Field Experiments for Dynamic Source Routing: S2C EvoLogics modems run the SUN protocol using the DESERT Underwater libraries

Giovanni Toso[⋆], **Riccardo Masiero**[§], **Paolo Casari**[§⋆],
Oleksiy Kebkal[◇], **Maksym Komar**[◇], **Michele Zorzi**[§⋆]

[§]DEI, University of Padova, via Gradenigo 6/B – 35131, Padova, Italy
[⋆]CFR, Consorzio Ferrara Ricerche, via Saragat 1 – 44122, Ferrara, Italy
[◇]EvoLogics GmbH, Ackerstrasse 76 – D-13355, Berlin, Germany

***Abstract*—In this paper we present a performance evaluation and feasibility test of SUN, a routing protocol for underwater networks inspired to Dynamic Source Routing (DSR), to which it adds several features that improve its behavior in underwater environments. The evaluation has been performed with real devices, and has been made possible through a collaboration between the Department of Information Engineering (DEI) of the University of Padova, Italy and EvoLogics GmbH, Germany. In detail, the idea put in practice in this work is to command real hardware, i.e., the S2C acoustic modems of EvoLogics, by means of the ns2/NS-Miracle engine developed and extensively used primarily by research institutions. This approach favors code reuse and speeds up the realization of flexible and easily modifiable network prototypes.**

Our results show that SUN can deal with typical network issues such as the disconnection of a node and the appearance of additional nodes, and that it copes well with dynamic topology changes.

***Index Terms*—Underwater networks, dynamic source routing, emulation, test-bed, DESERT Underwater, NS-Miracle, ns2.**

I. Introduction

Both academia and industry are showing interest in underwater network applications, and in the implementation of research-level communication solutions on actual devices. Testing different network protocols and/or physical layer solutions in real environments is seen as a valuable way to provide a comprehensive study for the realization of an effective communication technology. On one hand, this activity strengthens the study, as it allows researchers to support theoretic and simulation results via experimentation; on the other hand, it may unveil bottlenecks or practical issues that provide useful feedback for the design of reliable prototypes and, eventually, commercial products. The work presented in this paper moves in this direction by exploiting the integration of the network simulator NS-Miracle [1] (an extension of the well-known network simulator ns2 [2]) with the hardware of the S2C acoustic underwater modems.

The S2C hydro-acoustic modem is a system developed by Evologics [3] which exploits the Sweep-Spread Carrier (S2C) technology for underwater data communications and telemetry. It operates with two different frequency ranges (18–34 kHz or 48–78 kHz) and can reach data rates of up to 33 kbit/s. The S2C modem can transmit acoustically up to 2 km (optionally up to 6 km) and can operate at depths up to 100 m (optionally up to 6 km). The S2C modem can be controlled via direct host/modem communication using the standard AT command set. Furthermore, it features the integration of the NS-Miracle network simulator, so that we can also exploit the DESERT Underwater libraries [4], [5] to control the modem and to organize multiple S2C devices into an underwater network.

DESERT Underwater is a set of public C++ libraries [5] developed by the Department of Information Engineering (DEI) of the University of Padova; these libraries use the engine offered by the NS-Miracle and have been developed to support the design and implementation of underwater network protocols. The rationale behind DESERT Underwater is that it should be possible to evolve from simulations to the realization of actual prototypes, by reusing the same software already written for simulations to the largest extent. In this light, the idea is to make it possible for NS-Miracle modules to command acoustic modems. This approach favors not only code reuse, but also the realization of flexible and easily modifiable network prototypes (the software developed in DESERT Underwater is designed to be modular and easily adaptable to different network configurations, communication protocols and application scenarios).

Along with the modules that interface the network simulator with real hardware, the DESERT Underwater libraries provide modules that implement various communication protocols at different layers of the protocol stack. In this work, we focus on a network layer module that implements SUN, a dynamic, reactive Source routing protocol for Underwater Networks. According to this protocol, the generation of routing paths can be performed on-demand by the nodes in the network and 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. A detailed description of the SUN protocol is provided in Section II.

