DESERT Underwater: an NS–Miracle-based framework to DEsign, Simulate, Emulate and Realize Test-beds for Underwater network protocols

Riccardo Masiero[§], **Saiful Azad**[§], **Federico Favaro**[⋆], **Matteo Petrani**[§], **Giovanni Toso**[⋆], **Federico Guerra**[⋆], **Paolo Casari**[§⋆], **Michele Zorzi**[§⋆]

[§]DEI, University of Padova, via Gradenigo 6/B – 35131, Padova, Italy
[⋆]CFR, Consorzio Ferrara Ricerche, via Saragat 1 – 44122, Ferrara, Italy

***Abstract*—DESERT Underwater (short for DEsign, Simulate, Emulate and Realize Test-beds for Underwater network protocols) is a complete set of public C/C++ libraries to support the design and implementation of underwater network protocols. Its creation stems from the will to push the studies on underwater networking beyond simulations. Implementing research solutions on actual devices, in fact, is of key importance to realize a communication and networking architecture that allows heterogeneous nodes to communicate reliably in the underwater environment. In this paper, we first discuss the rationale behind this work, and, then we list and briefly describe all the DESERT Underwater libraries currently implemented. In line with the current trends in underwater networking, our approach makes it possible to reuse the same code prepared for simulations in order to realize underwater network prototypes. We also present some preliminary tests that confirm the feasibility of the proposed solution for the design and evaluation of underwater network protocols. In this perspective, we believe that DESERT Underwater is a useful tool to profitably develop and test real world applications.**

***Index Terms*—Underwater networks, simulation, emulation, test-bed, NS-Miracle, WOSS.**

I. Introduction

Ocean sensing and monitoring via underwater acoustic networks is fostering a lot of interest in the research community, as it can provide key information about the mechanisms that regulate our planet, as well as the ability to effectively survey water, sea floors, and coasts on a large scale, in support to various kinds of missions. Recent advances in robotics, acoustic modems, and advanced control, as well as the innovations expected in the near future, provide most of the ingredients required for the realization of such tasks. One of the missing key enablers for any practical application is, however, a communication and networking architecture that allows heterogeneous nodes to communicate effectively and reliably in the harsh underwater environment. When pursuing the latter goal, researchers need to easily simulate and prototype their protocol solutions, as well as to share the obtained results and allow others to easily repeat the same experiments. A flexible, reliable and publicly accessible tool for performance evaluation is of fundamental importance to test and improve the design of network protocols. Based on the well known and widespread network simulator ns2 [1], DESERT Underwater aims at becoming a useful tool to DEvelop, Simulate, Emulate and Realize Test-beds for Underwater network protocols for the research community interested in the applications of underwater acoustic communications.

The main objective of our work is the realization of a complete set of public C/C++ libraries [2] for supporting the design and implementation of underwater network protocols. In this perspective, DESERT Underwater will extend the NS-Miracle [3] simulation software library, developed at the University of Padova, in order to provide several protocol stacks for underwater networks, as well as the support routines required for the development of new protocols.

NS-Miracle enhances the network simulator ns2 with an engine for handling cross-layer messages and, at the same time, for enabling the co-existence of multiple modules within each layer of the protocol stack. In fact, NS-Miracle shows a high modularity and has been designed to simulate nodes whose logical architecture is as close as possible to what would be found on actual devices.

The set of libraries called World Ocean Simulation System (WOSS) [4] endows the network simulator with the chance to simulate the desired protocols over realistic

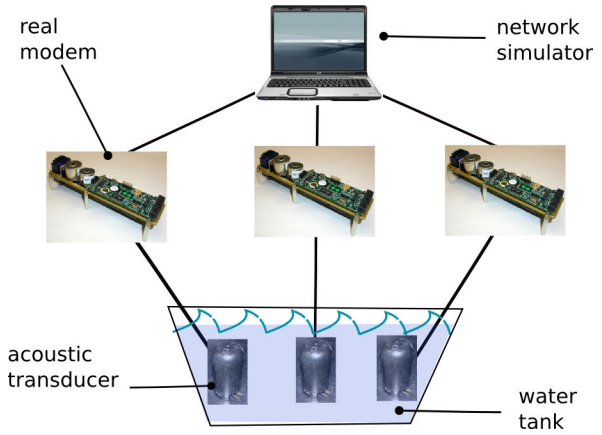

Fig. 1. Illustration of the EMULATION setting: a single host (or a single NS instance) controls multiple modems.

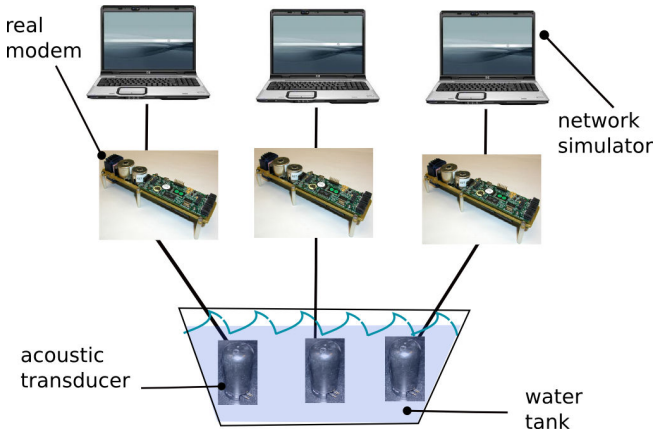

Fig. 2. Illustration of the TEST-BED setting: each modem is controlled by a single host (or a single NS instance).

