Embedded Systems for Prototyping Underwater Acoustic Networks: the DESERT Underwater Libraries on board the PandaBoard and NetDCU

Ivano Calabrese[‡], Riccardo Masiero[‡*], Paolo Casari[‡*],
Lorenzo Vangelista[*], Michele Zorzi[‡*]

[*]DEI, Department of Information Engineering, University of Padova, via Gradenigo 6/B - 35131, Padova, Italy
[‡]CFR, Consorzio Ferrara Ricerche, via Saragat 1 - 44122, Ferrara, Italy
E-mail: {name.surname}@dei.unipd.it

***Abstract*—In this paper, we consider underwater network prototyping using the network simulation engine NS-Miracle, and investigate different embedded computer boards that can be employed for this task. In particular, we consider two embedded platforms with considerably different capabilities: the PandaBoard (a powerful platform that does not require any cross-compilation effort) and version 5.2 of the NetDCU board, which is much more constrained in terms of computational power, RAM and storage space. After describing the steps required to install NS-Miracle and the DESERT Underwater libraries on board these platforms, we report on the field experiments conducted to test the corresponding prototypes.**

Our results include a comparison between the two investigated platforms in terms of resources required (e.g., memory occupancy and energy expenditure) and performance in the execution of real-time software (e.g., slops introduced within the simulation framework). We believe that our work represents an interesting step towards the realization of underwater network prototypes made of heterogeneous nodes.

***Index Terms*—Underwater prototypes, Embedded systems, NS-Miracle, DESERT Underwater, PandaBoard, NetDCU 5.2.**

I. Introduction

In the past few years, advances in robotics, acoustic modems and control provided most of the key enabling technologies for underwater applications. In this perspective, some of the activities the underwater research community is focusing on seek a complete solution for the network architecture and the communications protocols that are required to tele-operate underwater devices. When pursuing this goal, developers need to easily simulate and prototype their protocol solutions, as well as to share the obtained results and to allow others to repeat the same experiments.

Recently, DESERT Underwater [1] has been proposed and released as a flexible, publicly accessible tool for the design and performance evaluation of underwater network protocols. DESERT Underwater is a set of public C/C++ libraries [2] for supporting the implementation of underwater network protocols, by extending the NS-Miracle [3] simulation engine which, in turn, is based on the well-known network simulator ns2 [4].

The DESERT Underwater libraries implement several protocols for underwater networks, as well as a self-contained module designed to interface NS-Miracle with real modem hardware, thus enabling the realization of actual network prototypes. In this work, we discuss about the feasibility of portable, realistic testbed systems by using small platforms to replace actual computers running the NS-Miracle engine to control the modem hardware. In detail, we focus our study on two platforms for mobile software development:

- the **PandaBoard** [5] (dual core 1 GHz ARM processor, 1 GB of RAM, SD/MMC card cage with support for High-Speed and High-Capacity SD cards) which is based on OMAP technology developed by Texas Instruments;
- the **NetDCU 5.2** [6] (CPU Intel XScale PXA255 processor of 400 MHz, 64 MB of RAM, no SD/MMC card cage).

The PandaBoard is a powerful platform almost alike a normal computer: for example, no cross-compilation is required to prepare executables to be run on it. The latter, instead, is an old version of the NetDCU board developed by F&S Elektronik Systeme GmbH, which makes it possible to test our approach on a very constrained platform.

The paper is organized as follows. In the next section we present the main technical features of the PandaBoard and of the NetDCU 5.2. The necessary steps to install ns2, NS-Miracle and the DESERT Underwater libraries on these platforms are detailed in Section III. In Section IV we describe the real-world experiments conducted to test network prototyping efforts based on these platforms, and report the corresponding results in Section V. Finally, Section VI concludes the paper.

II. Main features of the PandaBoard and NetDCU 5.2

The use of embedded devices is currently quite common in telecommunications and in many other applications relying on information technology. This tendency is correlated with the technical improvements that embedded systems have gained during the last past years (more powerful processors, smaller form factors, lower power consumption, reduced costs).

The installation of one or more programmable processor within a system leads to more flexibility compared to a fully hardwired electronic system, and it may also translate into reduced time for system design and prototyping. In addition, it is easy to network several embedded systems to make them operating together as part of a larger and scalable system. This is why, for example, the latest car models have about twenty micro-controllers [7] performing functions such as anti-lock breaking system, fuel management, air-condition management or GPS navigator. Also in underwater vehicles (such as AUVs or ROVs[1]) and underwater acoustic modems, embedded platforms can play a key role to, e.g., control the movement of the vehicle or perform the modulation and demodulation of the acoustic signal to be transmitted.

