Verification of Symbolic Distributed Protocols for Networked Embedded Devices

Andrea Augello[†], Rosolino D'Antoni[*], Salvatore Gaglio[†‡], Giuseppe Lo Re[†], Gloria Martorella[†], and Daniele Peri[†]
[*] rosolino.dantoni@community.unipa.it, [†]{andrea.augello01, salvatore.gaglio, giuseppe.lore, gloria.martorella, daniele.peri}@unipa.it
[*†‡] Engineering Department, University of Palermo, Viale delle Scienze, Ed. 6, 90128 Palermo, Italy
[‡]ICAR-CNR, 90146 Palermo, Italy

***Abstract*—The availability of versatile and interconnected embedded devices makes it possible to build low-cost networks with a large number of nodes running even complex applications and protocols in a distributed manner. Common tools used for modeling and verification, such as simulators, present some limitations as application correctness is checked off-board and only focuses on source code. Execution in the real network is thus excluded from the early stages of design and verification. In this paper, a system for modeling and verification of symbolic distributed protocols running on embedded devices is introduced. The underlying methodology is rooted in a symbolic programming paradigm that makes it possible to model protocols with a high level of abstraction still permitting their execution on resource-constrained devices. The preliminary experimental results shown in this paper concern verification of a distributed averaging protocol in a simulated network at increasing number of nodes. The results support the feasibility of the approach to test distributed applications running on large networks of resource-constrained nodes.**

I. Introduction

Performance evaluation of distributed protocols [1] is a daunting task, especially considering the limited software and hardware resources available to embedded systems. In the relative abundance of methods and tools [2], network simulators are used to evaluate distributed protocols [3], mostly. Simulators differentiate according to a series of simulation parameters [4]. Despite the abundance of these tools [5], [6], [7], simulators fail to reproduce all the physical and natural phenomena of a real network [8]. Network emulators try to offset simulators shortcomings by executing the required code directly on real hardware and through the use of appropriate software interfaces [9], [10]. However, emulators do not scale easily as their application is bound to specific hardware. Adequate level of realism can only be achieved through the use of testbenches [11], [12]. These consist of a set of physical nodes distributed in well-defined areas, usually, accessible through a web-based interface supporting only preestablished and controlled scenarios. Several studies have analyzed the performance of distributed protocols starting from the performance evaluation of WSNs through specialized tools [13], [14], [15]. These studies evidenced that important limits of network simulators lie in their source-code focus and in checking protocol correctness only off-board. Hybrid verification tools instead include at least a real node in simulations. Nevertheless, these tools suffer from timing problems between the real part and the virtual one, and this error increases with the number of nodes in the network [16].

This work introduces an approach supporting the verification of distributed protocols during their execution on actual hardware. The approach is based on a rule-based system that contains the formal description of the code that carries out certain operations of a protocol and the related code for verification. The system is, therefore, able to send code to the nodes that can verify the correctness of the protocol, which is to say that the results obtained conform with the expected ones.

This paper shows the assessment of the feasibility of the approach through the simulated verification of a distributed average protocol. In detail, Section II describes the adopted methodology, and outlines each element of the proposed system. Section III describes the modeling and verification strategies. The case study is presented in Section IV and the corresponding experimental evaluation in Section V. Section VI concludes the paper discussing improvement areas and future development plans.

II. Approach and proposed system

The methodology discussed in this paper is based on the symbolic execution environment DC4CD [17] running on network nodes. Code is a concatenation of executable symbols, called words. The symbolic execution environment DC4CD is based on the Forth language and has been used to turn on-board hardware specifications into automatic verification code for resource-constrained subsystems [18]. Words are contained in a dictionary and new words can be easily added to this dictionary by including their definition, based on previously defined words. The REPL execution mechanism of the environment interpreter waits for a list of words executing one word after another [19]. Parameters and return values are passed between words through a stack.