underwater acoustic channel realizations, and has also been developed based on NS-Miracle. The design of protocol solutions for underwater networks in NS-Miracle yields two main advantages: i) the developers can reuse a lot of code already written for ns2 with minor modifications, and exploit the modularity of NS-Miracle to better organize the design of their solutions; ii) it is possible to evaluate the performance of the designed protocol stack via simulations that employ accurate channel model tools such as WOSS (introduced above). Moreover, DESERT Underwater will make it possible to evolve from pure simulation towards the realization of actual prototypes by framing the hardware of real acoustic modems into NS-Miracle itself. In line with recent papers such as [5], [6], the idea is to wrap all the commands required to communicate with the modem hardware within an NS-Miracle module. In this perspective, the developer can rely on two supported experimental settings: i) a (small-scale) EMULATION setting, where multiple acoustic modems are connected with a single device (e.g., PC, laptop) and controlled by a single NS-Miracle process as illustrated in Fig. 1; ii) a TEST-BED setting, where each acoustic modem is controlled by its corresponding unique device (or by a unique NS-Miracle instance, independent of other instances), as depicted in Fig. 2.

The rest of this paper is organized as follows. The different DESERT Underwater libraries are presented in Section II; in Section III we discuss preliminary tests that confirm the feasibility of a network prototype exploiting the code of DESERT; Section IV concludes the paper.

II. The DESERT Underwater libraries

In this section we summarize and briefly describe all the NS-Miracle modules that compose the first release of the DESERT Underwater libraries [2]. The objective is to provide a clear picture of the currently available DESERT protocols, in the form of an accessible list of modules grouped according to the stack layers defined by the TCP/IP standard. The presentation follows a top-down approach, i.e., it starts from the upper layers (Application and Transport layers), and proceeds through the Network and Data Link layers. Finally, the modules implemented for the Physical layer are illustrated: these include the interface between the network simulator and the actual modem hardware. At the end of the section, we also illustrate four additional modules implemented to simulate node mobility for underwater network scenarios.

To distinguish NS-Miracle modules that belong to DESERT from those coming from different libraries (e.g., the original NS-Miracle libraries or the libraries of WOSS), all the modules in DESERT are named with a prefix `UW-`. This prefix, however, does not mean that the protocol solution implemented by a given module is optimized for underwater networking (as a matter of fact, some modules are the same as their terrestrial radio counterparts, as is the case for the UDP module). Since ns2, and therefore NS-Miracle, is a simulator based on two different languages (C/C++ for module development and Tcl/OTcl for parameter settings[1]), generally we can refer to a given module by means of three names: i) the name of the module (which corresponds to the name of the folder that contains the C/C++ source files); ii) the name of the corresponding dynamic library that must be loaded before using the module itself, and iii) the

[1]Tcl is a scripting language that allows simple access to a set of library functions, whilst OTcl is an extension of Tcl to provide object-oriented functionality.

name[2] of the corresponding OTcl object (that we must use in the parameter configuration file to create the module itself). Since we need to know all three names in order to use a given NS-Miracle module, in the following we report them for all the presented modules. If not otherwise specified, all the DESERT modules implemented for communication protocols can be used for simulation as well as for both emulation and test-bed purposes.

A. Modules for the Application Layer

Currently, in DESERT there are two modules for the application layer: the `uwcbr` and the `uwvbr` modules, detailed in the following. Both modules send dummy packets (i.e., with a random payload) and serve to generate network traffic according to two different mechanisms. To work correctly, all the modules of the application layer must be connected to a module of the transport layer (for the technical details on how to connected two or more NS-Miracle modules, the reader may refer to the NS-Miracle documentation [3]).

Name: `uwcbr`
Description: This module implements a Constant Bit Rate (CBR) packet traffic between a sender and a receiver. The data traffic can be generated either by injecting packets in the network with a constant time period or according to a Poisson process with given mean. A single `uwcbr` module represents a data flow between a pair of nodes: if there are two or more nodes transmitting to the same destination, the latter should have an equal number of `uwcbr` modules, one for each flow.
Library name: `libuwcbr.so`
Tcl name: `Module/UW/CBR`

Name: `uwvbr`
Description: This module implements a Variable Bit Rate (VBR) packet traffic between a sender and a receiver. The data packet generation process takes place by switching between two different CBR processes, e.g., having different average packet inter-arrival times. The switch between the processes can be configured by the user by providing the switching epochs. Otherwise, the simulator can be instructed to switch at constant or exponentially distributed intervals. A single `uwvbr` module represents a data flow between a pair of nodes: if there are two or more nodes transmitting to the same node, the latter should have an equal number of `uwvbr` modules, one for each flow.
Library name: `libuwvbr.so`
Tcl name: `Module/UW/VBR`

[2]Technically speaking, this is the name of the "shadow" OTcl object associated with the C++ object implemented in the source files. For more details, we refer the reader to the ns2 manual [1].

B. Modules for the Transport Layer

In DESERT Underwater, 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 main features of these two modules are detailed below.