In this work, we consider two different embedded devices for building prototypes of underwater acoustic networks made of heterogeneous nodes: a PandaBoard and a NetDCU 5.2, see Figure 1. Here, we define a network node as the ensemble of an underwater modem, an acoustic transducer and the embedded platform which commands the modem through the DESERT Underwater framework. Accordingly, we define the network as heterogeneous if its nodes differ in at least one of these three components. For the experiments described in Section IV, we use two identical modems commanded by different embedded platforms. The main technical features of these platforms are detailed in the following.

A. PandaBoard

In this work, we used the first version of the PandaBoard platform family [5]. This device is a low-power, low-cost single-board computer development platform based on the Texas Instruments OMAP4430 [8], a powerful system-on-chip (SoC) featuring a very good tradeoff between power consumption and computational performance. Its processor automatically balances operations across four main engines: 1) a programmable multimedia engine based on the TI's C64x DSP and power-efficient, multi-format hardware accelerators; 2) a general-purpose processor based on the dual-core ARM Cortex A9 MPCore architecture, that supports symmetric multiprocessing (SMP) and is capable of speeds of more than 1 GHz per core; 3) a high-performance programmable graphics engine and 4) an Image Signal Processor (ISP) for video and imaging. The PandaBoard has 1 GB of low-power DDR2 RAM and can fully support High-Speed and High-Capacity SD flash cards. It supports Ethernet 10/100 Mbit for wired connections, and both the 802.11 b/g/n and Bluetooth standards for wireless communications. The considered device has a size of 114.3 mm (height) $\times$ 101.6 mm (width), with an overall weight of 74 grams.

B. NetDCU 5.2

The NetDCU 5.2, manufactured by F&S Elektronik Systeme GmbH, dates back to October 2004. This board is based on an Intel XScale CPU and on an additional Silicon Motion graphic-processor SM501; its processor is a 400-MHz PXA255. The board featured a Flash-EPROM of 64 MBytes and a RAM equal in size. It supports both Ethernet 10/100 Mbit and a RS232 Serial interface. The considered device has a size of 100 mm (height) $\times$ 80 mm (width), and an overall weight of 60 grams. Even if nowadays this platform is not the top gamma in the embedded system field, it used to be a groundbreaking device, widely used and well suited to the aim of this work.

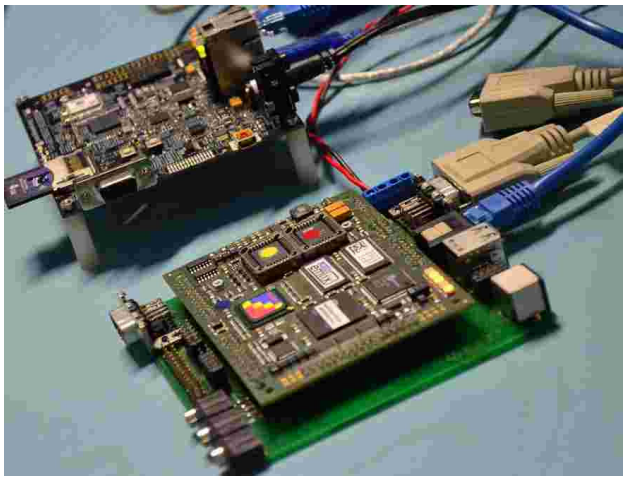

Fig. 1. The embedded platforms considered in this study: the PandaBoard (top) and the NetDCU 5.2 (bottom).

III. Installation of ns2, NS-Miracle and DESERT Underwater on board the PandaBoard and NetDCU 5.2

We will now provide an overview of the steps required to install the ns2 network simulator, its extension NS-Miracle and the DESERT Underwater libraries[2] on board both the PandaBoard and NetDCU 5.2. These tools will allow us to build our underwater network prototype.

As outlined in the the previous section, the two embedded platforms that we consider are very different in terms of technical specifications. In particular, the PandaBoard is a powerful device that can be seen almost as a normal computer (i.e., it can run a fully functional operative system such as an Ubuntu Linux distribution): consequently, the procedure to install ns2, NS-Miracle and DESERT Underwater is the same as for any personal computer. Namely, once the necessary source files have been downloaded from the corresponding websites, one needs to:

1) install ns2 by running the corresponding installation procedure;
2) install NS-Miracle using Autotools [9];
3) install DESERT Underwater using Autotools;
4) update the `.bashrc` configuration file for including the paths where libraries have been installed, according

[1]An Autonomous Underwater Vehicle (AUV) is a robot which travels underwater without requiring input from an operator, as opposed to a Remotely Operated Vehicle (ROV) which is typically tethered.

