Comparing the SUNSET and DESERT Frameworks for In Field Experiments in Underwater Acoustic Networks

Roberto Petroccia[†‡] and Daniele Spaccini[†‡]
[†] Dipartimento di Informatica
Università di Roma "La Sapienza," Rome, Italy
{petroccia, spaccini}@di.uniroma1.it
[‡] WSENSE s.r.l., Rome, Italy

***Abstract*—The emerging demand for pervasive underwater monitoring and control systems has significantly stimulated the research on network protocols for underwater acoustic sensor networks. In the last few years, several solutions have been proposed for this kind of networks at all layers of the protocol stack. However, to achieve a thorough understanding of the performance of these protocols running simulations is no longer enough and in field experiments are needed. Two different platforms, SUNSET and DESERT, have been recently developed and released open-source allowing to seamlessly simulate, emulate and test (at-sea) a variety of communication protocols. In this paper we compare the performance of these two frameworks, with a particular attention to their use during in field experimentation. Our tests show that when running simulations there is high compatibility and interoperability between the two systems. In actual underwater experiments, however, SUNSET represents a more mature, flexible and efficient solution.**

***Index Terms*—Underwater acoustic networks, underwater wireless sensor networks, sea trial testing, SUNSET, DESERT, simulation, emulation, ns-2, ns2-Miracle.**

I. Introduction

The recent advances in underwater acoustic modem technology have fostered the interest for undersea exploration using autonomous fixed and mobile assets [1]. Novel underwater communication solutions have been proposed in the past years to move from existing static and mobile platforms to cooperative underwater monitoring and control systems. The proposed solutions have been usually investigated by means of simulations and only few of them have been tested at sea. Simulations, however, can only capture a subset of the total environmental variability, resulting in an approximate and generally simplified model of the underwater acoustic channel and its dynamics. In field tests are therefore needed for a more accurate evaluation of the network performance in real scenarios. On the other hand, in field experiments introduce high costs and logistic complexity and they usually require to reimplement and modify the simulated solutions to support the use of real hardware. This, in turn, can result in possible errors during the code rewriting phase. To speed up the innovation in the underwater field, researchers and developers would greatly benefit from an open-source framework that could be shared by the network community at large and that could be used to test, evaluate, and compare the performance of the different protocols firstly in simulative and controlled environments, and then at sea without the need of any code rewriting. Moreover, such a framework would help identifying possible risks, malfunctioning and underperforming solutions during the preparation phase and before the actual in field experiments.

Recently two novel solutions for Underwater Acoustic Sensor Networks (UASNs) have been proposed, SUNSET and DESERT. SUNSET has been presented in 2011 [2], [3], with an improved version in 2012 [4], extending two open-source and well known network simulators (ns-2 [5] and its extension ns2-Miracle [6]) to seamlessly simulate, emulate and test in real-life novel communication protocols, without any need of code rewriting. DESERT, instead, has been presented in 2012 [7]. It reimplements the same approach proposed by SUNSET of extending the ns-2 and ns2-Miracle simulators and it provides a set of libraries to support the implementation of underwater network protocols and their investigation by means of both simulation, emulation and in field tests in a transparent way. Both of these solutions have been recently released open-source to the community (SUNSET [8] - DESERT [9]).

In this paper we investigate the similarities and differences between SUNSET and DESERT. This work aims at providing to the underwater community a more clear understanding on the pros and cons related to the use of these two frameworks, with a particular attention to their use during in field experimentation since, as presented above, it could result in a great interest by the underwater community. Since both SUNSET and DESERT extend the ns-2 and ns2-Miracle simulation engines, there is high compatibility and interoperability for the protocol solutions and additional modules developed in the two systems when running in simulation mode. However, the two frameworks make use of different architectures and modules when interacting with real hardware.

Differently from DESERT, SUNSET implements additional modules to move from simulation to in field tests in a more flexible and efficient way. DESERT instead implements a more simple architecture which is easier to learn and use, but

that incurs in additional overhead and more constraints when making use of real hardware.

Our analysis shows that SUNSET with respect to DESERT represents a more mature, flexible and efficient solution when performing experiments on UASNs during in field tests.

The rest of the paper is organized as follows. Previous works on frameworks investigating the use of underwater acoustic sensor networks are summarized in Section II. SUNSET and DESERT are then described in details in Section III and IV, respectively. Section V illustrates our comparison between the two frameworks and the collected results. Finally, Section VI concludes the paper.

II. Related Work

Different frameworks for UASNs have been proposed in the past years [10], [11], [12], [13]. All these tools implement the idea of combining acoustic modems with a software defined protocol stack and they all aim at providing flexible architectures to investigate the performance of different protocol solutions by means of both simulations and in field tests. More specifically, in [10] the Aqua-Lab testbed is presented. This system makes use of the WHOI Micro-Modem and defines software Application Programming Interfaces (APIs) as a middleware for the interaction with the modem. Developers can then use these APIs to implement new applications without knowing the exact mechanisms used at the lower layers. The Aqua-Lab framework allows the investigation of different network topologies, propagation delays, and signal attenuations. It has been used to conduct a set of experiments in both field and lab environments. In [11], the authors present the Aqua-Net architecture. It assumes a layered structure with the possibility for cross-layer optimization. Different acoustic modems can be used at the physical layer, such as WHOI Micro-Modems and Teledyne Benthos modems. The Aqua-Net architecture has been investigated running on an embedded system (Gumstix device [14]) for in field deployment and using WHOI Micro-Modems for acoustic transmissions. A similar approach is described in [12], where the authors propose a unified simulation and emulation architecture for underwater MAC protocol development. ARL OFDM modems have been considered for acoustic transmissions. The same C code implementing the MAC protocol is used for both the simulator and the modem. The simulator captures the essential behavior of the modem and uses the same software interfaces provided by the acoustic device. Using the same exact code for both simulations and in field tests, no code reimplementation is needed. This avoids the introduction of possible errors during the code rewriting phase and makes easier any performance comparison between simulation and at sea results. An extension of this work has been proposed in [13] where the UNET-2 framework is presented. It describes a software defined modem and communication protocol stack based on the ARL modem, providing the possibility to investigate new coding schemes and to evaluate complete networking solutions by means of both simulation and in field tests.

The limit of all the solutions presented above is that proprietary architectures and software have to be studied by the developers to let them use these systems, which is usually not trivial. Moreover, the proposed architectures are not open enough to be considered in different settings and to support the investigation of different hardware, thus limiting their use for in field testing.

