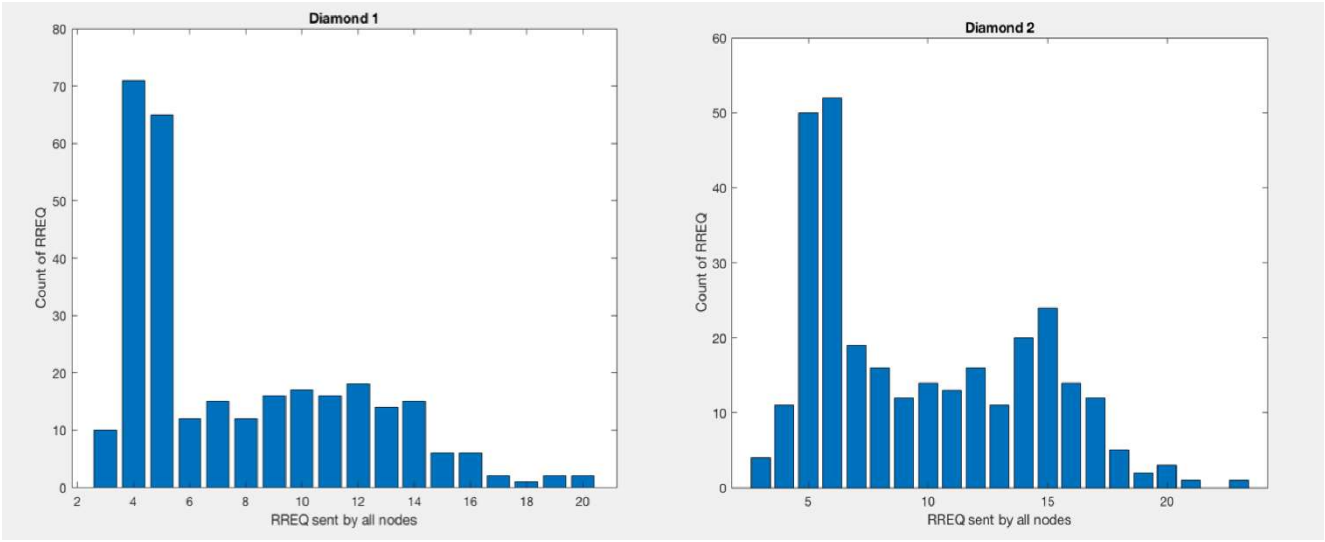

Figure 9. Histogram of the overall number of RREQs in D1 and D2.

C. *Retries*

According to the notation used in [9] "If the requested rout is not learned within the wait period, another RREQ is sent, up to total of DISCOVERY_ATTEMPTS_MAX". If the first try fails to obtain a route this procedure starts initiating a retry, which is the first discovery attempt.

Figure 8 shows the histogram of the number of retries needed to discover a route in both diamond topologies. The peak at 0 retries means that the route was discovered upon the first RREQ (i.e., no retry was needed), while the peak at 3 retries is explained by the fact that the maximum amount of retries was set to 3 in the protocol (as established by default in [9]). This last peak at 3 retries contains routes that were discovered at the third retry together with route discovery fails. It must be noted that the amount of routes discovered upon the third retry is marginal, below 1% in both topologies. Hence, the bar at 3 in Figure 8 shows in fact the figure for the number of discovery fails that was 33% for D1 and 42% for D2. The poorer performance of D2 can be associated to the fact that the number of hops is variable between 3 and 5 while in D1 the number of hops is always 3. While the mesh topology observed in D2 makes it more robust to link and node failures, the impact of more possible hops is negative on the protocol performance.