[2]The interested reader can find all the technical details of the installation procedures followed for both the PandaBoard and the NetDCU 5.2 at [2].

to the indications received during the execution of the previous points.

According to the above procedure, we installed all libraries with minimum effort, using up an overall memory area (ROM) of 142 MBytes (since the PandaBoard supports memory expansion through SD cards, this amount is much less than the typical SD card capacity).

Conversely, the NetDCU 5.2 supports only a basic operating system: therefore, the required executables can only be prepared through cross-compilation. Moreover, the total amount of Flash-EPROM size on this device is limited to 64 MByte part of which should be left for other purposes. Therefore, the above steps cannot be straightforwardly followed, but each of them requires a particular attention and some "work-arounds." Specifically:

1) we install ns2 in two steps: first, we prepare the executables files for the *Tcl* interpreter tools (i.e., the scripting language required by ns2 for parameter settings); subsequently, we polish the ns2 libraries by stripping off all applications that are not necessary for our purposes (e.g., the graphical libraries). Since we are cross-compiling, we also need to explicitly specify the right compiler for the ARM architecture in use.[3] At this point, we encounter the main "work-around" of the entire procedure. As a matter of fact, during the normal installation of ns2, we generate four binary files (i.e., `otclsh`, `tcl2c++`, `tclsh8.4` and `ptypes2tcl`) that are necessary for the compilation and installation of the overall framework: when we cross-compile, we have to pay attention to have such libraries compiled for the host that is performing the cross-compilation and not for the targeted ARM architecture (as it would happen if we straightforwardly proceed with the compilation of ns2 for this latter).
2) install NS-Miracle using Autotools and explicitly specify the right compiler for the ARM architecture in use;
3) install DESERT Underwater using Autotools and explicitly specify the right compiler for the ARM architecture in use[4];
4) update the `profile` configuration file as for the `.bashrc` of the PandaBoard.

After the above procedure, all the necessary executables and libraries for the NetDCU 5.2 are successfully installed and fit in an overall memory area (Flash-EPROM) of 23 MBytes. By using the `strip` command[5] on these files, we have been able to further reduce the required memory space for their storage to 12 MBytes; the drawback of this operation is that the RAM usage increases slightly, as shown in Section V.

[3]With the PandaBoard, this issue is handled directly by the PandaBoard's Operating Systems through symbolic links, and is therefore transparent to the user.

[4]In this case we also have to do some minor modifications to the source files of DESERT Underwater v1.0.0, that will be included in the next release.

[5]`strip` is a program that helps increase performance and reduce the overall disk space usage by removing unnecessary content (e.g., debug information) from both object and executable files.

IV. Laboratory and Real-World Tests: simple examples of heterogeneous networks

In this section, we illustrate and discuss the experiments that have been conducted at the Department of Information Engineering of the University of Padova to test the feasibility of underwater testbeds built using the PandaBoard and NetDCU devices presented in Section II. According to the proposed approach, these platforms have been exploited to replace actual computers running the NS-Miracle engine to control the modem hardware. In this perspective, we are interested in monitoring the memory resources required to run the framework and the energy expenditure of each platform. To this end, we have written a piece of software called *infoRAM*[6] that runs simultaneously with the network simulation process (`ns`), and records the percentage of RAM and CPU resources required by `ns` during any performed experiment. This evaluation is interesting, as the network simulator ns2, its extension NS-Miracle and the additional DESERT Underwater libraries were originally designed and implemented for more powerful computers. In addition, we have conducted some further investigation on the delays introduced within the simulation framework by each embedded platform, see Section V-B.

A. Protoype description

In our tests, we deployed two nodes configured as follows:

- a BTech BT-2RCL transducer [10];
- an FSK WHOI Micro-Modem [11], configured for the 25 kHz frequency range and working at a data rate of 80 bps;
- a PandaBoard or a NetDCU 5.2 running the DESERT Underwater libraries to control the modem.

We have conducted both real-world [2] and laboratory tests. During the first ones, we have placed the acoustic transducers 8 m apart in the Piovego channel, which flows nearby our department; in more detail, we put them at a distance of 2 m from the bank, where the channel is 80 to 90 cm deep, with a muddy bottom. For the laboratory tests, instead, we have placed the transducers in a tank or we directly connected the embedded platforms to the WHOI Micro-Modem Software Development Board, which contains two modems (see Figure 2). In all cases, we powered the Micro-Modems at 12 V, in order to decrease their transmission ranges (when powered at 36 V, the FSK WHOI Micro-Modems can reach a working range of up to 2 km horizontally and up to 9 km vertically [12]).