For example, the program that allows a node to measure the temperature can be simply run by executing the word

```
temperature
```

which leaves the measured temperature value on the stack.
Distributed processing is implemented by letting nodes exchange symbolic code.
To this purpose, the symbolic environment provides that the sequence of words included between the words `tell:` and

`:tell` is sent to the remote node specified by the physical address left on the stack. This simple but powerful construct carries out the exchange of executable symbolic code and it fully supports the implementation of distributed protocols. When the symbolic placeholder *tilde* (`~`) is executed, it is replaced with the value placed on the top of the stack. For example, to tell a remote node to measure the temperature, send the response back, and make the requesting node print its value, the requesting node executes:

```
tell: temperature reply tell: ~. :tell :tell
```

where the word *dot* (`.`) pops the topmost value of the stack and prints it. Exchanged messages do not need to have a fixed structure since symbolic code is included as the payload of datalink level packets. Since the code is sent as a sequence of symbols, the nodes can exchange data, symbolic rules, and even algorithms.

The proposed system is composed of two main parts: the rules system (RS) running on the host computer and the symbolic platform running on the nodes (Fig. 1).

The rules system generates the requests including the symbolic verification code. The requests are transmitted to the bridge node, the one connected to the host computer, and forwarded by the latter to the specified remote nodes which run the symbolic platform and execute the protocol along with its verification code.

As shown in Fig. 1, communication between the rules system and the bridge node is achieved through a USB serial interface, while communication between remote nodes and bridge node occurs through radio links. Once the verification results are sent back to the bridge node in the form of executable code, their execution leaves the verification values on the stack of the bridge. Finally, results can be picked up and analyzed by the rules system. Verification code is sent as simple text directly to each node and executed without intermediate translation steps. Symbolic text code abstracts the characteristics and the representation of target hardware extending its scope to a broad range of devices, improving interoperability, unlike network emulators.

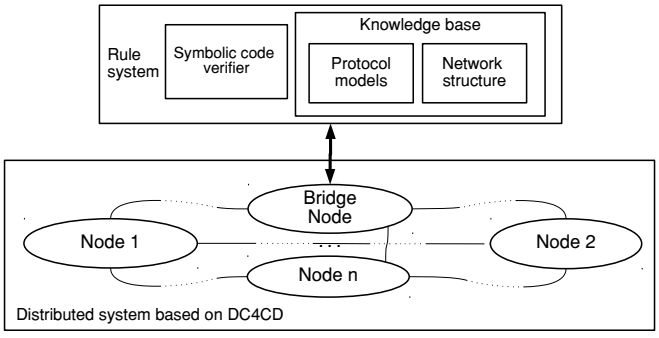

Fig. 1: Block diagram of the proposed system.

III. Modeling and verification

The Knowledge Base (KB) in the RS contains rules defining primitives for communication protocols (Fig. 1). Some of these model messages and transmission, including acknowledgments, between the network nodes. Other primitives manage recovery situations starting from possible errors that occur during communication. Routes, storage of incoming messages, and transmission times are also modeled. The KB also models the distribution within the network and their location using a two-dimensional Cartesian coordinate system as in the following definitions:

```
node(a, [2.4,3.1]). % node a at (2.4,3.1)
node(b, [4.1,1.0]). % node b at (4.1,1.0)
```

The RS also keeps track of which nodes are reliable so to base verification on their trustworthy replies.

While the primitives are modeled into the RS, the executable steps of a protocol specification are stored directly on the nodes as symbolic programs. Loading of the executable symbols defining the protocol under test (PUT) is done in the network initialization phase. The RS starts the PUT by sending the initiating code to the network through the bridge node.

The key aspect of verification is the association between the symbolic code of high-level operations and the symbolic code that verifies its execution on remote nodes. During the protocol execution and at proper time, the symbolic verification code is automatically generated by the rules system and sent to the network nodes.
To this purpose, in the KB are defined rules of this type:

```
verific_code(Label, ValueList,
VerificCode, RightMsg, WrongMsg)
```

in which `Label` specifies the operation to be verified, `ValueList` indicates possible return values expected for that operation, and `VerificCode` indicates the verification code that must be transmitted by the rules system, and executed by the network nodes to retrieve the desired results. The last two parameters indicate the message which correspond to a successful and to a failed verification for that specific operation. Verification of sensor network protocols concerns the proper collection of real or simulated readings of physical quantities but also the correct execution of the communication schemes as specified in the KB. Due to the flexibility of the symbolic platform, the same symbol may be defined differently on nodes. This way different behaviors can be modeled on a node-by-node basis for the same operation leading to a broader range of test configurations.