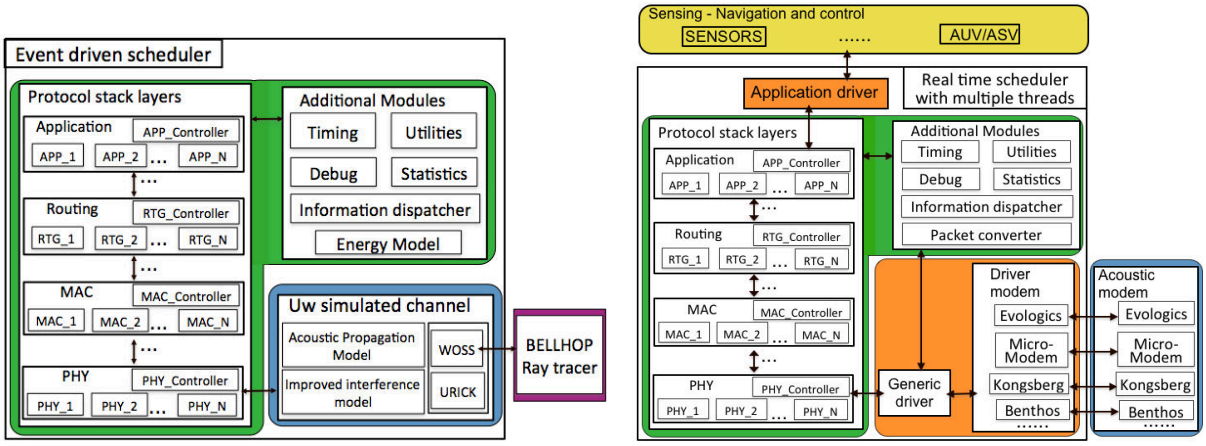
The first work considering the use of an open-source platform to create a simulation/emulation tool for UASNs is presented in [15]. The authors propose a software defined modem and communication stack, named UANT, which is based on the GNU radio and TinyOS or TOSSIM software. GNU radio is an open-source software development toolkit that provides signal processing blocks to implement software radios. It has been used by the authors to transmit the waveform on the acoustic channel using an acoustic transducer. TinyOS and TOSSIM are generally used to implement the communication protocol stack for terrestrial radio network. TinyOS is used when running on real wireless sensor devices, TOSSIM instead when running simulations. The same code simulated by TOSSIM can be ported to work on TinyOs. They have been used in UANT to implement the communication protocol stack for the underwater network.

Although the proposed UANT architecture is really interesting, it could be difficult to use it for real in field experimentation. In fact, the GNU radio software is computationally expensive and it needs to run on a PC, which is unlikely to be used during real-life experiments due to housing constraints. Additionally, to run on TinyOS, the protocol design has to follow rules that reflect features of IEEE 802.15.4 devices and that differ from features and constraints of underwater nodes and underwater communication devices. Moreover, TOSSIM is currently not supporting any underwater channel model when running simulations.

There is therefore the need to investigate novel open-source frameworks for UASNs that allow to simulate, emulate and test in field networking protocol solutions and that can be used to run on small embedded devices for real at sea deployment. To fill this gap, SUNSET and DESERT have been developed and released open-source to the community in May 2012, with an improved and updated version of SUNSET released in May 2013. In what follows we describe in details these two approaches, highlighting their similarities and differences. We focus our analysis on the use of SUNSET and DESERT during in field experimentation which has experienced an increasing interest by the underwater community in the past months.

III. SUNSET

The Sapienza University Networking framework for underwater Simulation, Emulation and real-life Testing (SUNSET) is a novel solution developed to seamlessly simulate, emulate and test at sea communication protocols. SUNSET has been the first solution presented in the literature making use of the open-source and well known network simulator ns-2 [5], and its extension ns2-Miracle [6], to investigate the performance of novel protocols for UASNs by means of both simulations



(a) SUNSET modules in simulation mode.

(b) SUNSET modules in emulation mode.

Figure 1: SUNSET architecture - Simulation and emulation mode.

and in field tests. Using SUNSET, anyone willing to implement its solutions for simulation experiments using ns-2 can use the same exact code in emulation mode, adopting real acoustic modems for data transmission and additional external devices for sensing and navigation operations. The SUNSET architecture, shown in Figure 1, inherits from ns-2 and ns2-Miracle the use of a layered organization, the possibility to have multiple solutions at each layer of the protocol stack and the possibility of cross-layer interaction between the different layers of the stack. The design and implementation of novel solutions in SUNSET follow the same basic rules of the ns-2 and ns2-Miracle simulators and the additional effort required by researchers and developers to use SUNSET is really limited.

Several MAC and routing solutions have been implemented and validated in the past years, composing the *SUNSET Networking Components*.

The *SUNSET Networking Components* include:

- **MAC**: Aloha, CSMA-Aloha, Slotted-CSMA, TDMA, CSMA [16], Tone-Lohi, [17] and DACAP [18].
- **Routing**: Flooding, probabilistic flooding, static routing, an improved version of the solution presented in [19].
- **Cross-layering**: Channel-Aware Routing Protocol (CARP) [20].

Additional modules have also been designed and implemented in SUNSET to make the execution of simulation and emulation easier and more transparent to the user (Figure 1). Some of these modules are used for both simulation and emulation (Timing, Utilities, Information Dispatcher, Debug and Statistics), composing the *SUNSET Core Components*; others are only needed when using real hardware (Real time scheduler, Packet Converter and Drivers for the interaction with the external devices), composing the *SUNSET Emulation Components*. With respect to ns-2, SUNSET provides also a novel energy model when running in simulation mode, which more accurately captures the energy consumption of the underwater devices.

A brief description of the additional core and emulation components is presented below.

SUNSET Core Components.

- Timing module: It is responsible to compute the delays related to the use of real devices, which are usually ignored when running simulations. When in simulation mode, the Timing module returns the simulated values defined by the user. When running in emulation mode instead it collects these delays from the real hardware inquiring the corresponding drivers.

- Utilities module: It is responsible to schedule events and update timing information according to the used scheduler, i.e., the event driven scheduler, when running in simulation mode, and the real-time scheduler, when running in emulation mode. Moreover, the utilities module is also responsible to properly free the memory allocated for a ns-2 packet and its payload, if any. Timing and Utilities modules are used to make transparent to the users the use of the framework in simulation and emulation mode.

- Information dispatcher module: It implements a publish-subscribe pattern, allowing each module to request or provide information to the other modules without the need to send cross-layer messages across the entire protocol stack, as it is done by ns2-Miracle. Each information is timestamped in order to check its freshness and it can work together with the ns2-Miracle cross-layer messaging system.

- Debug module: It implements a more flexible system to log the debugging information according to their priorities.

- Statistics module: This module provides basic functionalities to evaluate the protocol performance. The user can easily collect all the metrics (per node or in the network) it is interested in: Packet delivery ratio, network throughput, packet latency, energy consumption, etc. This module can be also extended according to the additional information and metrics the user wants to evaluate.

- Energy module: It implements a more accurate energy model to trace the energy consumption of underwater devices, supporting also the use of different transmission powers.