The rest of the paper is organized as follows. In Section III we describe the S2C acoustic modem hardware and firmware protocol stack, and also present the S2CR WiSE edition, a series of modems explicitly designed to work with the ns2/NS-Miracle engine, that we used for our field experiments. In Section IV, we detail the DESERT Underwater module that implements the interface between NS-Miracle and the S2C EvoLogics modem. In Section V we present the evaluation of SUN performed during real-world experiments. Section VI concludes the paper.

II. The Source routing for Underwater Networks (SUN) protocol

SUN is a reactive source routing protocol developed at DEI and inspired to the Dynamic Source Routing (DSR) solution described in [6]. DSR is a routing protocol specifically designed for wireless radio networks and the typically high communication delays that characterize the underwater acoustic channel may severely impair its performance. The SUN protocol aims at overcoming such difficulties and does not rely upon any centralized control unit. Moreover, SUN can be exploited in both static and mobile networks and is oblivious of the network topology: so long as the network is connected, SUN does not require any information about node locations or depths. SUN has been designed as a reactive protocol, in order to avoid wasting bandwidth with proactive route discovery procedures.

The SUN protocol is explicitly designed for those applications that entail the presence of two kind of nodes: the sink nodes, that have to gather all the application data generated in the network, and the sensor nodes, that both generate and relay the information of interest (e.g., sampled sensor measurements of some physical phenomenon to monitor). Sink nodes are passive entities that, according to the SUN protocol, perform only two tasks: i) to periodically send probe messages in broadcast, in order to notify their own presence to the nodes within their communications range; ii) to receive data packets. Sensor nodes, instead, can: i) generate and send data packets; ii) request for available routing paths or start a new path discovery; iii) answer path discovery requests; iv) act as relays, i.e., forward incoming data packets toward the chosen path, and v) notify routing problems, such as the loss of contact with a sink or a relay.

Both sinks and sensor nodes communicate according to the SUN protocol via six different packet messages:

1) *Probe*. The packet sent by the sink nodes to notify about their presence. Sensor nodes that receive this message understand to be one-hop neighbors of the sink and are termed “end nodes” in what follows;
2) *Path Establishment-Request*. The messages sent by source nodes to start, if necessary, the discovery of a routing path when application data must be sent over the channel;
3) *Path Establishment-Answer*. The messages used by sensor nodes to reply to the *Path Establishment-Request* messages in order to successfully establish a complete route toward a sink;
4) *Data*. The packet containing the application data of interest. Sensor nodes both generate and forward packets of this kind; sink nodes, instead, can only receive them;
5) *Acknowledgment*. The packets used to notify about the correct reception of a *Data* packet over a point-to-point link. This message can by sent by both sensor and sink nodes;
6) *Route Error*. The packet messages sent by sensor nodes to notify the data packet source about the failure of a known path.

SUN is a source routing protocol. Hence, for each generated data packet, each source node completely specifies the best route towards one of the sinks in the network. This route is chosen according to a metric based on local information provided by each relay. Currently, SUN supports two metrics: the *Lowest Hop Count* and the *MaxMin Signal to Noise Ratio (SNR)*. Implementing a very simple algorithm, the first metric allows a source node to build the routing path with the lowest number of hops thus reducing the number of relays involved in the communication; however, maximizing the link distances may translate into a decrease of the average SNR per link. The second metric, conversely, is slightly more computational demanding and may lead to a higher exchange of *Path Establishment* messages with a consequent increase of interference among nodes. Also, this metric requires the implementation of cross-layer messages to retrieve at the network layer the SNR measured at the physical layer; however, *MaxMin SNR* usually leads to the creation of more reliable routes maximizing the average SNR per link. Which metric to use depends on the actual application and network scenarios. Once selected, the indication of the entire chosen route is reported in the packet header, thus avoiding a hop-by-hop relay election process. To enhance the protocol reliability, SUN also implements the following additional features:

- an internal buffering mechanism, employed to store data packets from upper and lower layers. In case a valid path to the sink is not immediately available, this system gives SUN the possibility of storing packets, sending a *Path Establishment-Request* message and waiting for an answer; when the node receives such answer, it will remove the data packet from the buffer, fill its header with a valid route and send it over the channel. To avoid buffer overflow problems, each stored packet is in any case removed from the buffer if a valid route is not established within a given amount of time (i.e., the packet buffering timeout);
- a Stop-and-Wait Automatic Repeat reQuest (ARQ) mechanism, that performs packet retransmissions at the network level without delegating them to the link control. This gives us the possibility to directly evaluate the state of the network, e.g., unreliable paths, congested links, moving nodes, without implementing specific mechanisms of interaction between the data-link and routing layers, and therefore reducing the complexity of the overall network protocol stack (which is always desirable, especially in underwater scenarios).