B. Test description

Overall, we conducted eight real-world tests (as well as some preliminary tests performed in laboratory) in order to assess the feasibility of prototyping underwater heterogeneous network according to our approach. An overview of these tests is reported in Table I: they realize all a point to point communication between two heterogeneous nodes identified, respectively, with the PandaBoard and the NetDCU 5.2. In detail, using the engine of NS-Miracle and the DESERT

[6]The source code of *infoRAM* can be freely downloaded at [2].

Fig. 2. Hardware prepared for a laboratory test to be conducted with the WHOI Micro-Modem Software Development Board (the black box on the upper right corner of the picture).

Underwater libraries running on board the two embedded platforms, each node has been fit out with a simple but complete protocol stack (i.e., a User Data Protocol, UDP, for the transport layer, a simple static routing protocol for the network layer, and a Media Access Control, MAC, protocol for the data link layer). At the transmitter, each data packet is generated by a CBR application, namely a traffic generator that injects packets in the network with a constant time period or according to a Poisson process with given mean. In tests 1 to 4, both the PandaBoard and the NetDCU act as transmitter or receiver, alternatively (simplex communication); in tests 5 to 8, instead, we make the two nodes transmit simultaneously according to a random pattern (duplex communications). We run the experiments by changing the CBR period (i.e., the period between the generation of two subsequent data packets) as reported in table I and, for the duplex communications, we also tested the performance of two different MAC protocols: the ALOHA [13] and the CSMA 1-persistent [14] protocols.

V. Results of the Prototype Tests

In this section we report the results of the tests described in Section IV: we first discuss the real-world experiments and the difference observed between the PandaBoard and the NetDCU 5.2 in terms of resource usage; then, we compare the impact of the two platforms on the event dispatcher of ns2.

TABLE I
Overview of the experimental setups

TEST	PANDABOARD	NETDCU 5.2
1	role: *Transmitter* CBR period: *5 s* MAC: *ALOHA*	role: *Receiver* CBR period: *5 s* MAC: *ALOHA*
2	role: *Receiver* CBR period: *5 s* MAC: *ALOHA*	role: *Transmitter* CBR period: *5 s* MAC: *ALOHA*
3	role: *Transmitter* CBR period: *10 s* MAC: *ALOHA*	role: *Receiver* CBR period: *10 s* MAC: *ALOHA*
4	role: *Receiver* CBR period: *10 s* MAC: *ALOHA*	role: *Transmitter* CBR period: *10 s* MAC: *ALOHA*
5	role: Transmitter/Receiver CBR period: *5 s* MAC: *ALOHA*	role: *Transmitter/Receiver* CBR period: *5 s* MAC: *ALOHA*
6	role: *Transmitter/Receiver* CBR period: *10 s* MAC: *ALOHA*	role: *Transmitter/Receiver* CBR period: *10 s* MAC: *ALOHA*
7	role: *Transmitter/Receiver* CBR period: *5 s* MAC: *CSMA 1-persistent*	role: *Transmitter/Receiver* CBR period: 5 s MAC: *CSMA 1-persistent*
8	role: *Transmitter/Receiver* CBR period: *10 s* MAC: *CSMA 1-persistent*	role: *Transmitter/Receiver* CBR period: *10 s* MAC: *CSMA 1-persistent*

A. Feasibility of the Proposed Approach and Resource Usage of the Embedded Platforms

Table II reports the observed packet error rate (PER) during the tests of Table I conducted in the Piovego channel. We can observe that, despite the adverse conditions (very shallow water, wind-generated surface ripples and noise, proximity to the bank, water turbidity), we have been able to successfully transmit data in both the simplex and the duplex configurations. This result allowed us to verify the feasibility of our approach. Furthermore, exploiting the modularity of the NS-Miracle framework, and therefore of the DESERT Underwater libraries, we have been able to effectively compare (with only minor modifications[7]) two different solutions for the MAC layers: the ALOHA and the CSMA 1-persistent protocol. In fact, when the generation rate of the data traffic is higher (i.e., a data packet generated on average each 5 s), we observe that the PER is lower when 1-persistent CSMA access scheme is applied (see test 7) with respect to the pure ALOHA mechanism (see test 1); differently, when the traffic generation rate is low (i.e., one packet generated on average each 10 s), the two MAC protocols perform equally (see tests 6 and 8).

As previously mentioned, during the real-world tests we have been monitoring the RAM and CPU usage in the PandaBoard and in the NetDCU. We note that these metrics also depend on the chosen protocol stack and on the performed experiment: more complex protocols and scenarios likely result in an higher CPU and RAM requirements. The investigation of the resource usage in these cases is left as a future activity.

[7]As a matter of fact, we just needed to specify the actual module to use for the MAC layer in one single line of the Tcl script written for the simulation parameter setting.