SUNSET Emulation Components.

- Real-time scheduler: It implements a new real-time scheduler to more accurately capture the elapsed time with respect

to the real-time scheduler provided by ns-2. It allows also to have multiple threads interacting with the scheduler in a more efficient way.

- Packet converter module: It allows to convert an ns-2 packet into a stream of bytes and vice versa, compressing the information in the header to the minimum size. The user can decide what fields in the packet header have to be converted and how many bits have to be reserved for each field. Any change can be performed in an easy and fast way without the need to recompile the implemented code.

- Drivers: SUNSET has been interfaced with several external devices: 1) Acoustic modems (WHOI Micro-Modem; Evologics modem, Kongsberg modem and Teledyne Benthos modem); 2) Sensing platforms (temperature, CO_2 and methane concentrations [21], Acoustic Doppler Current Profiler - ADCP); 3) Mobile vehicles (MARES AUV and INESC/TEC ASV [22], eFolaga AUV). The designed architecture is flexible and open enough to allow the integration of whatever external device once APIs are provided to control its operations.

SUNSET has been recently released open-source to the scientific community [8], however, some of the SUNSET modules presented above are not freely available to the community since they are covered by a No Disclosure Agreement (NDA) and cannot be released open-source at the moment.

All the SUNSET modules and drivers have been extensively tested, validated and improved during more than twelve at sea campaigns conducted in the past three years. When running in field tests, SUNSET has been successfully ported on small portable devices (Gumstix [14], IGEPv2 [23], PC104 and other ARM-based systems), thus allowing the user to embed it inside modem or AUV housings, making easier the deployment at sea.

IV. DESERT

DESERT Underwater (short for DEsign, Simulate, Emulate and Realize Test-beds for Underwater network protocols) is a complete set of public C++ libraries to support the design and implementation of underwater network protocols. DESERT reimplements the same idea proposed by SUNSET, providing a second simulation/emulation tool based on the ns-2 and ns2-Miracle simulators. Similarly to SUNSET, DESERT implements a communication and networking architecture that allows heterogeneous nodes to communicate reliably in the underwater environment. This architecture provides to the user a complete toolkit to investigate the performance of underwater network solutions by means of simulations and in field tests without the need of any code rewriting. This allows to speed-up the protocol investigation and validation, avoiding the possibility to introduce errors during the code reimplementation phase.

DESERT implements different solutions at all layers of the protocol stack which are grouped according to the stack layers defined by the OSI model (Figure 2). In what follows we briefly describe the different DESERT modules:

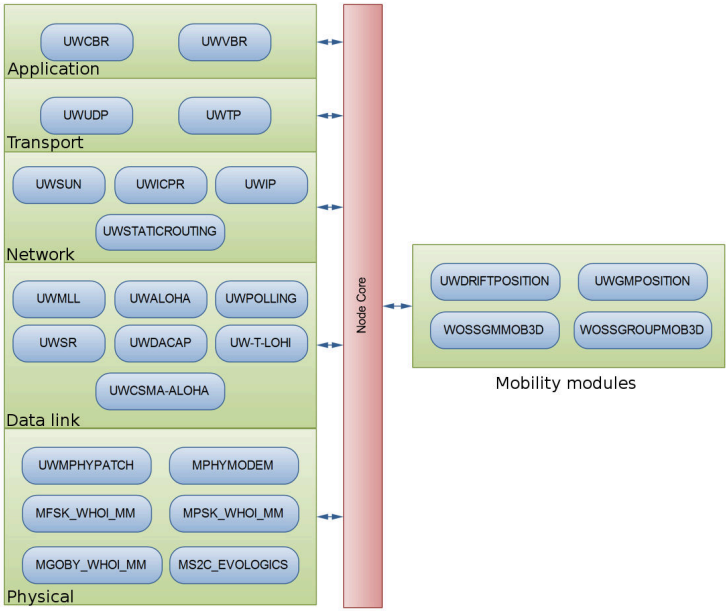


Figure 2: DESERT architecture as presented in [9].

- Application layer. Two modules are provided, *uwcbr* and *uwvbr*, implementing constant and variable bit rate traffic generations, respectively. A single *uwcbr* or *uwvbr* module represents a data flow between a pair of nodes (source and destination) and if one node is the destination for more than one source, multiple modules are needed, one for each source.

- Transport layer. Two modules are provided for the transport layer, a simple module called *uwudp* and a more sophisticated one named *uwtp*, short for underwater transport protocol. The *uwudp* module implements flow multiplexing and demultiplexing mechanisms for the interaction with the upper layers. The *uwtp* provides also error control technique and functionalities for in-order data delivery.

- Network layer. Three different routing protocols are implemented: Static routing protocol (*uwstaticrouting*); Flooding based mechanism (*uwicrp*); Dynamic, reactive source routing protocol (*uwsun*) which implements different criteria for next hop relay selection, e.g., minimum hop-count, maximum Signal to Noise Ratio (SNR) along the links of the path [24].

- Data link layer. The core of the data link layer is the Medium Access Control. Six different MAC protocols are implemented together with a link layer module mapping IP addresses to MAC addresses (*uwmll*). The six MAC protocols provided by DESERT are: Aloha (*uwaloha*), CSMA-Aloha (*uw-csma-aloha*), Tone-Lohi (*uw-t-lohi*) [17], DACAP (*uwdacap*) [18], UW-Polling (*uwpolling*) [25] and an error controlled CSMA-Aloha (*uwsr*) [26].

- Physical layer. This layer includes the interface between the network simulator and the actual modem hardware. When running in emulation mode, DESERT has been interfaced with FSK and PSK WHOI Micro-Modems and with the Evologics modem.

Additionally, DESERT implements four different mobility models that can be used to simulate underwater robot movements. These modules can simulate node mobility in 2D as well as 3D scenarios. The *uwdriftposition* module mimics the

drift of a node caused by ocean currents according to the speed and direction of the waves. The *uwgmposition* module implements a Gauss-Markov Mobility Model to produce smooth and realistic traces by appropriately tuning a correlation parameter α. This module updates node speed and direction according to a finite state Markov process. Similarly, the *wossgmmob3D* implements the Gauss Markov Mobility Model, but with some changes that make it directly usable with the WOSS [27] interface. Finally the *wossgroupmob3D* module implements a group mobility model where a leader node guides the movement of one or more follower nodes.

All the DESERT modules presented above are freely available to the community and have been released open-source in the current DESERT distribution [9].

DESERT has been used in several in field campaigns during the past year. To make easier the network deployment at sea, DESERT has been successfully ported on small computational platforms including PandaBoard, NetDCU – as described in [28] – and IGEPv2.