An example of a physical quantity that can be measured on nodes is temperature. The temperature value can be actually measured on a node through sensors or simulated on the node based on a specific model useful for the test (eg. expected trends). For communication schemes, an example is the verification of acknowledgment reception (ACK). This mechanism is modeled by returning a predefined numerical value (eg. 99). Unlike the temperature calculation, in this case the word definition for acknowledgment transmission and reception changes depending on whether the node is the bridge or not:

- for the bridge, the ACK definition is

```
: ack 99 ;
```

since the node in question must simply put the value on the stack.

- for a remote node the ACK definition is

```
: ack reply [tell:] 99 [:tell] ;
```

since the node in question must reply to the bridge with the ACK value.

IV. Case Study

As a case study we analyzed a simple protocol for distributed aggregation of physical quantities, instantiating it to collect temperature data (Fig. 2). Briefly,

1) the RS makes the bridge node execute the symbolic code: `bcst tell: 0 0 update :tell` The `bcst` keyword specifies that the message be broadcast to the network;
2) each receiving node sets the content of two of its local variables to 0. The first variable (`num`) stores the number of nodes that already carried out the temperature measurement. The second (`aggr`) stores the current aggregate temperature value, i.e. the sum of the measurements communicated by the nodes so far;
3) each node waits for a time proportional to its physical ID. When this time expires, the node performs the temperature measurement and updates the values of its local variables. The node also updates the value of another local variable (`avg`) containing the average value of the temperatures measured up to this point. Then the node executes the symbolic code `bcst tell: <num> <aggr> update :tell`, broadcasts the updated values of the variables `num` and `aggr`. The receiving nodes then execute the code updating the values of the two corresponding local variables as in the first step;
4) the protocol ends when all the network nodes have performed step 3. This happens when the time corresponding to the maximum ID expires.

The proposed system allows many verification schemes, in this work two are shown.

In the first scheme, a single request to retrieve the values of the three local variables is transmitted to one of the reliable nodes (Fig. 3). The verification here consists in checking that the number of received values is equal to three and that the value of the `num` variable equals the number of nodes involved in the communication. This verification is carried out at the end of the protocol execution.

In the second scheme, at the end of the protocol execution, all the nodes are requested to send the values of the three local variables. Then, a control ensures that each response generated exactly three values. After this, it is verified that all the triplets of retrieved values are equal and that the value of the `num` variable equals to the number of nodes involved in the communication.

V. Experimental Evaluation

Even though the proposed system is meant to perform hybrid tests on heterogeneous networks made of real hardware

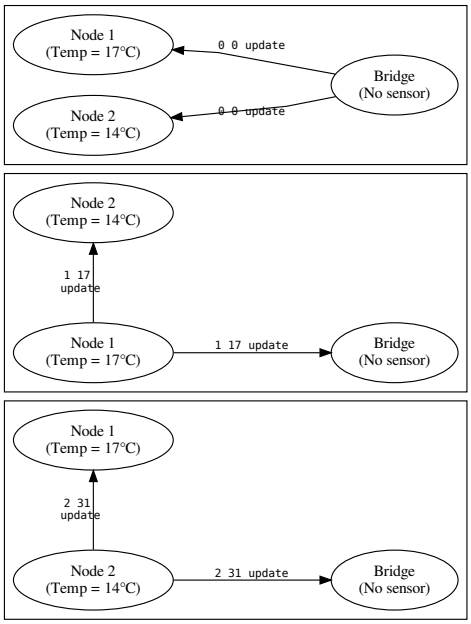

Fig. 2: Code sent by nodes during the execution of the distributed averaging protocol.