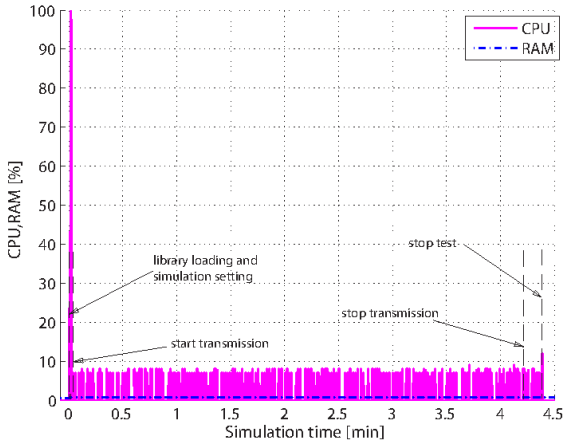

Fig. 3. Percentage of RAM and CPU usage of the PandaBoard (transmitter) in test 1 for the `ns` process. 50 data packets transmitted, on average one packet every 5 s.

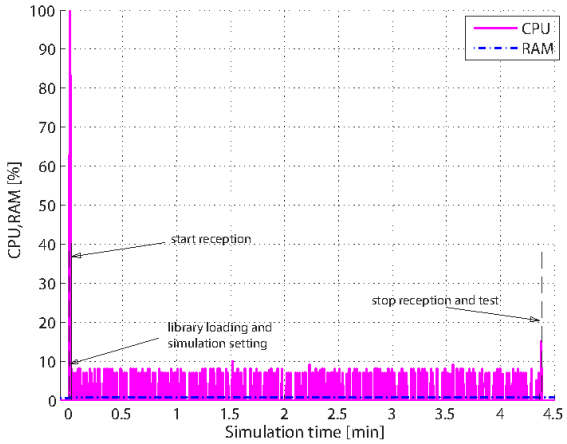

Fig. 4. Percentage of RAM and CPU usage of the PandaBoard (receiver) in test 2 for the `ns` process. 50 data packets transmitted, on average one packet every 5 s.

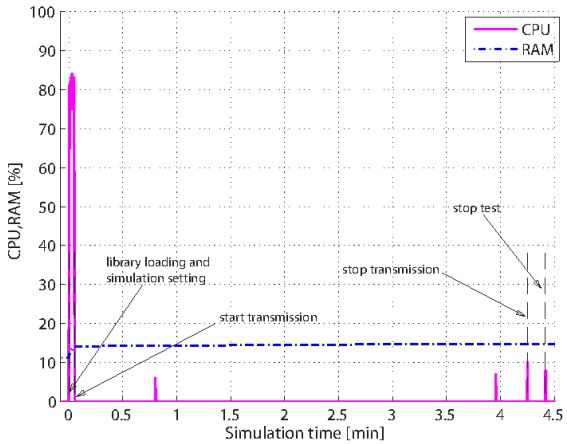

Fig. 5. Percentage of RAM and CPU usage of the NetDCU 5.2 (transmitter) in test 2 for the `ns` process. 50 data packets transmitted, on average one packet every 5 s.

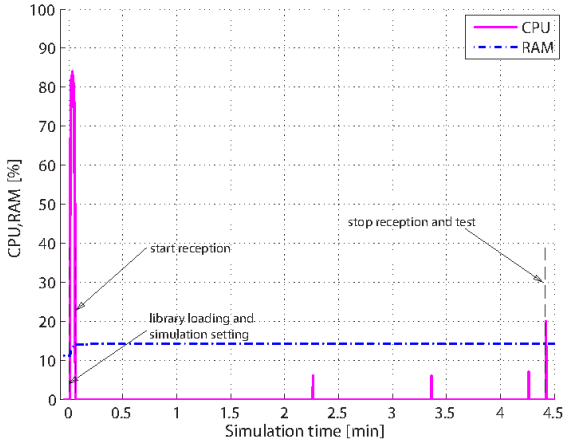

Fig. 6. Percentage of RAM and CPU usage of the NetDCU 5.2 (receiver) in test 1 for the `ns` process. 50 data packets transmitted, on average one packet every 5 s.

TABLE II
PACKET ERROR RATE (PER) OBSERVED DURING THE EXPERIMENTS IN THE PIOVEGO CHANNEL

Simplex Communication			
TEST 1	TEST 2	TEST 3	TEST 4
0.14	0.0278	0	0.028

Duplex Communication			
TEST 5	TEST 6	TEST 7	TEST 8
0.53	0.04	0.3061	0.04

In Figures 3-6 we show the percentage of used RAM and CPU for the PandaBoard and the NetDCU 5.2 during tests 1 and 2. In test 1, the PandaBoard was the transmitter and the NetDCU 5.2 the receiver, vice-versa in test 2. From the graphs we observe that the role of the nodes does not strongly affect the overall resource usage of the two boards.