Name: `uwudp`
Description: This module implements the flow multiplexing and demultiplexing from and to the upper layers, respectively. It does not support link reliability, error detection or flow control.
Library name: `libuwudp.so`
Tcl name: `Module/UW/UDP`

Name: `uwtp`
Description: As the previous `uwudp`, this transport layer module handles the multiplexing and demultiplexing of data flows, but it also supports an error control technique, as well as in-order data delivery to the upper layers. In more detail, `uwtp` includes an Automatic Repeat reQuest (ARQ) error control technique. This module can handle both ACK and NACK messages, as well as cumulative ACKs. To work correctly, `uwtp` has to know the destination port number of each application flow it handles: this information must be provided during the setting of the parameters for the simulations or tests that have to be conducted.
Library name: `libuwtp.so`
Tcl name: `Module/UW/TP`

C. Modules for the Network Layer

The Network Layer is in charge of providing tools for the network interfaces (e.g., addresses) and mechanisms for data routing. In DESERT, we have developed three algorithms for routing and a simple module to manage network addresses whose format is compliant with the IP standard. The details of these modules follow.

Name: `uwstaticrouting`
Description: This module makes it possible to simulate and test data traffic which has to follow predetermined routes. For each network node, there is an option to choose a default gateway and/or fill a static routing table (whose maximum size is hard-coded and fixed to 100

entries). This information is then exploited locally at each node to forward the network packets, hop by hop, throughout the predetermined paths.
Library name: `libuwstaticrouting.so`
Tcl name: `Module/UW/StaticRouting`

Name: `uwsun`
Description: This module implements a dynamic, reactive source routing protocol. The generation of routing paths can be made based on different criteria, such as the minimization of the hop-count or the maximization of the minimum Signal to Noise Ratio (SNR) along the links of the path. `uwsun` is also designed to collect and process different statistics of interest for the routing level. Currently, this module supports all application modules provided in DESERT (i.e., `uwcbr` and `uwvbr`), and it can be easily extended.
Library name: `libuwsun.so`
Tcl name: `Module/UW/SUNNode` for the nodes; `Module/UW/SUNSink` for the sinks.

Name: `uwicrp`
Description: This module, which requires very few configuration parameters, implements a simple flooding-based routing mechanism called Information-Carrying Based Routing protocol, see [7].
Library name: `libuwicrp.so`
Tcl name: `Module/UW/ICRPNode` for the nodes; `Moudle/UW/ICRPSink` for the sinks.

Name: `uwip`
Description: This module is used to assign an address to the nodes in a given network according to the standard IPv4 addresses; it provides the Time-To-Live (TTL) functionality and does not implement any routing mechanism. It can be configured to provide all the functional and procedural means intended for an Internet Protocol module (e.g., fragmentation, data reassembly and notification of delivery errors).
Library name: `libuwip.so`
Tcl name: `Module/UW/IP`

D. Modules for the Data Link Layer

The core of the data link layer is the Medium Access Control (MAC), that administrates the access to the acoustic communication channel. The DESERT Underwater libraries provide six modules which implement as many MAC techniques: `uwaloha`, `uwsr`, `uw-csma-aloha`, `uwdacap`, `uwpolling` and `uw-t-lohi`, all explained in the following. Additionally, DESERT provides `uwmll`, a module to map IP addresses to their corresponding MAC addresses.

Name: `uwmll`
Description: Since node-to-node communications at the link layer are performed using MAC addresses whereas the communications at the upper layers employ IP addresses, a method to associate the latter to the former is required. With `uwmll`, it is possible to set the correspondence between IP and MAC addresses a priori, by filling an ARP table for each network node. Alternatively, ARP tables can be automatically filled using the Address Resolution Protocol (ARP). The `uwmll` module must be placed between one (or more) IP module(s) and one MAC module.
Library name: `libuwmll.so`
Tcl name: `Module/UW/MLL`

Name: `uwaloha`
Description: ALOHA is a random access scheme, i.e., a protocol that allows nodes to send data packets directly without any preliminary channel reservation process. In its original version [8], neither channel sensing nor retransmission is implemented and each node can transmit whenever it has data packets to send. As a consequence, packet losses can occur. In later adaptations [9], ALOHA has been enhanced with acknowledgment packets (ALOHA-ACK). The `uwaloha` module implements the functionality of the basic ALOHA protocol as well as its enhanced version using ARQ for error control. It is possible to freely switch between basic ALOHA and ALOHA-ACK.
Library name: `libuwaloha.so`
Tcl name: `Module/UW/ALOHA`

Name: `uw-csma-aloha`
Description: This module implements ALOHA-CS [9], an enhanced version of the ALOHA protocol introduced above. ALOHA-CS adds a carrier sensing mechanism to basic ALOHA, in order to help reduce the occurrence of collisions.
Library name: `libuwcsmaaloha.so`
Tcl name: `Module/UW/CSMA_ALOHA`

Name: `uwdacap`
Description: This module implements DACAP (Distance Aware Collision Avoidance Protocol) [10], which provides a collision avoidance mechanism via a handshake phase prior to packet transmission. This phase involves the exchange of signaling packets such as Request-To-Send (RTS) and Clear-To-Send (CTS). This protocol in-

troduces also a very short warning packet in the RTS-CTS mechanism to further prevent collisions among nodes. DACAP is designed for underwater networks with long propagation delays and can be implemented with and without acknowledgements; `uwdacap` implements both solutions.
Library name: `libuwdacap.so`
Tcl name: `Module/UW/DACAP`