To sum up, the SUN routing protocol works as follows. When a sink switches on, it starts the SUN protocol by sending a probe to notify about its existence; probe messages can then continue to be sent periodically by each sink in the network. All the sensor nodes that overhear probe messages understand to be one-hop neighbors of the sink[1] and are called *end-nodes*. These nodes will consider themselves as end-nodes as long as they will be able to hear sink probes and, in any case, for only a predefined period of time since the last probe heard. When a sensor node generates a packet to transmit, first it checks if a valid route towards a sink already exists (e.g., it may be an end-node); if this is not the case, the source node sends a *Path Establishment-Request* message that is propagated over the network by all the other sensor nodes, avoiding flooding. Eventually, such *Path Establishment-Request* message reaches one or more end-nodes that reply with *Path Establishment-Answer* messages. Traveling back to the source node, the *Path Establishment- Answer* messages collect all the necessary local information to let the source node choose the best route, i.e., relay nodes and corresponding metrics. Note that the propagation of a *Path Establishment-Request* message does not necessary terminate at end-nodes: if an intermediate relay node already knows a path towards a sink, it immediately replies to this message with a *Path Establishment-Answer*. Once the route is chosen by the source node, the data packet and the corresponding routing information is forwarded toward the selected path. Every established route remains valid for a predetermined amount of time. If the SUN ARQ mechanism is also enabled, each node of the selected route which receives a data packet will reply back to its predecessor with an *Acknowledgement* packet. Finally, when an ongoing data flow is interrupted because of a link failure (i.e., either a lost packet is detected or an end-node is not able to hear sink probes anymore), the node that registers the failure will send a *Route Error* packet to notify the source node that will start a new path discovery procedure. The explained mechanisms allow SUN to promptly react and adapt to the variable conditions of the underwater channel. In Section V, we report evaluation results of such mechanisms in different real-world experiments.

III. The S2CR WiSE Underwater Acoustic Modems

In this section we describe the modem hardware used during the field experiments of Section V, namely the S2CR White Line Science Edition (WiSE) underwater acoustic modems. The S2CR WiSE series offers an open environment for network protocol developers, providing a flexible framework for testing new network protocols and applications on real hardware. The WiSE modems include a Linux virtual machine running on the modem's hardware, providing the user with a powerful tool to develop and implement applications directly on-board the modem. The main specific features of the S2CR WiSE series are the following:

- easy access to the modem via SSH/FTP;
- communication via TCP/IP sockets between the virtual machine and the modem firmware;
- a GNU GCC-based toolchain with C/C++ compilers;
- pre-installed tcl/expect interpreters for fast prototyping or algorithm customization;

[1]This is true if the channel is symmetric. Such condition, however, is not necessarily met in practice; for this reason, according to SUN, sensor nodes discard probes received with an SNR below a given threshold. Through simulations conducted with the World Ocean Simulation System (WOSS) [7], we verified that a SNR threshold of 15 dB always ensured the existence of a return channel with good quality.

- inclusion of the ns2 and NS-Miracle network simulators, as well as of the DESERT Underwater libraries and the SUNSET framework [8], [9], both relying on the ns2/NS-Miracle engine and explicitly designed for underwater networking.

Generally speaking, instead, all the S2CR acoustic modem models manufactured by EvoLogics GmbH comprise the following components:

- *transducer with the transmit/receive amplifier*, where the physical properties of such transducer define the beam pattern of the acoustic modem and the frequency range;
- *a digital stack*, made of an analog-to-digital and a digital-to-analog converters, a physical layer protocol called S2CPhy and a data-link layer protocol named D-MAC;
- *optional ulta-short baseline (USBL) antenna*, that is a grid of 5 receivers and corresponding unique amplifiers integrated together with the transducer into one single housing;
- *optional Wake Up module*, that enables power saving by turning on the rest of the device only when an incoming acoustic message is detected, until the end of its reception, or when a data packet has to be sent.