Differently from SUNSET, DESERT does not implement any additional module when moving from simulation to emulation. DESERT uses the basic ns-2 real-time scheduler and the cross-layer mechanism provided by ns2-Miracle. Moreover, no additional modules have been implemented to support the conversion of ns-2 packets into streams of bytes for in water transmissions. The conversion of the packet header fields and the size assigned to the each of these fields are hard coded and cannot be fast changed by the user. Any change requires that the implemented code is recompiled and that the updated libraries are uploaded again to each node in the network.

DESERT, implementing a more simple architecture, can be more easily used and learned by the user, but it results in a less flexible and efficient solution with respect to SUNSET when used during in field experimentation, as described in the next section.

V. Performance Evaluation

This section describes SUNSET and DESERT similarities when running in simulation mode and their differences when running in emulation mode.

A. Simulation mode

Both SUNSET and DESERT extend the ns-2 and ns2-Miracle simulators, which means that they are both compatible with these two simulators, inheriting the functionalities they provide. This translates in high compatibility and interoperability between the protocol solutions and additional modules developed in SUNSET and DESERT when running in simulation mode. In this case in fact, both frameworks use the same ns-2 event driven scheduler and the same ns-2 and ns2-Miracle main modules. They can both use the different underwater acoustic channel models supported by ns2-Miracle, e.g., empirical formulas [29] and Bellhop ray tracing [30] via the WOSS [27] interface, and they are both able to use any additional module or more accurate channel models designed for ns2-Miracle.

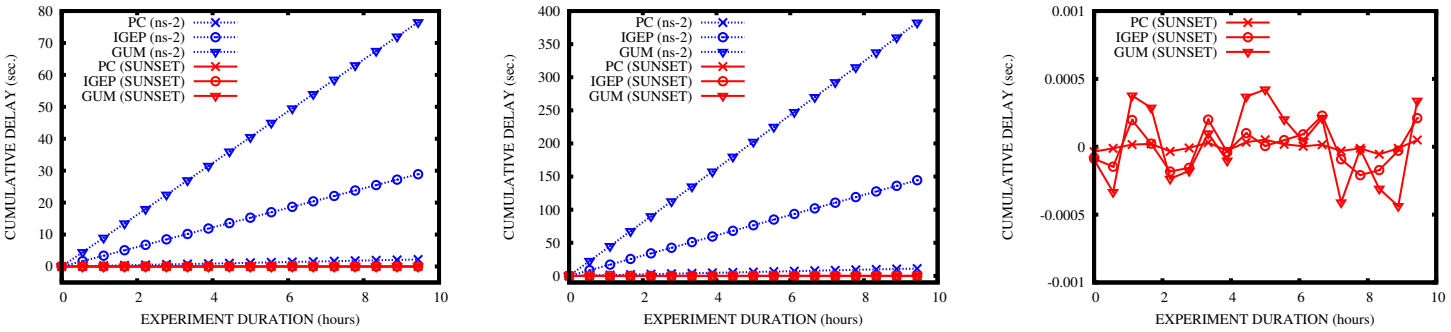
This means that the mobility models implemented by DESERT, being themselves ns2-Miracle modules, can be loaded by SUNSET in a transparent way and that the more accurate energy model implemented by SUNSET can be easily used by DESERT. Similarly, the networking protocol solutions provided by each of the two frameworks can be easily used by the other one when running simulations. All DESERT communication protocols can be loaded by SUNSET in a transparent way, while to let DESERT use the SUNSET protocol solutions, the *SUNSET Core Components* have to be loaded as well. This is because the SUNSET network solutions make use of the Information Dispatcher module to exchange cross-layer information, and of the Timing and Utilities modules to compute the system delays and schedule the event according to the framework running mode (simulation or emulation). To avoid the use of the *SUNSET Core Components* few changes in the SUNSET network solutions code are needed: Replacing the use of the Information Dispatcher with ns2-Miracle cross-layer messages; making direct use of the ns-2 event driven scheduler; and loading all the considered delays directly from the experiment script file, as it is done by DESERT.

Overall, we can conclude that the modules and solutions designed for each framework can be easily used by the other one and that there is no real difference in the selection of one system or the other when performing simulations.

B. Emulation mode

Differently from the simulation mode, there are several differences between SUNSET and DESERT when they run in emulation mode, resulting in a different efficiency and flexibility achieved by the two frameworks when performing in field tests. The main differences between SUNSET and DESERT stay in the use of two different real-time schedulers and of two different approaches to convert ns-2 packets into streams of bytes for underwater transmissions. In what follows we first investigate the use of the two real-time schedulers, evaluating the performance of each scheduler in terms of event scheduling accuracy and in terms of CPU usage for the operations required by each scheduler to interact with the external devices. We then investigate the CPU and RAM usage related to the two packet conversion approaches.

Monitoring the CPU usage of a process in an accurate way is not an easy task, especially when the process is not introducing an intense and continual computational activity but short and sporadic operations. During the conducted analysis, we have seen that the CPU usage of the two frameworks is very limited and it varies too fast to be monitored using tools like systat (pidstat) or infoRAM [28]. In fact these tools, using kernel calls, can properly monitor the CPU usage with a sampling frequency no higher than 1000Hz, which actually represents the maximum tick rate supported by the Linux kernel, as reported in [31]. This sample frequency is still too low to properly capture and trace the CPU usage and the energy consumption for both SUNSET and DESERT. To accomplish this task we have therefore considered an hardware power meter, more specifically the Monsoon FTA22D Mobile



(a) One event generated every 1 sec. (b) One event generated every 0.2 sec. (c) Zoom into one event generated every 0.2 sec.

Figure 3: Cumulative computational delay added by the different platforms.

Device Power Monitor [32]. This tool is largely used by the networking community to monitor the power consumption of cellphone and smartphone devices for benchmark purposes and it is able to accurately sample the power consumption at a rate of 5000Hz. Using this device it has been possible to accurately trace the power and energy consumption related to the SUNSET and DESERT operations.

To monitor the RAM usage instead the infoRAM software has been considered. This software is provided by the DESERT developers and it has already been used to evaluate DESERT performance when running on embedded platforms, as presented in [28].

We have monitored the energy consumption and RAM usage of both SUNSET and DESERT when they are in idle state and when they are performing packet transmissions/receptions.

Three platforms with different computational powers have been considered for our tests: 1) Gumstix verdex pro (CPU:Marvell PXA270 @600MHz, RAM:128MB low-power DDR; 2) IGEPv2 (CPU:ARM Cortex-A8 DM3730 @1Ghz, RAM: 512 MB); 3) Desktop PC (CPU:Core(TM)2 Quad CPU Q6600 @2.40GHz, RAM: 4 GB).

Real-time scheduler

DESERT uses the real-time scheduler provided by ns-2, while SUNSET implements its own real-time scheduler, which is an improved version of the ns-2 one. The reasons why SUNSET implements a different real-time scheduler are: 1) The ns-2 real-time scheduler does not compensate for any computational delay added by the specific platform used to run the software; 2) The ns-2 real-time scheduler does not support straightforward interaction among multiple threads and it requires that the main thread is actively checking for new incoming events or data, which in turn results in additional CPU usage and additional constraints on the selection of the checking rate.