Name: `uwpolling`
Description: This module implements a MAC protocol which is based on a centralized polling scheme. To fix ideas, focus on an AUV patrolling an area covered by an underwater sensor field; the AUV coordinates the data gathering from the sensors in a centralized fashion using a polling mechanism. This mechanism is based on the exchange of three types of messages: a broadcast TRIGGER message, that the AUV sends to notify the sensor nodes of its presence; a PROBE message, that the sensors use to answer the initial TRIGGER message; and a POLL message, sent again by the AUV and containing the order in which the underwater nodes can access the channel to communicate their data. This order of polling is determined by the AUV, given the information collected from the PROBE messages which may include, among others, such metrics as the residual energy of the nodes, the timestamp of the data to be transmitted or a measure of their priority. Because of its nature, the algorithm implemented by `uwpolling` does not require any routing mechanism on top of it.
Library name: `libuwpolling.so`
Tcl name: `Module/UW/POLLING_AUV` for the AUV; `Module/UW/POLLING_NODE` for the sensor nodes.

Name: `uw-t-lohi`
Description: Tone-Lohi (T-Lohi) [11] is a MAC protocol that uses tones during contention rounds to reserve the channel. Other nodes competing for channel access are detected during a contention round, by listening to the channel after sending the reservation tone. T-Lohi takes advantage of the long propagation delays present in underwater networks to count the number of contenders reliably, and act accordingly during the contention round. Tone-Lohi can be: 1) synchronized (ST-Lohi); 2) conservative unsynchronized (cUT-Lohi), which enables the counting of all contenders by extending the duration of the contention round to twice the maximum expected propagation delay, and 3) aggressive unsynchronized (aUT-Lohi), to reduce idle times (while increasing the probability of collisions). The `uw-t-lohi` module implements all of three versions of T-Lohi above; the user can freely choose which one to enable. However, since the possibility of transmitting actual tones depends on the available hardware, this module has not been considered for emulation and test-bed; differently, it can be exploited for simulation purposes using WOSS.
Library name: `libuwtlohi.so`
Tcl name: `Module/UW/TLOHI`

Name: `uwsr`
Description: Automatic Repeat reQuest (ARQ) is the basic mechanism to ensure that no erroneous packets are delivered to layers higher than the data-link. When a packet is received with errors, the receiver may ask for retransmissions by sending a small packet called NACK (Negative ACKnowledgment). Similarly, a successful transmission can be notified to the transmitter via an ACK packet. Selective Repeat ARQ allows the transmitter to send up to N consecutive packets before waiting for ACKs or NACKs. The packets sent and not yet acknowledged must be buffered by the transmitter; the receiver can also buffer packets and, in case of errors, only the erroneous packets are sent again. `uwsr` implements a Selective Repeat ARQ mechanism in combination with an Additive Increase and Multiplicative Decrease (AIMD) congestion control technique, similar to TCP's congestion window size adaptation. This protocol has been shown to be effective because the underwater channel propagation delay is sufficiently large to accommodate more than one packet transmission within one round-trip time (RTT) [12].
Library name: `libuwsr.so`
Tcl name: `Module/UW/USR`

E. Modules for the Physical Layer: interfaces for real acoustic modem hardware

Currently, DESERT is supporting three different hardware platforms for emulation and test-bed realization: the S2C hydroacoustic modem, a system developed by Evologics [13] which exploits the Sweep-Spread Carrier (S2C) technology for underwater data telemetry and communications; the FSK and PSK WHOI micro-modems, two small-footprint, low-power acoustic modems based on a Texas Instruments DSP and developed by the Woods Hole Oceanographic Institution (WHOI) [14]. However, the `uwmphy_modem` module, which implements the general interface between the acoustic modem hardware and NS-Miracle, can be easily extended to support other different hardware. In the following, we describe `uwmphy_modem`, along with the DESERT modules

specialized for the above hardware (mfsk_whoi_mm, mpsk_whoi_mm and ms2c_evologics), as well as an additional module called uwmphypatch that is intended for preparing the OTcl scripts needed to set the parameters of the networking prototypes to test.

Name: uwmphypatch
Description: uwmphypatch is a dumb module to patch the absence of a physical layer when a MAC module is used. It just receives and forwards a packet handling the cross-layer messages required by all the MAC layers of DESERT. The main aim of this module is to observe the behavior of a given network protocol over an ideal channel, before interfacing the network simulator engine with real hardware. It should be used in conjunction with the underwater channel or the dumb wireless channel provided by the NS-Miracle libraries, and mainly to gather insight about the mechanisms of the investigated network protocol (independently of the errors that can be introduced by the channel). Any usage related to performance evaluation is not recommended.
Library name: libuwmphypatch.so
Tcl name: Module/UW/MPhypatch

Name: uwmphy_modem
Description: This module defines and implements the general interface between ns2/NS-Miracle and real acoustic modems. uwmphy_modem manages all the messages needed by NS-Miracle (e.g., cross layer messages between MAC and PHY layers) and contains all the simulation parameters that can be set by the user, along with the methods to change them. This module is an abstract class that must be used as base class for any derived class that interfaces NS-Miracle with a given hardware. Therefore, neither an actual object for this module nor its corresponding shadowed Tcl object can be created, hence there is no "Tcl name" associated to this module.
Library name: libuwmphy_modem.so
Tcl name: —