In what follows, we briefly overview two main components of the digital stack: S2CPhy, which is implemented in a Digital Signal Processor (DSP) and a Field Programmable Gate Array (FPGA), and D-MAC, implemented into an host-processor.

A. S2CPhy: the S2C physical layer protocol

S2CPhy implements the patented S2C (Sweep Spread Carrier) signal modulation technique. S2C is based on the assumptions that, for an underwater acoustic channel, a received acoustic signal is well described by a sum of multipath components with random amplitudes and phases, and that the multipath intensity profile is discrete.

S2C method utilizes broad-spectrum signals with continuously varying frequency. After passing the underwater acoustic channel, the received signal is a sum of multipath components and, after matched filtering, it can be presented as a series of time-shifted correlation responses. For the S2C signal, these responses can be isolated, eliminating the signal distortion associated with multipath propagation. Moreover, the continuous variations of the carrier frequencies help in lowering the inter-symbol interference generated by successive multi-path components.

Differently from other common methods of digital underwater acoustic communication, an S2C signal is characterized by two levels of modulation: first, the internal modulation, for continuous variation of the carrier frequency (analog modulation); second, the external modulation, for coding the information within the signal (via discrete signal manipulation).

With the frequency band ranging from one to tens of kHz, the signal can be just hundreds of microseconds long, and the transmission speed can reach tens of kbits per second.

The key concepts of the S2C method are implemented in S2CPhy to perform the following tasks:

1) estimation of the parameters of the underwater acoustic channel (multipath intensity profile, peak component, propagation delay and Doppler shift);
2) positioning
3) packet and symbol synchronization;
4) modulation and demodulation.

B. D-MAC: the S2C data-link layer protocol

D-MAC is based on the recently developed data delivery algorithms of the S2C acoustic modems and supports two different types of data: burst data and instant messages, both detailed in the following.

Burst Data: Establishing a connection for burst data delivery requires an estimation of the channel parameters. As described in [10], the delivery algorithm optimizes the channel utilization efficiency and adapts the bitrate to the highest possible value for a particular underwater acoustic channel. All data to be transmitted is buffered and then dynamically split into smaller packets according to channel parameters. The receiver reassembles the data packet and sends it to the user in the original format.

Instant Messages: Establishing a connection is not required for delivering instant messages (IMs). A fixed bitrate (relatively low and acceptable for a wide range of acoustic channel parameters [11]) is used to deliver short IMs minimizing their delivery time. Moreover, exchange of instant messages does not interrupt ongoing burst data transmissions: IMs can be used as independent messages or as an extension of service messages of the burst data delivery protocol. An instant message can be hundreds of bits long and the S2C physical layer protocol provides reliable transmission of IMs with a 1 kbps bitrate. IMs can be classified according to the message addressing type, acknowledgment and synchronization requirements.

In the field experiments presented in Section V, nodes communicate through asynchronous instant messages whose delivery is based on an ALOHA-like scheme [12] when there is no simultaneously ongoing burst data exchange in the network. Further details on D-MAC and the classification of the difference messages supported by the S2C modems can be found in [11].

IV. MS2C_EvoLogics: a DESERT Underwater module to interface NS-Miracle with the S2C acoustic modems

In this section we describe `MS2C_EvoLogics`, a module derived from `uwmphy_modem` to implement the interface between ns2/NS-Miracle and the S2C acoustic modem presented in the previous section. `uwmphy_modem` is a stand-alone module of NS-Miracle that defines the structure in blocks, and corresponding block functions, of a general interface between the NS-Miracle network simulator and a generic real acoustic modem. In detail, `uwmphy_modem` is made of the following components:

- the main block, implemented as the `UMPhy_modem` class, that 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;
- the block in charge of managing all the communications between host and modem (in both direction), which is implemented as the abstract class `UWMdriver`;
- the block where to implement all the methods needed to map an ns2/NS-Miracle packet into a legal modem payload and do the reverse operation. The class which implements this block is called `UWMcodec`;
- the block where to implement all the methods needed to build or parse the string messages that must be sent or that come from the modem, respectively. The class which implements this block is called `UWMinterpreter`;
- finally, the block that handles the actual physical transmission and reception of messages to and from the modem, respectively. The class which implements this block is called `UWMconnector`;