For both the RAM and CPU we observe a usage peak at the beginning of the test, namely, when the simulation scripts must be interpreted and the libraries are loaded: after this, the RAM usage is almost constant for the whole duration of the tests; concerning the CPU, instead, we observe an oscillating behavior for the PandaBoard (between 0 and 8% of the total CPU) and sparse spikes for the NetDCU 5.2. We believe that this is due to the different process management performed by the dispatcher and/or the CPU scheduler[8] of the OS in either of the devices. The percentage of used RAM, instead, is clearly lower for the PandaBoard ($\sim$ 1%) than for the NetDCU 5.2 ($\sim$ 14%). Counterintuitively, this is not only due to the different amount of RAM available on each board. In fact, consider Figures 7-10, which show the RAM usage

[8]The CPU or short-term scheduler is the OS engine that selects among the active processes the one to be executed next; the dispatcher, instead, is the module that actually gives control of the CPU to such selected process.

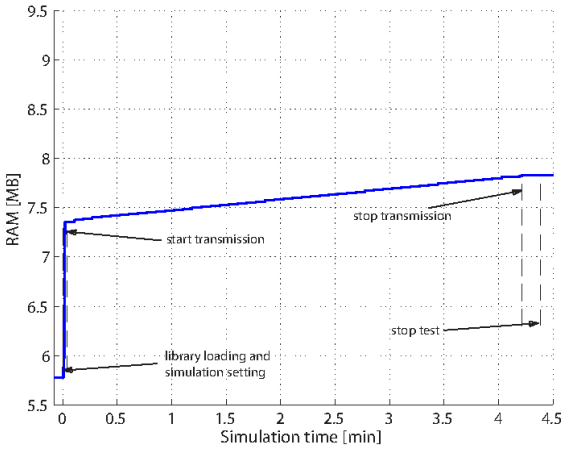

Fig. 7. RAM usage (in MBytes) of the PandaBoard (transmitter) in test 1 for the `ns` process. 50 data packets transmitted, on average one packet every 5 s.

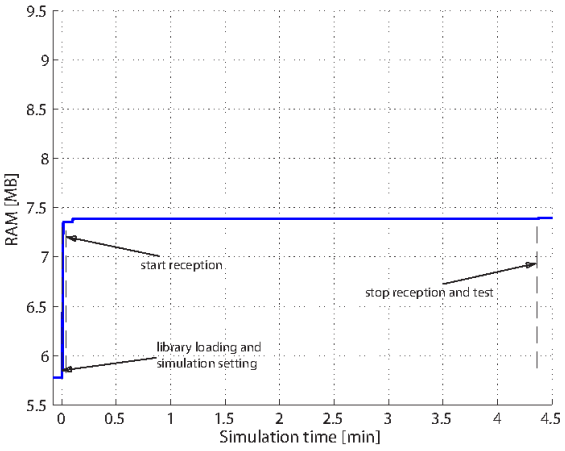

Fig. 8. RAM usage (in MBytes) of the PandaBoard (receiver) in test 2 for the `ns` process. 50 data packets transmitted, on average one packet every 5 s.

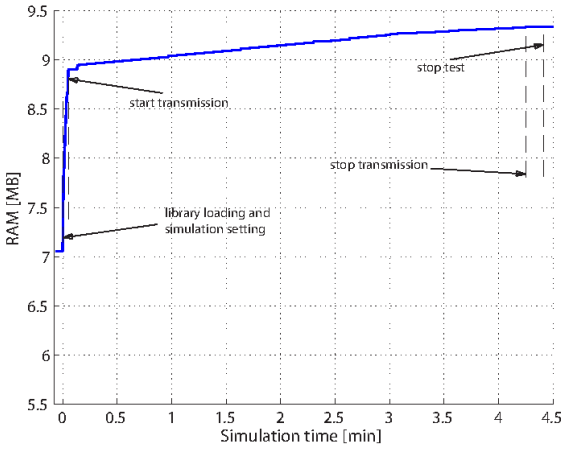

Fig. 9. RAM usage (in MBytes) of the NetDCU 5.2 (transmitter) in test 2 for the `ns` process. 50 data packets transmitted, on average one packet every 5 s.

