Poster Abstract - WSN-Erlang: a Functional, High Level Approach to WSN Development

Alessandro Sivieri
Dipartimento di Elettronica e Informazione
Politecnico di Milano, Italy
sivieri@elet.polimi.it

Gianpaolo Cugola
Dipartimento di Elettronica e Informazione
Politecnico di Milano, Italy
cugola@elet.polimi.it

***Abstract*—The complexity in designing, coding, and testing WSN applications is considered among the most relevant factors that limit the diffusion of WSN technology. In this work we claim that Erlang, a concurrency and distribution-oriented, functional language, could greatly reduce this complexity. Moving from this premise we introduce a new platform, named WSN-Erlang, which leverages the Erlang peculiarities adapting them to the requirements of WSNs. WSN-Erlang also includes a testing and simulation framework, fully supporting the entire life-cycle of WSN applications.**

I. Introduction

Most of the researchers operating in the WSN area agree that what limits the diffusion of WSNs and pervasive systems in general is the complexity in designing, programming, testing, and deploying real scale applications based on such technologies. In particular, the research community is still debating about the best programming paradigm to use [1].

Indeed, the peculiarities of the hardware platform introduce several new challenges: WSN applications are executed on a distributed environment composed of heterogeneous devices, with different capabilities and limited resources, interconnected using short and unreliable wireless links.

The research community has proposed two main paradigms to tackle these issues: *micro-programming* gives the programmer the responsibility of decomposing the problem and developing code for each of the network nodes, comprising the low level hardware and protocol details; this approach does not try to hide the complexity of WSNs, it gives its burden on the developers' shoulders. *Macro-programming* allows developers to access the network as a single entity, which can be programmed using abstract primitives, while the underlying framework takes care of the low level and communication details; this approach reduces the complexity of development, but it loses generality with respect to the previous one, so that many of the frameworks developed under this paradigm can be applied to a limited number of similar scenarios. Besides these differences, many of the approaches analyzed in [1] have never been tested in real world scenarios and their apparent applicability remains limited. At the same time, the main development process adopted for WSN applications is the "code and fix" one [2], which relies on the developers' skills to overcome the limitations of the platform and the difficulties in programming it, with limited attention to the issues of reusability and maintainability.

To address this situation, we need a new language and programming framework, and we claim that Erlang [3] could represent a good starting point: it is a high-level, functional programming language, which supports facilities like:

- lightweight concurrency using the actor model [4];
- distributed programming, with high-level communication primitives;
- transparent resolution of process names over the network;
- handling of heterogeneity through the use of a virtual machine;
- pattern matching on bit streams;
- support for fault-tolerant applications.

Many of these characteristics are tipically found in WSN applications, or can be quite useful for this area. Moreover, the combination of the actor-model of concurrency with high-level communication facilities, and the use of a virtual machine, ease the development of simulation tools for testing applications before deployment.

II. The framework

WSN-Erlang is a new programming framework and run-time system adapting Erlang to WSN requirements. Indeed, the standard Erlang platform, even if originally developed for embedded systems (telecommunication devices), has grown overtime and nowadays contains a huge number of accessory libraries and facilities, most of which make no sense in WSNs. Because of this, we had to strip the run-time from all those parts that are not useful for our target applications. This reduced the memory, storage, and processing requirements of the platform, which as of today runs on an embedded device having an ARM926EJ-S CPU with 64 MB of RAM and 64 MB of flash, while we are currently porting it to a smaller device with a RT3050 CPU with 32 MB of RAM and 8 MB of flash. This is enough to show the advantages of our platform in terms of expressiveness and ease of use. Next step will be to enter more deeply into the virtual machine, to further reduce its requirements.

On the other hand, reducing the memory, storage, and processing requirements of the virtual machine is only a pre-requisite to use Erlang in WSNs. The most important part of our work was to adapt the language and library to the peculiarities of the platform. In particular, Erlang assumes

the availability of a full fledged TCP/IP stack, offering reliable message passing among nodes, independently from their physical location. This is a strong assumption for most WSN scenarios, which we relaxed in WSN-Erlang to assume the only availability of a (unreliable) link-layer protocol. In a wireless network this means that a message may reach its destination only if the sender and the receiver are in range. Accordingly, WSN-Erlang changes the semantics of message passing to reflect this assumption. A similar change applies to the spawning of new processes among different nodes, which is unreliable and only happens if the two nodes are in range. At the same time, we leverage the availability of a shared medium to offer a broadcast version of the communication and process spawning facilities.