- *Scheduler accuracy*.

Each time an event has to be scheduled some computational delay is introduced by the hardware platform which has to execute the event scheduling operations. We have estimated that using the ns-2 real-time scheduler the delay added to create the event, to add it to the ordered structure of the scheduler, and to remove it from the scheduler at the proper time is about: 0.065 milliseconds on the Desktop PC, 0.75 milliseconds on the IGEPv2 board (IGEP) and 1.95 milliseconds on the Gumstix board (GUM). The Gumstix being the less powerful board is the one introducing the higher delay but is also the one consuming less energy, which is quite important for underwater applications. The ns-2 real-time scheduler does not compensate for these delays that are cumulative added to the system clock. Let us consider the case where the basic ns-2 real-time scheduler is used together with a periodic timer, assuming a time period of t seconds. Let us assume also that each time the timer expires, the user simply restarts it repeating the period of t seconds, as it is usually done in ns-2, without actively compensating for the additional delays. In such a case, these delays are cumulative added to the system, resulting in an increasing difference between the time when the event is supposed to occur with respect to the actual time when it occurs. This difference depends on the specific platform used to run the system, which means that if there are different platforms in the network all equipped with a really accurate clock and all synchronized, even if the clocks keep staying synchronized the difference in the computational delays results in a desynchronized event scheduling in the network. The user has to therefore actively compensate for these delays, which could make the software development complicated. The SUNSET real-time scheduler instead introduces on average a computational delay which is 65% shorter than the one of the ns-2 scheduler and it also automatically compensates for this delay, without requiring any additional effort by the user.

Figure 3 shows the effect of the computational delay on the three considered platforms when using the basic ns-2 and the SUNSET real-time scheduler. Figure 3a shows the case where one event is periodically scheduled every 1 second and Figure 3b every 0.2 seconds (these cases mimic a possible periodic reading of data from a sensor or from the navigation system of a vehicle). We can see that using the ns-2 real-time scheduler the cumulative delay introduced by the platform operations increases over time (up to several tens of seconds) depending on the platform computational power. SUNSET instead compensating for the additional delays results in a clock difference that is not increasing over time and it is always close to 0. Figure 3c zooms into the case where SUNSET schedules a periodic event every 0.2 seconds. We can clearly see that the clock difference for each event is always shorter than 0.5 milliseconds.

- *Multithreading and interaction with external devices.*
Both SUNSET and DESERT use multiple threads when interacting with external devices. The main thread is always running the scheduler and the protocol solutions while secondary threads are used for data exchange with the external hardware. The ns-2 real-time scheduler does not support straightforward interaction among multiple threads. Every secondary thread which has to notify a new event to the main thread, e.g., incoming data from an acoustic modem, sensing platform, mobile vehicle, etc., cannot directly inform the main thread and let it immediately check if an action has to be taken. This means that the main thread has to actively control for any new event and data provided by the secondary threads. This active checking results in an additional CPU usage and also in the issue of proper tuning the checking rate. In fact, checking too fast requires the use of additional resources (CPU and energy) and results in a reduction of the device lifetime. On the other hand, if the checking rate is too low, possible important events or data detected by secondary threads (e.g., actions related to an alarm coming from a sensor or a vehicle) need to be delayed till the next time the main thread will check for them. Additionally, since the information provided by the secondary threads cannot be immediately used by the main thread, these information have to be stored in some way and have to be then read by the main thread when checking for them. Differently from the ns-2 real-time scheduler, the SUNSET scheduler permits to secondary threads to immediately inform and activate the main thread in case of new events or data detection. In this way the main thread can be passively waiting without using additional computational resources and without delaying any of the notified events.

DESERT, making use of the ns-2 real-time scheduler, incurs in the need to periodically check for incoming events. Moreover, the current mechanism implemented by DESERT to notify to the main thread information coming from an external device makes use of a file. When the secondary thread receives a message from the device, it writes the received data to the file. When the main thread checks for novel information coming from the device, it reads them from the file. Therefore, the additional resources requested by DESERT are not only related to the need of actively check for novel incoming information but also to the fact that a file has to be accessed.

To measure the energy consumption related to the use of the SUNSET and DESERT approaches we have used the Monsoon FTA22D device. It is not possible to connect such tool to a PC, we have therefore restricted our investigation only to the Gumstix and the IGEPv2 boards.

We have firstly investigated the average power and energy consumption for the Gumstix and IGEPv2 when none of the two frameworks is running on these devices (referred below as "IDLE"). We have then run the same tests considering SUNSET and DESERT in idle state, i.e., without performing any data generation, transmission or reception. For DESERT, we have investigated three different checking rates: Every 1 second (default value), every 0.5 seconds, and every 0.25 seconds.

Although no data generation and packet transmission/reception have been considered for this test, we have assumed for both systems a complete protocol stack (Application, Transport, Network, MAC and Phy) and Evologics acoustic modem with Ethernet connection as communication device. We wanted somehow to investigate the resources required by SUNSET and DESERT when used for an underwater environmental monitoring system where the underwater nodes stay in idle state most of the time waiting for data collected by the sensors. For a fair comparison, we have used one of the scripts provided by DESERT ("TESTBED_n1_sample.tcl") to configure the protocol stack of underwater nodes. The same protocol stack configuration has been then considered in SUNSET, taking solutions from the ns2-Miracle libraries when a SUNSET module for that layer was not available.

Multiple tests have been investigated, each of them running for 10 minutes, which was enough to achieve an accurate statistical confidence for the collected data.

In the presented results we can see that SUNSET outperforms DESERT in all the considered scenarios (red lines vs. blue ones, respectively, if the paper is seen in colors; steady lines vs. dotted lines otherwise).

Figure 4 shows a portion of 20 seconds from the power consumption traces collected for the Gumstix board. In this Figure a checking rate of 0.25 seconds has been considered for DESERT. We can clearly see that every 0.25 seconds there is a peak in the power consumption meaning that the main thread is checking for new incoming events and accessing the file.

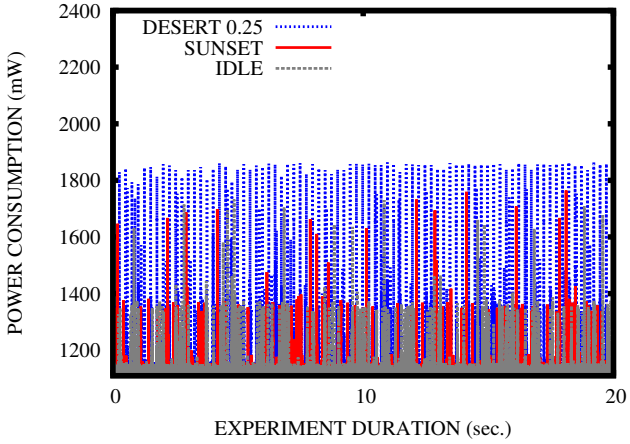


Figure 4: Gumstix - Power consumption.