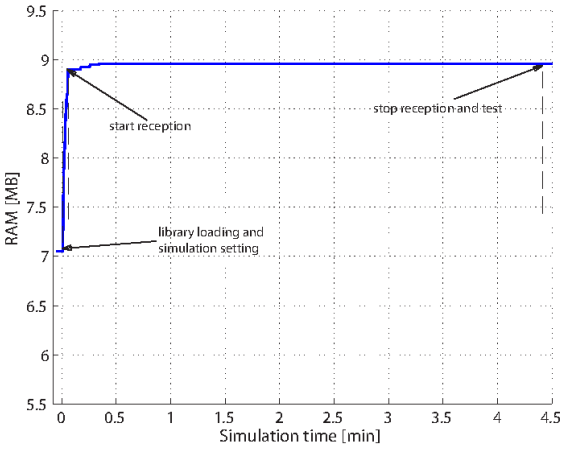

Fig. 10. RAM usage (in MBytes) of the NetDCU 5.2 (receiver) in test 1 for the `ns` process. 50 data packets transmitted, on average one packet every 5 s.

in MBytes for the two platforms in tests 1 and 2. Here the step of the beginning of simulation is even clearer, but what we can observe is that the mean RAM usage along the two tests is slightly lower for the PandaBoard (around 7.5 MByte) than for the NetDCU 5.2 (around 9 MByte). As previously mentioned, we believe that this difference is due to the fact that in the NetDCU 5.2 we compress the executable files and libraries with the `strip` command requiring then more RAM to balance the effect of such operation. Another important observation is that, the RAM usage displays an increasing trend when a board is used as a transmitter. This effect is shown by both boards and is caused by the dynamic memory allocation to the data structures of the packets: regardless of when this memory is released within the code, the OS will actually free it in bundles, in a manner that is transparent to the running processes.

B. Impact of the embedded platforms on the real-time scheduler of ns2

In this section we show the results of the additional laboratory tests that we carried out to study the impact of the considered embedded platforms in terms of the possible delays introduced within ns2. We recall that ns2 is an event-driven network simulator: therefore, it is difficult to guarantee the execution of scheduled operations in real time as, instead, it would be desirable when we interface ns2 with real hardware to realize actual prototypes. To this end, ns2 implements a soft real-time scheduler which ties the execution of the simulator events with the actual time, by regularly checking the evolving simulation time against the system clock. From a practical perspective, the difference between the evolving simulation time and the corresponding system clock should be less than a constant factor called “slop factor”, that can be set by the user.

If sufficient CPU resources are allocated to the ns2 process (as it would generally happen with normal computers), the virtual time of the simulator can closely follow the real-time evolution and the simulation will run without problem; otherwise, ns2 outputs a warning message to notify the user that a given event has not been executed at the scheduled time, i.e., the difference between the simulation time and the system clock exceeded the slop factor.

TABLE III
SIMULATION EVENTS EXECUTED WITH DELAYS ON THE PANDABOARD AND ON THE NETDCU 5.2 (WITH A SLOP FACTOR OF 0.001 S).

PANDABOARD			
# of simulated events	# of delayed event executions	mean delay	delay covariance
494	387	0.3269	0.1295
2349	2338	0.1179	0.0019
20340	20276	0.0142	2.027e-06

NETDCU 5.2			
# of simulated events	# of delayed event executions	mean delay	delay covariance
494	10	0.0069	0.0235
410739	189	4.7366e-06	4.0958e-04

In Table III we compare the performance of the real-time scheduler of ns2 on the PandaBoard and on the NetDCU 5.2. We fixed the slop factor to 1 ms and varied the number of events that the network simulator must handle (e.g., by increasing the number of packet to sent or scheduling several times the reading of a given data structure) and we measured both the number of warning messages printed by ns2 (that correspond to the number of events whose execution is delayed more than the maximum slop set by the user) as well as the mean and the covariance of the event execution delays with respect to the corresponding scheduled times. Unexpectedly, from this data we can observe how the PandaBoard has not been able to guarantee the execution at the scheduled time of almost all the simulator events (the number of printed warning messages, and therefore of delayed event executions, is always very close to the number of simulated events), even if the corresponding mean delay is low compared to the typical timing of the application (from 0.014 s to a maximum of 0.33 s when the packet data transmission rate is of one every 5 s). Conversely, the NetDCU 5.2 board guaranteed the execution of almost all simulation events in time, with a percentage of delayed events below 2% and with a smaller average delay (less than 0.024 s). Such performance is related with the CPU behavior already observed in Figures 3-6 and with the way in which the OS of each platform allocates resources to the active processes (i.e., if system resources are allocated more often to the `ns` process, the real-time scheduler of ns2 can better track the system clock). A possible solution to improve the PandaBoard performance is the adoption of a Real-time Kernel instead of the one [15] used in the pre-installed Ubuntu distribution for the ARM/OMAP architecture [16], and also used for this comparison. This activity is also left as a future extension.