Name: mfsk_whoi_mm
Description: Module derived from uwmphy_modem to implement the interface between ns2/NS-Miracle and the FSK WHOI micro-modem.
Library name: libmfsk_whoi_mm.so
Tcl name: Module/UW/MPhy_modem/FSK_WHOI_MM

Name: mpsk_whoi_mm
Description: Module derived from uwmphy_modem to implement the interface between ns2/NS-Miracle and the PSK WHOI micro-modem.
Library name: libmpsk_whoi_mm.so
Tcl name: Module/UW/MPhy_modem/PSK_WHOI_MM

Name: mgoby_whoi_mm
Description: Module derived from uwmphy_modem to implement the interface between ns2/NS-Miracle and the WHOI micro-modems (both the FSK and PSK version) using the Goby software [15] to handle the connection with the modems.
Library name: libmgoby_whoi_mm.so
Tcl name: Module/UW/MPhy_modem/GOBY_WHOI_MM

Name: MS2C_EvoLogics
Description: Module derived from uwmphy_modem to implement the interface between ns2/NS-Miracle and the S2C EvoLogics modem.
Library name: libmstwoc_evologics.so
Tcl name: Module/UW/MPhy_modem/S2C

F. Additional modules to simulate mobility

When underwater networks are simulated, the tested solution should be investigated using accurate models that encompass, among other things, realistic mobility patterns. The DESERT libraries also include four modules to simulate node mobility in 2D as well as 3D scenarios (since in underwater environments nodes can move along any direction in space): uwdriftposition and uwgmposition are stand-alone mobility modules, whereas both wossgmmob3D and wossgroupmob3D require the installation of WOSS to work. All of them are detailed in the follows.

Name: uwdriftposition
Description: This module implements a mobility model that mimics the drift of a node caused by ocean currents. Given the mean speed and direction of the waves, the initial node's position and its velocity, uwdriftposition continuously updates the node's location in order to follow the direction of the current. At each update, the new node position is also affected by a random noise that aims at reproducing the waving movement typical of objects floating in the water.
Library name: libuwdriftposition.so
Tcl name: Position/UW/DRIFT

Name: uwgmposition
Description: This module implements the Gauss-Markov Mobility Model [16] (both in 2D and 3D), a solution designed to produce smooth and realistic traces by appropriately tuning a correlation parameter α. When required,

`uwgmposition` updates node speed and direction according to a finite state Markov process. Once the desired mean speed v_{mean} is fixed, α controls the correlation between the speed vector and direction at state k and that at $k-1$.
Library name: `libuwgmposition.so`
Tcl name: `Position/UW/GM`

Name: `wossgmmob3D`
Description: This module also implements the Gauss Markov Mobility Model, but with some changes that make it directly usable with WOSS. For example, WOSS employs the geographic coordinate system (i.e., latitude, longitude and altitude/depth) for the positions of the nodes; therefore, `wossgmmob3D` also adopts the geographical coordinate system to describe the node movements.
Library name: `libwossgmmobility.so`
Tcl name: `WOSS/GMMob3D`

Name: `wossgroupmob3D`
Description: This module implements a leader-follower paradigm (also known as "group mobility model"). According to this model, we have: i) a leader node, that moves either randomly (i.e., according to a Gauss-Markov Mobility Model) or by following a pre-determined path, and ii) one or more followers that tune their movements so as to mimic the route of the leader. The movement of the followers is generated as the sum of two components: a movement that attracts the follower towards the leader and a random movement. The first one is obtained according to the mathematical model that describes the attraction between two electrical charges [17], whereas the second one is still based on a Gauss-Markov Mobility Model. Three parameters regulate the attractive component of the overall movement of a follower: the "charge of the leader", the "charge of the follower", and a third parameter α which is used to determine the intensity of the "attraction field"; in particular, a negative value of α attracts the follower towards the leader, whereas a positive value pushes the follower away. Like `wossgmmob3D`, also `wossgroupmob3D` supports 3D mobility and, currently, can be used only in conjunction with WOSS.
Library name: `libwossgroupmobility.so`
Tcl name: `WOSS/GroupMob3D`

III. Emulation and Testbed settings: A first feasibility test

As illustrated in the previous section, the DESERT Underwater libraries currently provide interfaces between the NS-Miracle network simulator and two commercial modems: the WHOI Micro-Modem, developed by the Woods Hole Oceanographic Institution, and the S2C acoustic modem, manufactured by EvoLogics. To thoroughly test and improve the designed interfaces, we are planning to realize extensive experimental campaigns in collaboration with both partners. However, some preliminary tests have already been performed to show the feasibility of the adopted solution.