For each of these peaks, the power consumption is on average $\sim$ 62% higher than the case where the board is in IDLE. SUNSET instead without requiring any additional operation has always a power consumption really close to the IDLE one. Considering a checking rate of 1 second and 0.5 seconds, the same exact peaks occur but at a lower frequency.

Table I shows the average power consumption of the different systems. We can see that increasing the checking rate the power consumption increases as well.

Figure 5 shows the additional energy that would be required to run SUNSET and DESERT over several months of

	mW
IDLE	1117.26
SUNSET	1117.44
DESERT 1	1119.20
DESERT 0.5	1120.01
DESERT 0.25	1121.76

Table I: Gumstix - Average power consumption (idle state).

	mW
IDLE	2558.19
SUNSET	2558.28
DESERT 1	2559.65
DESERT 0.5	2559.83
DESERT 0.25	2560.70

Table II: IGEPv2 - Average power consumption (idle state).

deployment using a Gumstix board. These values does not consider the energy required to run the board itself, but only the additional energy required by the two systems. We can clearly see that when the system has to last for months the additional energy required to run DESERT can significantly reduce the lifetime of the underwater nodes.

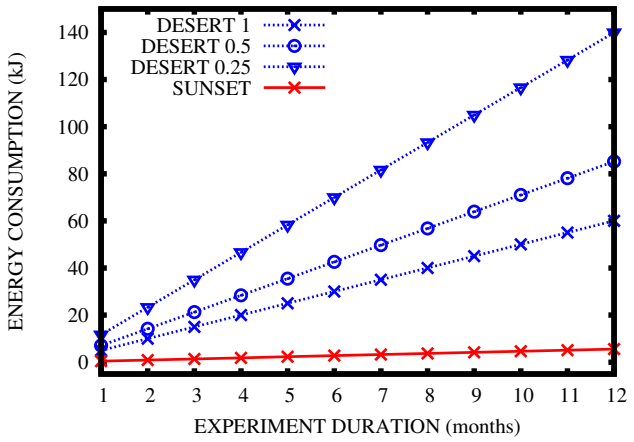


Figure 5: Gumstix - Additional energy consumption.

Figure 6 shows the instantaneous power consumption when the IGEPv2 board is considered. We can see that this board, being more powerful, requires more resources to run.

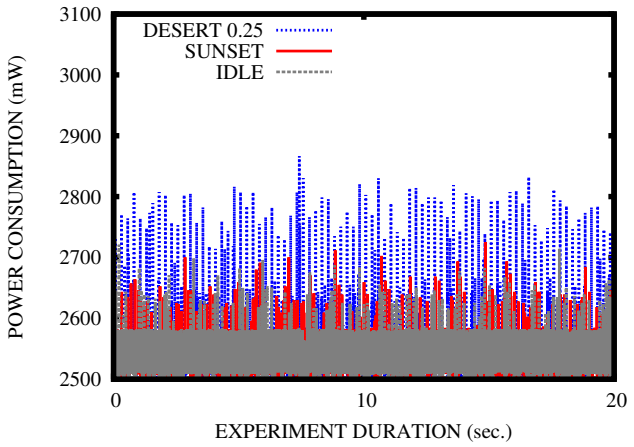


Figure 6: IGEPv2 - Power consumption.

The power consumption for the IGEPv2 in IDLE is more than twice higher with respect to the Gumstix one. Again the different peaks occurring for DESERT are clearly visible, but in this case the higher computational power of the board results in power consumption peaks that are on average only 8% higher with respect to the IDLE case. Table II shows the average power consumption of the different systems when using the IGEPv2 board. The additional energy required by SUNSET and DESERT to run on this board reflects the higher power consumption with respect to the Gumstix board but follows a similar trend.

The presented results clearly show that the SUNSET real-time scheduler not only allows to achieve an higher scheduling accuracy, but it also supports a higher efficiency when multiple threads are used, resulting in the need of less resources to run with respect to DESERT.

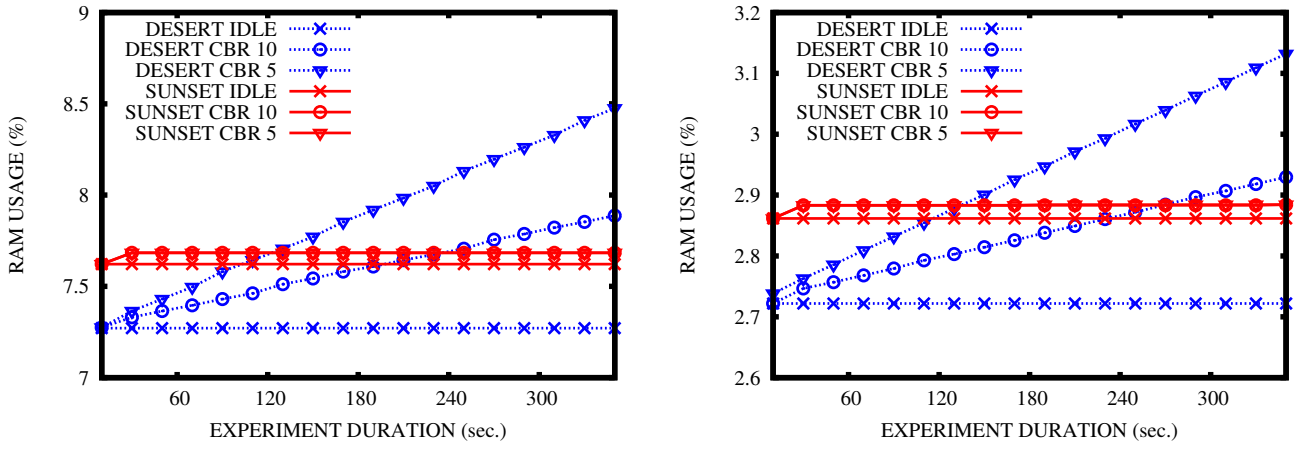
Packet converter

Another difference between SUNSET and DESERT consists in the mechanism they use to convert the ns-2 packets into streams of bytes for in water transmissions. SUNSET allows the user to select what packet header fields have to be considered during the conversion and what size (bits) to use for each field. The amount of information to transmit can be therefore reduced to the minimal size according to investigated network and experiment settings. To implement this flexible mechanism, only 2 bits of overhead are added by each layer involved in the packet conversion process. However, this mechanism requires that when the user defines a new packet header, it also provides the methods for the packet conversion, according to the basic rules defined by the SUNSET Packet Converter module. In this way the user, which knows exactly all the information it has defined and what they represent, can properly define the policies to convert and compress these information.