All these changes not only adapt the distribution model of Erlang to the peculiarities of WSNs (whose nodes rarely implement a full fledged TCP/IP stack), but it also represent the key pre-requisite to remove all the IP-based mechanisms from the standard Erlang interpreter, our next step to further reduce the WSN-Erlang requirements.

As pointed out previously, testing and simulation are an area not well supported by existing WSN frameworks; by leveraging the Erlang peculiarity of blurring the distinction between centralized and distributed applications, which are both organized as a set of concurrent processes that communicate with each other, we were able to integrate a WSN simulator in WSN-Erlang, capable of handling a completely simulated network of Erlang processes, running the very same code that runs on real nodes. In particular, the communication between nodes is simulated using the same channel and propagation models used by *TinyOS* [5]; two modes can be employed:

- complete simulation of the network, where each physical node is virtualized and executed inside a single computer; the developer can also monitor each node and inject messages in the network;
- mixed simulation of the network, where part of the nodes is virtualized and part is executed inside the target devices.

This approach allows to start debugging a WSN application in a fully simulated deployment, to move afterwards toward a mixed deployment with only one node running on the target device and the others being simulated, to conclude with a mixed deployment in which few physical nodes interact with the simulated ones. All with the guarantee that the code that is being tested coincides, line by line, with the code that will build the final application.

III. Evaluation

We tested our prototype by implementing several algorithms that realize typical WSN activities, like collecting data from sensors or broadcasting information to the whole network. We compared them with the same algorithms developed using common WSN platforms, i.e., *TinyOS* and *Contiki*.

Results of our experiments show improvements at the source code level, in terms of readability with respect to the other implementations (which use different dialects of C). Among the features we benefit were bit sequence parsing and pattern matching on bit groups, which increased readability and code compactness of the main networking operations. Overall, the WSN-Erlang versions of the various algorithms gain one order of magnitude in terms of number of lines of code, with respect to the other versions (see Table I).

TABLE I
Algorithms LOC

Algorithm	TinyOS	Contiki	WSN-Erlang
Opportunistic flooder	495	187	100
Trickle	219	194	61
Collection algorithm	2169	1470	303

Reusability is also increased through the modularization facilities that WSN-Erlang inherits from Erlang. In particular, the usage of multiple processes inside each node can be leveraged for separating the different aspects, which can be more easily reused inside other applications, also improving the readability of the code.

IV. Conclusions and future works

WSN-Erlang addresses two limitations of currently available platforms to develop WSN applications: the lack of adequate, high-level programming abstractions to easily write reusable, maintainable code, and the difficulty in developing and testing code that may run on heterogeneous networks, with good support for debugging. Preliminary testing shows that this approach gives good results and improves the overall process of writing WSN applications.

Our current prototype cannot support the most resource constrained WSN scenarios, but at the same time WSN-Erlang runs smoothly on the kind of devices that can be found in several WSN scenarios, like health-care, where the need of saving resources is balanced by a need of reliability and fault-tolerance. Moreover, we consider the current WSN-Erlang prototype only as a first step to demonstrate the advantages of the Erlang programming model when applied to WSNs. In the future we plan to continue our work on the WSN-Erlang run-time to further reduce its requirements, putting our hands more deeply into the virtual-machine, if required.

Acknowledgments

This work was partially supported by the European Commission, Programme IDEAS-ERC, Project 227977-SMScom.

References

[1] L. Mottola and G. P. Picco, "Programming wireless sensor networks: Fundamental concepts and state of the art," *ACM Comput. Surv.*, vol. 43, pp. 19:1–19:51, 2011.

[2] G. P. Picco, "Software engineering and wireless sensor networks: happy marriage or consensual divorce?" in *Proceedings of the FSE/SDP workshop on Future of software engineering research*, 2010.

[3] J. Armstrong, *Programming Erlang: Software for a Concurrent World*. Pragmatic Bookshelf, 2007.

[4] C. Hewitt, P. Bishop, and R. Steiger, "A universal modular actor formalism for artificial intelligence," in *Proceedings of the 3rd international joint conference on Artificial intelligence*, 1973.

[5] 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.