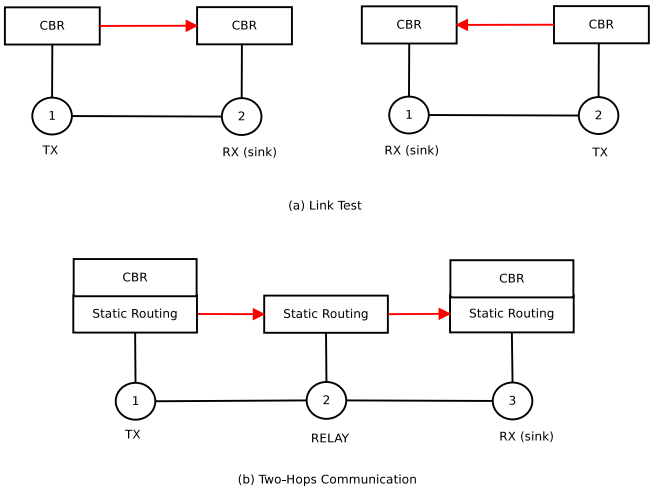

Fig. 3. Sketches of the performed feasibility tests: (a) bidirectional Link Test (from node 1 to node 2 and vice-versa); (b) Two-Hop Communication.

Part of these tests have been conducted at the Department of Information Engineering of the University of Padova, using three FSK WHOI Micro-Modems. These modems work in the 3-30 kHz frequency range, have a data rate of 80 bit/s and, when powered at 36 V, can reach a working range of up to 2 km horizontally and up to 9 km vertically. They can be controlled using the NMEA [18] communications standard to handle both host-to-modem and modem-to-modem communications. The communication software also provides the use of "minipackets", that have been exploited during our experiments, as they do not require the preliminary transmission of control packets (e.g., initialization messages compliant with the NMEA standard).

A schematic representation of our first feasibility tests is reported in Fig. 3: (a) first, we have successfully realized a point to point communication, both in the EMULATION and TEST-BED setting; (b) then, in the TEST-BED setting, we have realized a two-hop communication between nodes 1 and 3, using node 2 as a relay. In detail, using the engine of NS-Miracle, each node has been created according to a simple but complete protocol

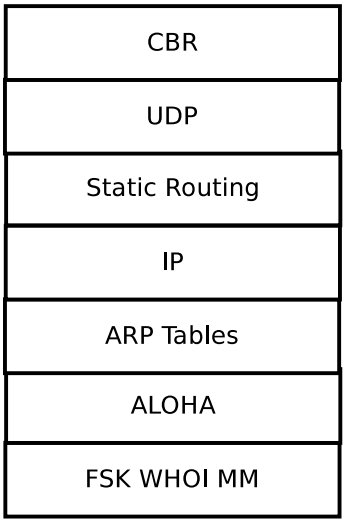

Fig. 4. Illustration of the protocol stack used during the feasibility tests.

stack using the following modules: `uwcbr`, `uwudp`, `uwstaticrouting`, `uwip`, `uwmll`, `uwaloha` and `mfsk_whoi_mm` (see Fig. 4). At the transmitter, each data packet is generated by the CBR application; the packet is then sent down through several layers of the chosen protocol stack according to the engine of the network simulator; the information contained in each packet and in its header is then compressed to fit in the Micro Modem minipacket payload using the `mfsk_whoi_mm` module; finally, the minipacket is transmitted over the acoustic channel. At the destination, each received packet follows the reverse path: it is re-generated starting from the payload of the minipacket and mapped into the corresponding data structure of the network simulator; finally, it is sent through the whole protocol stack, up to the application layer of the destination node.

While there is indeed a functional difference between the EMULATION and the TEST-BED setting, the only technical difference between the two approaches is the fitting of NS-Miracle packets into modem payloads and the reverse operation. In the EMULATION setting, in fact, multiple acoustic modems are connected with a single device and therefore packets can be exchanged among the nodes using the same mechanism as in the network simulator (in our implementation, this is accomplished by transmitting a pointer to the data structure containing all the needed packet fields). In the TEST-BED setting, instead, we need to send all the information necessary to rebuild, at the receiver side, the original NS-Miracle packet as it had been created and modified by the network simulator running on the transmitter host.

TABLE I
TRACE FILE LOGGED DURING THE FEASIBILITY TEST (A) IN FIGURE 3 (EMULATION SETTING). NODE 1 IS THE TRANSMITTER AND NODE 2 IS THE RECEIVER.

```
s 7.025577068 1 CBR UDP SN=1
s 7.025577068 1 UDP Static_Routing SN=1
s 7.025577068 1 Static_Routing IP SN=1
s 7.025577068 1 IP ARP_Tables SN=1
s 7.025577068 1 ARP_Tables ALOHA SN=1
s 7.025723219 1 ALOHA FSK_WHOI_MM SN=1
r 10.015380144 2 FSK_WHOI_MM ALOHA SN=1
r 10.015380144 2 ALOHA ARP_Tables SN=1
r 10.015380144 2 ARP_Tables IP SN=1
r 10.015380144 2 IP Static_Routing SN=1
r 10.015380144 2 Static_Routing UDP SN=1
r 10.015380144 2 UDP CBR SN=1
s 14.025732994 1 CBR UDP SN=2
s 14.025732994 1 UDP Static_Routing SN=2
s 14.025732994 1 Static_Routing IP SN=2
s 14.025732994 1 IP ARP_Tables SN=2
s 14.025732994 1 ARP_Tables ALOHA SN=2
s 14.025867224 1 ALOHA FSK_WHOI_MM SN=2
r 17.018894196 2 FSK_WHOI_MM ALOHA SN=2
r 17.018894196 2 ALOHA ARP_Tables SN=2
r 17.018894196 2 ARP_Tables IP SN=2
r 17.018894196 2 IP Static_Routing SN=2
r 17.018894196 2 Static_Routing UDP SN=2
r 17.018894196 2 UDP CBR SN=2
...
```