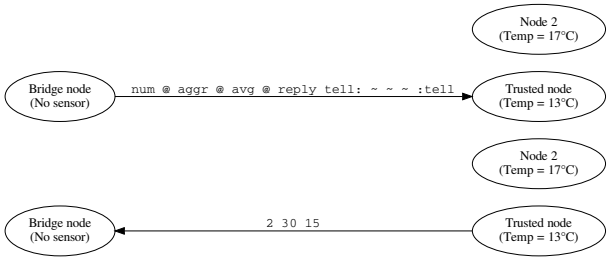

Fig. 3: Messages exchanged during trusted node verification.

nodes, in this preliminary work we tested our approach by performing several simulated experiments measuring the number of exchanged messages and the average required time.

The exchanged messages during the protocol execution and verification only refer to that messages needed to verify the protocol itself, while the time required value includes execution of the distributed averaging protocol.

The verification mode that interrogates only a reliable remote node is the most efficient, but it is necessary to be sure that the reliable node gives the right results. The second verification mode provides the assurance that the protocol has been performed correctly but requires a longer verification time and a higher number of exchanged messages. To test the scalability of the proposed verification tool, experiments were performed by simulating nodes in the RS, modeling waiting times as a fixed delay specified for each node in

the knowledge base, calculated accounting for the serial line baud rate, the IEEE 802.15.4 bandwidth and the processing time of the interpreter on the actual nodes. The PUT assumes that all the nodes are 1-hop neighbors, so the propagation delay is considered negligible. A subset of the simulated nodes (10%) was defined as unreliable. Queries to unreliable nodes would always return incorrect values. For both "Reliable" and "All nodes" schemes five repetitions of the protocol verification were carried out randomly simulating a network with increasing number of nodes. For each repetition, fifteen runs of verification were carried out, measuring the number of exchanged messages and the average required time. Since the protocol execution time depends on the node IDs, they were randomly generated with a different seed on each run.

Table I collects all the results of the tests. The exchanged messaged for the "All nodes" scheme were twice the number of nodes minus one as we include the message that starts the protocol execution in the count but the bridge does not take part on the computation. The time for verification grows linearly with the number of nodes as nodes are queried sequentially, so, before querying the next node, the RS must wait the reply from the previous one and read it from the serial interface.

Results show that verification time is comparable to protocol execution, hinting at an effective application of the tool to the test of distributed applications.

Nodes	Reliable node		All nodes	
	Time required (s)	Exchanged messages	Time required (s)	Exchanged messages
10	163.64	3	257.77	19
20	173.87	3	386.18	39
50	176.39	3	743.97	99
100	177.77	3	1337.41	199

TABLE I: Results of the verification of the distributed averaging protocol at increasing network size.

VI. Conclusions

In this paper a system for modeling and verification of symbolic distributed protocols running on embedded devices was introduced. The underlying methodology is rooted in a symbolic programming paradigm that makes it possible to model protocols with a high level of abstraction still permitting their execution on resource-constrained devices.

The preliminary experimental results shown in this paper concern verification of a distributed averaging protocol in a simulated network at increasing number of nodes. The results support the feasibility of the approach to test distributed applications running on large networks of resource-constrained nodes.

Future work will consider testing the protocols during their execution, rather than only at the end, to detect failure earlier and avoid wasting resources, running experiments on physical nodes alongside virtualized ones. Finally, more complex protocols, like CTP, will be modeled and tested to assess the scalability and efficiency of the system.

References

[1] S. Park, A. Savvides, and M. B. Srivastava, "SensorSim: A simulation framework for sensor networks," in *Proceedings of the 3rd ACM international workshop on Modeling, analysis and simulation of wireless and mobile systems*, 2000, pp. 104–111.

[2] M. Imran, A. M. Said, and H. Hasbullah, "A survey of simulators, emulators and testbeds for wireless sensor networks," in *2010 International Symposium on Information Technology*, vol. 2. IEEE, 2010, pp. 897–902.