VI. CONCLUSIONS

In this paper we exploited embedded platforms running the NS-Miracle network simulator for prototyping underwater networks. In detail, we tested the feasibility of commanding real modems by running the DESERT Underwater libraries on board of diverse mobile platforms instead of on actual computers. After describing the mean technical features of the considered embedded devices (the PandaBoard and the NetDCU 5.2), we reviewed the necessary steps to install ns2, NS-Miracle and the DESERT Underwater libraries on board the considered hardware and discussed the different approach needed for each board.

Finally, following the proposed approach, we built a prototype of heterogeneous underwater network and compared the two investigated platforms by means of both laboratory and real-world tests. We measured the memory occupancy, energy expenditure and delays introduced within the adopted prototyping framework. These tests allowed us to asses the feasibility of the proposed approach and to understand how to improve it in the perspective of realizing more complex and reliable underwater network prototypes.

VII. ACKNOWLEDGMENTS

This work has been supported by the multi-national four-year project "Robust Acoustic Communications in Underwater Networks" (RACUN) under the EDA Project Arrangement No. B 0386 ESM1 GC, and by the Italian Institute of Technology within the Project SEED framework (NAUTILUS project). The authors gratefully thank L-3 ELAC Nautik, Germany for the loan of the NetDCU 5.2 board, that made it possible to carry out this comparative study. Many thanks also to Federico Beccaro, Achille Forzan and Moreno Zorzetto for their precious help in the organization and realization of the tests in the Piovego channel.

REFERENCES

[1] R. Masiero, S. Azad, F. Favaro, M. Petrani, G. Toso, F. Guerra, P. Casari, and M. Zorzi, "DESERT Underwater: an NS–Miracle-based framework to DEsign, Simulate, Emulate and Realize Test-beds for Underwater network protocols," in *MTS/IEEE Oceans*, Yeosu, Republic of Korea, Jun. 2012.

[2] "The DESERT Underwater libraries - *DESERT*," Last time accessed: August 2012. [Online]. Available: http://nautilus.dei.unipd.it/desert-underwater

[3] "The Network Simulator - *NS-Miracle*," Last time accessed: August 2012. [Online]. Available: http://dgt.dei.unipd.it/download

[4] "The Network Simulator - *ns-2*," Last time accessed: August 2012. [Online]. Available: http://nsnam.isi.edu/nsnam/index.php/User_Information

[5] "PandaBoard," PandaBoard's Homepage, Last time accessed: August 2012. [Online]. Available: http://www.pandaboard.org/

[6] "NetDCU5.2," Emlix's Web site, Last time accessed: August 2012. [Online]. Available: http://www.fs-net.de/cms/index.php?id=22&L=1

[7] B. Graaf, M. Lormans, and H. Toetenel, "Embedded software engineering: the state of the practice," *Software, IEEE*, vol. 20, no. 6, pp. 61–69, 2003.

[8] "OMAP4430," OMAP4 platform's Homepage, Last time accessed: August 2012. [Online]. Available: http://www.ti.com/product/omap4430

[9] G. V. Vaughan, B. Elliston, T. Tromey, and I. L. Taylor, *GNU Autoconf, Automake, and Libtool*. Open Publication License, 2000, Last time accessed: August 2012. [Online]. Available: http://sources.redhat.com/autobook/

[10] "BTech Acoustics, LLC," Last time accessed: August 2012. [Online]. Available: http://www.btechacoustics.com/

[11] "Woods Hole Oceanographic Institution," Last time accessed: August 2012. [Online]. Available: http://www.whoi.edu/

[12] S. Singh, S. Webster, L. Freitag, L. Whitcomb, K. Ball, J. Bailey, and C. Taylor, "Acoustic communication performance of the WHOI Micro-Modem in sea trials of the Nereus vehicle to 11,000 m depth ," in *MTS/IEEE Oceans*, Biloxi, Mississippi, USA, Oct. 2009.

[13] N. Abramson, "Development of the ALOHANET," *IEEE Transactions on Information Theory*, vol. 31, no. 2, pp. 119–123, 1985.

[14] P. Casari and M. Zorzi, "Protocol Design Issues in Underwater Acoustic Networks," *Elsevier Computer Communications*, vol. 34, no. 17, pp. 2013–2025, 2011.

[15] "Ubuntu kernel for OMAP4," Last time accessed: August 2012. [Online]. Available: http://www.omappedia.com/wiki/Ubuntu_kernel_for_OMAP4

[16] "UBUNTU release: Texas Instruments OMAP4 preinstalled server image," Last time accessed: August 2012. [Online]. Available: http://cdimage.ubuntu.com/releases/11.10/release/