TABLE II
TRACE FILE FOR THE FEASIBILITY TEST (A) IN FIGURE 3 (TEST-BED SETTING) LOGGED AT NODE 1 (TRANSMITTER).

```
s 7.012847900 1 CBR UDP SN=1
s 7.012847900 1 UDP Static_Routing SN=1
s 7.012847900 1 Static_Routing IP SN=1
s 7.012847900 1 IP ARP_Tables SN=1
s 7.012847900 1 ARP_Tables ALOHA SN=1
s 7.013003111 1 ALOHA FSK_WHOI_MM SN=1
s 14.013986111 1 CBR UDP SN=2
s 14.013986111 1 UDP Static_Routing SN=2
s 14.013986111 1 Static_Routing IP SN=2
s 14.013986111 1 IP ARP_Tables SN=2
s 14.013986111 1 ARP_Tables ALOHA SN=2
s 14.014182091 1 ALOHA FSK_WHOI_MM SN=2
...
```

In Tables I–IV, we report portions of typical trace-files obtained for the feasibility tests illustrated in Fig. 3, when data packets have been sent from the source to the destination with a packet inter-arrival period of 7 s. These files trace the packet transition from one layer of the protocol stack (identified by the corresponding NS-Miracle module) to the subsequent one; they are organized in six columns, with the following meaning: 1) packet origin (sent packet, “s” or received packet, “r”);

TABLE III
TRACE FILE FOR THE FEASIBILITY TEST (A) IN FIGURE 3 (TEST-BED SETTING) LOGGED AT NODE 2 (RECEIVER).

```
r 12.013406992 2 FSK_WHOI_MM ALOHA SN=1
r 12.013406992 2 ALOHA ARP_Tables SN=1
r 12.013406992 2 ARP_Tables IP SN=1
r 12.013406992 2 IP Static_Routing SN=1
r 12.013406992 2 Static_Routing UDP SN=1
r 12.013406992 2 UDP CBR SN=1
r 19.020930052 2 FSK_WHOI_MM ALOHA SN=2
r 19.020930052 2 ALOHA ARP_Tables SN=2
r 19.020930052 2 ARP_Tables IP SN=2
r 19.020930052 2 IP Static_Routing SN=2
r 19.020930052 2 Static_Routing UDP SN=2
r 19.020930052 2 UDP CBR SN=2
...
```

2) simulation time at which the packet transition between layers has occurred; 3) node ID; 4) source DESERT module; 5) destination DESERT module; 6) sequence number of the data packet. Table I and Tables II–III refer to the feasibility test (a), for the EMULATION and the TEST-BED setting, respectively. In the case of EMULATION, we collected all the traces in a single file (as commonly done when we run simulations using NS-Miracle); in the case of TEST-BED, instead, the traces have been created independently for each modem by the corresponding hosts (or ns2 process). In both cases, we can observe that all the considered protocol stacks have been correctly traversed by the exchanged packets, at the transmitter as well as at the receiver (the first packet is rendered using a red boldface font, both at the transmitter and at the receiver). Note also that the simulation time elapsed between the transmission of a given packet and its reception in the TEST-BED case increases with respect to the EMULATION case (from about 3 to about 5 s); this is due to two issues related to the TEST-BED case: the asynchronous simulation timers of the two nodes (during these TEST-BED tests, simulations have been launched manually and always starting from the receiver), and the increased complexity for mapping and de-mapping NS-Miracle packets into modem payloads (even though the delay introduced by this second issue is negligible with respect to the first one). Table IV, instead, shows a portion of the trace file generated for the relay node 2 during the feasibility test (b). In this case, we can observe the behavior that we imposed for the routing protocol at the network layer: since node 2 must act as a relay, when it receives a packet for node 3 from node 1, this packet is propagated only up to the network layer and then immediately sent down to be forwarded to its intended destination.

TABLE IV
TRACE FILE OBTAINED FOR THE FEASIBILITY TEST (B) IN FIGURE 3 (TEST-BED SETTING) LOGGED AT NODE 2 (RELAY NODE).

```
r 11.011646986 2 FSK_WHOI_MM ALOHA SN=1
r 11.011646986 2 ALOHA ARP_Tables SN=1
r 11.011646986 2 ARP_Tables IP SN=1
r 11.011646986 2 IP Static_Routing SN=1
s 11.011646986 2 Static_Routing IP SN=1
s 11.011646986 2 IP ARP_Tables SN=1
s 11.011646986 2 ARP_Tables ALOHA SN=1
s 11.011960030 2 ALOHA FSK_WHOI_MM SN=1
...
```

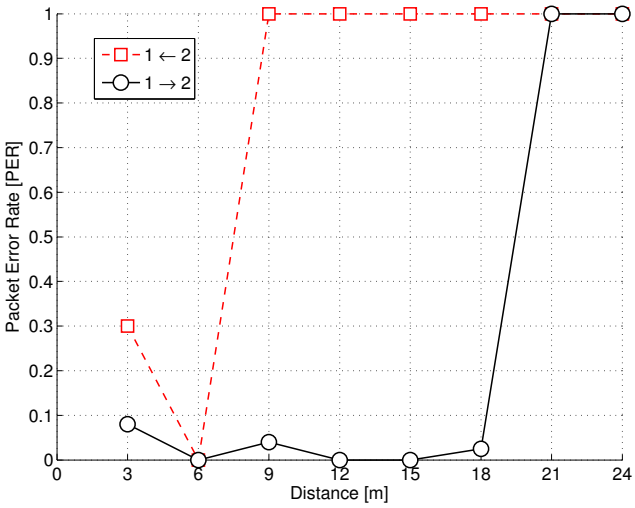

Fig. 5. Packet Error Rate observed during the field experiment in the Piovego channel.