DESERT instead provides a static approach for packet conversion. This approach is much more simple than the one provided by SUNSET and does not require any additional effort by the user. However, the conversion of the packet header fields and the size assigned to them are hard coded and cannot be fast changed by the user. Any change requires to recompile the implemented code and to upload the updated libraries to all the underwater devices.

The amount of information currently converted by DESERT is statically set to 17 Bytes, despite the actual number of nodes in the network, the expected number of generated packets, or any of the other possible parameters of the experiment.

With respect to DESERT, the SUNSET packet conversion mechanism allows to significantly reduce the amount of bytes needed to convert the packet header information according to the considered network and experiment settings. If less than 16 nodes are in the network and less than 1024 packets are generated by each node, to convert the same packet header information considered by DESERT only 6 Bytes will be needed. If we double the number of nodes or the number of generated packets only few more bits will be



(a) Gumstix.

(b) IGEPv2.

Figure 7: RAM usage when running on Gumstix and IGEPv2.

added. Considering Evologics modem instant messages and the highest transmission power supported by the Evologics acoustic modem, every byte of data transmitted in water results in a energy consumption of $\sim$ 1.2 joule. SUNSET allows therefore to reduce the time and energy required by each transmission/reception which is a critical aspect for underwater acoustic sensor networks. To run this flexible conversion mechanism, however, SUNSET introduces a higher number of operations and needs to load more modules with respect to DESERT, resulting in a higher CPU and RAM usage.

Using the Mobile Device Power Monitor we have investigated the case where both systems are transmitting/receiving packets according to a constant bit rate traffic generation. We have considered the case where a packet header of 17 Bytes without payload (as supported by DESERT) is transmitted/received every 10 and 5 seconds. For SUNSET, 17 Bytes are the size of the packet header after the conversion and compression process to have the same amount of bytes transmitted and received by the acoustic devices for both the frameworks. We have estimated that for each packet transmission/reception, SUNSET incurs in an average power consumption $\sim$ 5% higher than DESERT when running on the Gumstix, while on the IGEPv2 board the additional power consumption is on average lower than 1%. This means that, even if SUNSET requires more energy to transmit/receive each packet, the overall energy needed to run DESERT and to actively check for incoming events is always higher than the one used by SUNSET. To consume less energy than SUNSET, the DESERT checking rate should be 10 times lower than the transmission/reception rate when using the Gumstix board and 6 times lower when using the IGEPv2 board, which is however not practical since it would result in several new incoming data waiting to be used before the main thread can check for them.

To monitor the additional amount of RAM required by SUNSET, the infoRAM tool has been used. Figures 7a and 7b show the percentage of RAM used by SUNSET and DESERT when running on the Gumstix and the IGEPv2, respectively. The two boards provide different RAM capacities, however, the same differences in the percentage of RAM used by the two frameworks have been observed. No differences in the RAM usage have been experienced when using DESERT with different checking rates. We can see that for both of the boards, loading the additional libraries required by SUNSET (*SUNSET Core and Emulation Components*) results in a RAM usage in idle state $\sim$ 4.7% higher than the one for DESERT. When packets start to be transmitted/received, SUNSET and DESERT RAM usage slightly increases since the memory for these packets has to be allocated. In this case, the SUNSET additional RAM usage increases to $\sim$ 5.1% since some additional memory is needed by the conversion process itself, which is however still very limited. We can see that the RAM usage of SUNSET is constant over time. In fact it frees the allocated memory when the packets are not needed anymore so that this memory can be reused for future packets. This also explains why there is no difference in the SUNSET RAM usage for the two considered traffic loads. When considering DESERT instead, it seems there is a memory leak and the RAM usage starts increasing over time. We can see also that the RAM usage increases faster when a higher traffic load is considered, meaning that the memory leak should be related to the packet memory allocation, as also noted by the DESERT authors in [28].

The presented results show that the SUNSET packet conversion mechanism provides a higher control on the amount of information transmitted in water. These information can be reduced to the minimal size allowing to transmit less bytes in water and to reduce the amount of energy consumed by the acoustic modem. With respect to DESERT, this flexibility comes at the price of a slight increase in the RAM usage and of the implementation of few additional lines of code.

VI. Conclusions

We have investigated the use of two frameworks, SUNSET and DESERT, recently released open-source, that allow to investigate the performance of underwater sensor networks by means of both simulations and in field tests. Both frameworks extend the well known and largely used ns-2 and ns2-Miracle open-source software. This means that there is high compatibility and interoperability between the two systems

when running in simulation mode. However, they implement different approaches when moving to in field experiments. When running in emulation mode, DESERT implements a more simple but less efficient solution. SUNSET instead, defining a slightly more complex architecture and making use of additional modules, e.g., modules for event scheduling and packet conversion, results in a much higher scheduling accuracy, in a higher efficiency and flexibility when converting packet information, and in the possibility to significantly save system resources when running on low power embedded devices.

The conducted analysis clearly shows that SUNSET represents a more mature, flexible and robust framework for in field testing with respect to DESERT. This is also the result of the larger learning gained during the more intense SUNSET experimental activities conducted in the past years.

ACKNOWLEDGMENTS

This work was supported in part by the EU FP 7 STREP project CLAM "CoLlAborative EMbedded Networks for Submarine Surveillance."

REFERENCES

[1] J. Heidemann, M. Stojanovic, and M. Zorzi, "Underwater sensor networks: Applications, advances, and challenges," *Royal Society*, vol. 370, no. 1958, pp. 158–175, May 2012.

[2] C. Petrioli, R. Petroccia, J. Shusta, and L. Freitag, "From underwater simulation to at-sea testing using the ns-2 network simulator," in *Proceedings of IEEE/OES OCEANS 2011*, Santander, Spain, June, 6–9 2011.

[3] C. Petrioli, R. Petroccia, and J. Potter, "Performance evaluation of underwater mac protocols: From simulation to at-sea testing," in *Proceedings of IEEE/OES OCEANS 2011*, Santander, Spain, June, 6–9 2011.

[4] C. Petrioli and R. Petroccia, "SUNSET: Simulation, emulation and real-life testing of underwater wireless sensor networks," in *Proceedings of IEEE UComms 2012*, Sestri Levante, Italy, September, 12–14 2012.

[5] The VINT Project, *The ns Manual*. http://www.isi.edu/nsnam/ns/, 2002.

[6] N. Baldo, F. Maguolo, M. Miozzo, M. Rossi, and M. Zorzi, "ns2-MIRACLE: A modular framework for multi-technology and cross-layer support in network simulator 2," in *Proceedings of the 2nd International Conference on Performance Evaluation Methodologies and Tools, ValueTools 2007*, Nantes, France, October 23–25 2007, pp. 1–8.