D. *Overhead*

Figure 9 shows the histogram of the overall number of RREQs sent by all nodes in the network for topologies D1 and D2 respectively, this representing the total amount of overhead traffic injected by the protocol. Notice that in D1 a RREQ can generate a maximum of 5 RREQs belonging to the source and to the four intermediate nodes. Hence the peak at 4 and 5 shows how the route is discovered upon the first try. The larger number of possible paths in D2 leads to a noticeable increase of the overall traffic together with a smaller concentration of the number of RREQ around a single value. Notice also how the tail of the distribution is related to the topology. In D1a total of 5 nodes (source plus 4 before the destination) can send a maximum of 4 RREQ each one (one first plus 3 retries) reaching a maximum of 20 RREQ in total. In D2 the maximum is much larger since there 7 nodes generating RREQ with variable number of possible hops, however, the tail decreases drastically above 20.

V. CONCLUSIONS

A practical implementation of the DYMO protocol in devices working with the ARM architecture has been developed and tested. Also a performance analysis of this protocol working in real scenarios has been presented. To the authors' knowledge, this is the first attempt at providing the research community with the code of a common routing protocol in wireless sensor networks. Moreover, it is the first time that results from a real, even though simple, testbed are presented and discussed.

For simple chain topologies, the results show how the route discovery time increases along with the number of necessary hops, and presents more dispersion as the number of hops increases since every hop introduces a time which is partially random. When the topology is more compact, with more possible paths between source and destination, the discovery time reduced. The number of retries needed for a route discovery has also been checked showing how the benefit of more possible paths does not grant a better performance since the traffic caused by the protocol increases involving more collisions, which can lead to a lower percentage of route discovery.

ACKNOWLEDGMENT

This research was supported by the Spanish Government and ERDF through CICYT project PGC2018-099945-B-I00.

The authors would like to thank Carlos Valdés Pérez and Rocco Clemente for their valuable work during their BSc final thesis.

REFERENCES

[1] A.K. Gupta, H.Sadawarti, A.K. Verma, Implementation of DYMO routing protocol, International Journal of Information Technology, Modeling and Computing (IJITMC) Vol.1, No.2, May 2013, pp. 49-57.

[2] A. Hinds, M. Ngulube, S. Zhu, H. Al.Aqrabi, "A Review of Routing Protocols for Mobile Ad-Hoc NETworks (MANET)",International Journal of Information and Education Technology, Vol. 3, No. 1, February 2013.

[3] V. Rao, A.K. Gupta, "A Comprehensive Study of AODVv2-02 Routing Protocol in MANET", International Journal of Computer Applications (0975 – 8887) International Conference on Advancements in Engineering and Technology (ICAET 2015), March 2015.

[4] C. Valdés. "Implementación del protocolo de encaminamiento AODVv2 en Linux". UPC internal report, TFG-EETAC, October 2017

[5] E. Zola, F. Barceló-Arroyo, I. Martín-Escalona. "DYMO Self-Forwarding: A Simple Way for Reducing the Routing Overhead in MANETs", Mobile Information Systems, Volume 2017 Article ID 6265021, July 2017.

[6] K.S. Namjoshi, R.J. Trefler "Loop Freedom in AODVv2". In: Graf S., Viswanathan M. (eds) Formal Techniques for Distributed Objects, Components, and Systems. FORTE 2015. Lecture Notes in Computer Science, vol 9039. Springer, Cham. 2015.

[7] E. Zola, F.Barcelo-Arroyo, I. Martin-Escalona. A Modification of DYMO RoutingProtocol with Knowledge of Nodes' Position: Proposal and Evaluation. 13th International Confer-ence on Wired/ Wireless Internet Communication (WWIC), May 2015, Malaga, Spain. pp.289-298

[8] https://www.gnu.org/software/automake/faq/autotools-faq.html, Autotools FAQ

[9] Draft-ietf-manet-aodvv2-16] C. Perkins, S. Ratliff, J. Dowdell, L. Steenbrink V. Mercieca, "Ad Hoc On demand Distance Vector Version 2 (AODVv2) Routing", May 2016, <https://tools.ietf.org/html/draft-ietf-manet-aodvv2-16>

[10] Wireshark. https://es.wikipedia.org/wiki/Wireshark