To move beyond just feasibility tests done in the laboratory, we also conducted a first field experiment in the Piovego channel which flows near our department. We put the transducers at a distance of 2 m from the bank, where the channel is around 80 to 90 cm depth, with a muddy bottom. We also decreased the supply voltage of the micro-modems to 12 V, in order to decrease their transmission ranges. Using the same script prepared for test (a) in Fig. 3, we have performed several point-to-point transmissions (in the TEST-BED setting) varying the distance between nodes: i.e., we kept node 1 fixed and moved node 2 away, in steps of 3 m. For each tested link, we sent 50 packets, separated by a time interval of 4 seconds in both directions (i.e., from node 1 to node 2 and vice-versa).

Fig. 5 shows the observed packet error rate (PER) as a function of the distance between the sender and the receiver, in both directions (note that, as expected, the

observed underwater channel is not symmetric). Over the acoustic channel we sent FSK packets with an overall length of 32 bits, at a bit rate of 80 bit/s. Despite the adverse conditions (very shallow water, wind-generated surface ripples and noise, proximity to the bank, water turbidity), this test allowed us to verify the feasibility of the DESERT Underwater libraries, when used to drive real modem hardware.

IV. Conclusions

Focusing on underwater applications, in this paper we discussed the chance to evolve from simulations to the realization of underwater network prototypes by interfacing real hardware devices with software network simulators. After having recognized this activity as a key effort to profitably develop and test real world applications, we presented the DESERT Underwater framework. This is a set of public C/C++ libraries based on the NS-Miracle [3] simulation software, developed at the University of Padova. The DESERT Underwater libraries are meant as a flexible and reliable tool to support the design and implementation of underwater network protocols.

We have listed and briefly described all the developed libraries currently available. These libraries are intended to be used jointly, possibly in many different combinations, thus realizing several protocol stacks for underwater networks. We also introduced the general interface designed to integrate the NS-Miracle network simulator with real hardware, as well as its specialized versions for two existing commercial modems.

Finally, we performed some preliminary tests to assess the feasibility of the DESERT Underwater libraries for network prototyping. These tests allow us to successfully perform single-hop as well as two-hop transmissions using the same code implemented in NS-Miracle for simulation purposes. We believe that our work, the public release of the DESERT Underwater libraries [2], and their future development, represent a fundamental step for the study of effective underwater network protocols, moving from simulations to the real world.

V. Acknowledgments

This work has been supported by the Italian Institute of Technology within the Project SEED framework (NAUTILUS project). The authors gratefully thank Federico Beccaro, Achille Forzan, Moreno Zorzetto and Ivano Calabrese for their precious help in the organization and realization of the field experiments.

References

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

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

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

[4] "The World Ocean Simulation System - *WOSS*," Last time accessed: March 2012. [Online]. Available: http://telecom.dei.unipd.it/ns/woss/

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

[6] C. Petrioli, R. Petroccia, and J. Potter, "Performance evaluation of underwater MAC protocols: From simulation to at-sea testing," in *OES/IEEE Oceans*, Santander, Spain, 2011.

[7] W. Liang, H. Yu, L. Liu, B. Li, and C. Che, "Information-carrying based routing protocol for underwater acoustic sensor network," in *Proc. of ICMA*, Takamatsu, Kagawa, Japan, Aug. 2007.

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

[9] X. Guo, M. Frater, and M. Ryan, "A propagation-delay-tolerant collision avoidance protocol for underwater acoustic sensor networks," in *OES/IEEE Oceans*, Singapore, 2006.

[10] 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, 2007.

[11] A. A. Syed, W. Ye, and J. Heidemann, "T-Lohi: A new class of MAC protocols for underwater acoustic sensor networks," in *IEEE INFOCOM*, Phoenix, AZ, US, Apr. 2008.

[12] S. Azad, P. Casari, F. Guerra, and M. Zorzi, "On ARQ Strategies over Random Access Protocols in Underwater Acoustic Networks," in *OES/IEEE Oceans*, Santander, Spain, 2011.

[13] "Evologics," Last time accessed: March 2012. [Online]. Available: http://www.evologics.de/

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

[15] "The Goby Underwater Autonomy Project - *Goby*," Last time accessed: March 2012. [Online]. Available: http://gobysoft.com/

[16] B. Liang and Z. Haas, "Predictive distance-based mobility management for PCS networks." in *IEEE INFOCOM*, New York, NY, US, Mar. 1999.

[17] L. Badia and N. Bui, "A Group Mobility Model Based on Nodes' Attraction for Next Generation Wireless Networks," in *IEE Mobility'06*, Bangkok, Thailand, Oct. 2006.

[18] "The National Marine Electronics Association - *NMEA*," Last time accessed: March 2012. [Online]. Available: http://www.nmea.org/