[3] A. Nayyar and R. Singh, "A comprehensive review of simulation tools for wireless sensor networks (WSNs)," *Journal of Wireless Networking and Communications*, vol. 5, no. 1, pp. 19–47, 2015.

[4] E. Weingartner, H. Vom Lehn, and K. Wehrle, "A performance comparison of recent network simulators," in *2009 IEEE International Conference on Communications*. IEEE, 2009, pp. 1–5.

[5] B. Musznicki and P. Zwierzykowski, "Survey of simulators for wireless sensor networks," *International Journal of Grid and Distributed Computing*, vol. 5, no. 3, pp. 23–50, 2012.

[6] M. Jevtić, N. Zogović, and G. Dimić, "Evaluation of wireless sensor network simulators," in *Proceedings of the 17th telecommunications forum (TELFOR 2009), Belgrade, Serbia*. Citeseer, 2009, pp. 1303–1306.

[7] H. Sundani, H. Li, V. Devabhaktuni, M. Alam, and P. Bhattacharya, "Wireless sensor network simulators a survey and comparisons," *International Journal of Computer Networks*, vol. 2, no. 5, pp. 249–265, 2011.

[8] A. El-Mouaffak and A. E. B. El Alaoui, "Considering the environment's characteristics in wireless networks simulations and emulations: Case of popular simulators and WSN," in *Proceedings of the 3rd International Conference on Networking, Information Systems & Security*, ser. NISS2020. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3386723.3387821

[9] H. Wu, Q. Luo, P. Zheng, and L. M. Ni, "VMNet: Realistic emulation of wireless sensor networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 2, pp. 277–288, 2007.

[10] P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: Accurate and scalable simulation of entire tinyos applications," in *Proceedings of the 1st international conference on Embedded networked sensor systems*, 2003, pp. 126–137.

[11] J. Albesa, R. Casas, M. T. Penella, and M. Gasulla, "Realnet: An environmental wsn testbed," in *2007 International Conference on Sensor Technologies and Applications (SENSORCOMM 2007)*. IEEE, 2007, pp. 502–507.

[12] L. P. Steyn and G. P. Hancke, "A survey of wireless sensor network testbeds," in *IEEE Africon '11*, 2011, pp. 1–6.

[13] U. M. Colesanti, C. Crociani, and A. Vitaletti, "On the accuracy of omnet++ in the wireless sensornetworks domain: simulation vs. testbed," in *Proceedings of the 4th ACM workshop on Performance evaluation of wireless ad hoc, sensor, and ubiquitous networks*, 2007, pp. 25–31.

[14] X. Xian, W. Shi, and H. Huang, "Comparison of OMNET++ and other simulator for WSN simulation," in *2008 3rd IEEE Conference on Industrial Electronics and Applications*. IEEE, 2008, pp. 1439–1443.

[15] P. Nayak, "Comparison of routing protocols in WSN using netsim simulator: LEACH vs LEACH-C," *International Journal of Computer Applications*, vol. 106, no. 11, 2014.

[16] S. Saginbekov and C. Shakenov, "Testing wireless sensor networks with hybrid simulators," *arXiv preprint arXiv:1602.01567*, 2016.

[17] S. Gaglio, G. Lo Re, G. Martorella, and D. Peri, "DC4CD: A Platform for Distributed Computing on Constrained Devices," *ACM Transactions on Embedded Computing Systems*, vol. 17, no. 1, Dec. 2017. [Online]. Available: https://doi.org/10.1145/3105923

[18] ——, "WSN Design and Verification Using On-Board Executable Specifications," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 2, pp. 710–718, 2019.

[19] ——, "A Lightweight Middleware Platform for Distributed Computing on Wireless Sensor Networks," *Procedia Computer Science*, vol. 32, no. 0, pp. 908 – 913, 2014, the 5th International Conference on Ambient Systems, Networks and Technologies (ANT-2014), the 4th International Conference on Sustainable Energy Information Technology (SEIT-2014). [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050914007108