[7] R. Masiero, S. Azad, F. Favaro, M. Petrani, G. Toso, F. Guerra, P. Casari, and M. Zorzi, "DESERT underwater: an NSmiracle-based framework to DEsign, simulate, emulate and realize test-beds for underwater network protocols," in *Proceedings of MTS/IEEE OCEANS 2012*, Yeosu, Korea, May, 21–24 2012.

[8] SENSES Lab, "SUNSET: Sapienza university networking framework for underwater simulation, emulation and real-life testing," Last time accessed: March 2013. [Online]. Available: http://reti.dsi.uniroma1.it/UWSN_Group/index.php?page=sunset

[9] Nautilus project, "DESERT Underwater: an ns-miracle extension to design, simulate, emulate and realize test-beds for underwater network protocols," Last time accessed: March 2013. [Online]. Available: http://nautilus.dei.unipd.it/desert-underwater

[10] Z. Peng, J.-H. Cui, B. Wang, K. Ball, and L. Freitag, "An underwater network testbed: Design, implementation and measurement," in *Proceedings of the third ACM International Workshop on UnderWater Networks (WUWNet '07)*, Montréal, Quebec, Canada, September 14 2007.

[11] Z. Peng, Z. Zhou, J.-H. Cui, and A. Shi, "Aqua-Net: An underwater sensor network architecture: Design,implementation, and initial testing," in *Proceedings of MTS/IEEE OCEANS 2009*, Biloxi, Mississippi, USA, October, 26–29 2009.

[12] S. Shahabudeen, M. Chitre, M. Motani, and A. L. Y. Siah, "Unified simulation and implementation software framework for underwater MAC protocol development," in *Proceedings of MTS/IEEE OCEANS 2009*, Biloxi, Mississippi, USA, October, October, 26–29 2009.

[13] M. Chitre, I. Topor, and T.-B. Koay, "The UNET-2 modem - An extensible tool for underwater networking research," in *Proceedings of MTS/IEEE OCEANS 2012*, Yeosu, Korea, May, 21–24 2012.

[14] "Gumstix inc." Last time accessed: March 2013. [Online]. Available: http://www.gumstix.com

[15] D. Torres, J. Friedman, T. Schmid, and M. B. Srivastava, "Software-defined underwater acoustic networking platform," in *Proceedings of the Fourth ACM International Workshop on UnderWater Networks (WUWNet '09)*, Berkeley, California, USA, November 3 2009, pp. 7:1–7:8.

[16] S. Basagni, C. Petrioli, R. Petroccia, and M. Stojanovic, "Choosing the packet size in multi-hop underwater networks," in *Proceedings of IEEE OCEANS 2010*, Sydney, Australia, May, 24–27 2010, pp. 1–9.

[17] A. Syed, W. Ye, and J. Heidemann, "Comparison and evaluation of the T-Lohi MAC for underwater acoustic sensor networks," *IEEE Journal on Selected Areas in Communications*, vol. 26, no. 9, pp. 1731–1743, December 2008.

[18] B. Peleato and M. Stojanovic, "Distance aware collision avoidance protocol for ad-hoc underwater acoustic sensor networks." *IEEE Communications Letters*, vol. 11, no. 12, pp. 1025–1027, December 2007.

[19] J. Alves and G. Zappa, "Low overhead routing for underwater acoustic networks," in *Proceedings of IEEE/OES OCEANS 2011*, Santander, Spain, June, 6–9 2011.

[20] S. Basagni, C. Petrioli, R. Petroccia, and D. Spaccini, "Channel-aware routing for underwater wireless networks," in *Proceedings of MTS/IEEE OCEANS 2012*, Yeosu, Korea, May, 21–24 2012.

[21] A. Annunziatellis, S. Graziani, S. Lombardi, C. Petrioli, and Petroccia, "CO2Net: A marine monitoring system for CO_2 leakage detection," in *Proceedings of MTS/IEEE OCEANS 2012*, Yeosu, Korea, May, 21–24 2012.

[22] N. A. Cruz, B. M. Ferreira, A. C. Matos, C. Petrioli, R. Petroccia, and D. Spaccini, "Implementation of an underwater acoustic network using multiple heterogeneous vehicles," in *Proceedings of MTS/IEEE OCEANS 2012*, Hampton Roads, Virginia, October, 14–19 2012.

[23] "Isee," Last time accessed: March 2013. [Online]. Available: http://www.isee.biz/products/processor-boards/igepv2-board

[24] G. Toso, R. Masiero, P. Casari, O. Kebkal, M. Komar, and M. Zorzi, "Field experiments for dynamic source routing: S2C EvoLogics modems run the SUN protocol using the DESERT Underwater libraries," in *Proc. of MTS/IEEE OCEANS*, Hampton Roads, VA, Oct. 2012.

[25] F. Favaro, F. Guerra, and M. Zorzi, "Data upload from a static underwater network to an AUV: Polling or random access?" in *Proceedings of MTS/IEEE OCEANS 2012*, Yeosu, Korea, May, 21–24 2012.

[26] S. Azad, P. Casari, F. Guerra, and M. Zorzi, "On ARQ strategies over random access protocols in underwater acoustic networks," in *Proc. of IEEE/OES Oceans*, Santander, Spain, May 2011.

[27] F. Guerra, P. Casari, and M. Zorzi, "World ocean simulation system (WOSS): a simulation tool for underwater networks with realistic propagation modeling," in *Proceedings of the Fourth ACM International Workshop on UnderWater Networks*, ser. WUWNet '09, Berkeley, California, USA, 3 November 2009, pp. 1–8.

[28] I. Calabrese, R. Masiero, P. Casari, L. Vangelista, and M. Zorzi, "Embedded systems for prototyping underwater acoustic networks: The DESERT underwater libraries on board the PandaBoard and NetDCU," in *Proceedings of MTS/IEEE OCEANS 2012*, Hampton Roads, Virginia, October, 14–19 2012.

[29] R. Urick, *Principles of Underwater Sound*. McGraw-Hill, 1983.

[30] M. Porter *et al.*, "Bellhop code," Last time accessed: March 2013. [Online]. Available: http://oalib.hlsresearch.com/Rays/index.html

[31] L. Brown, K. A. Karasyov, V. P. Lebedev, A. Y. Starikovskiy, and R. P. Stanley, "Linux laptop battery life: Measurement tools, techniques, and results," in *Proceedings of the Linux Symposium - Volume One*, Ottawa, Ontario, Canada, July 19–22 2006, pp. 127–146, Last time accessed: March 2013. [Online]. Available: http://www.linuxsymposium.org/2006/linuxsymposium_procv1.pdf

[32] "Monsoon solutions inc. monsoon power monitor." Last time accessed: March 2013. [Online]. Available: http://www.msoon.com/