`MS2C_EvoLogics` specifies `uwmphy_modem` for the S2C hardware and Fig. 1 sketches its main components: `MS2C_EvoLogics` is a class that derives `UMPhy_modem`, contains the `McodecS2C_EvoLogics` object as well as the `MdriverS2C_EvoLogics` object, and implements all the linkages among the blocks (C++ objects) depicted in the figure; the class `McodecS2C_EvoLogics` derives `UWMcodec` and makes it possible to compress into 29 Bytes (to be sent as payload of an modem message) and decompress from them (into a new NS-Miracle packet) all the information necessary to

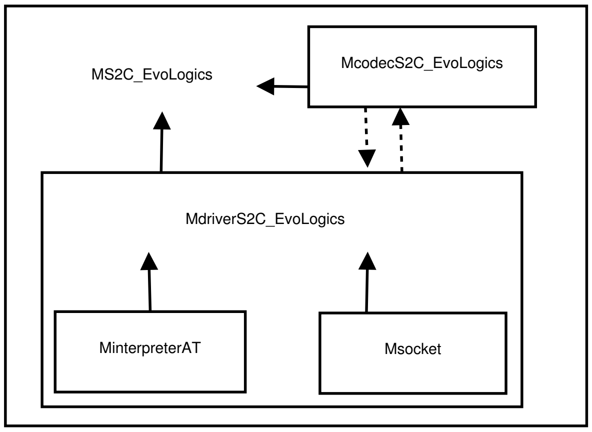

Fig. 1. Illustration of the function blocks that form the `MS2C_EvoLogics` interface.

run the SUN protocol with the NS-Miracle engine; the class `MdriverS2C_EvoLogics` derives `UWMdriver` and handles the exchange of messages between the network simulator and the S2C hardware. To this end, `MdriverS2C_EvoLogics` also contains two objects: `MinterpreterAT`, that derives `UWMinterpreter` to build and parse standard modem AT messages, and `Msocket`, that derives `UWMconnector` and implements a TCP/IP client to communicate with the S2C acoustic modem.

Exploiting the `MS2C_EvoLogics` interface[2] we have been able to conduct real-world experiments by reusing the same code prepared for simulation with the network simulator NS-Miracle. These experiments are presented and discussed in the next section.

V. Field Experiments

In this section we present four field experiments that have been conducted to test the recovery and adaptive mechanisms of the SUN protocol in a real-world environment. To this end, we deployed six S2CR WiSE Underwater Acoustic Modems in the Werbellinsee lake, near Berlin (Germany) according to the topology in Fig. 2. On board of these modems we installed the ns2/NS-Miracle engine and the DESERT Underwater framework, which contains the `MS2C_EvoLogics` interface presented in Section IV and all the necessary libraries to run, on each modem, a complete network protocol stack; this stack includes: for the application layer, a Constant Bit Rate (CBR) traffic generator that, at the transmitter, injects packets into the network according to a Poisson process with given mean; for the transport layer, a User Data

[2]Like the SUN protocol, the source code of `MS2C_EvoLogics` is part of the DESERT Underwater library [4], and can be freely downloaded at [5].

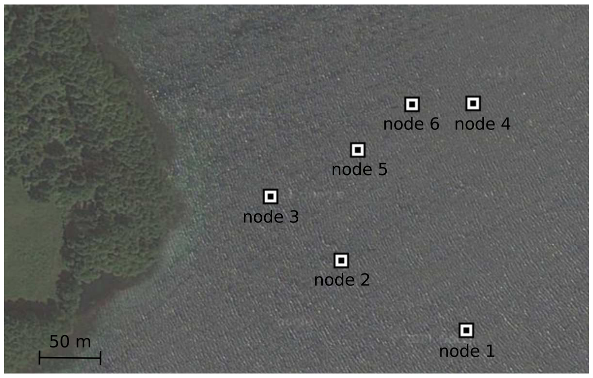

Fig. 2. Deployment of S2CR WiSE Underwater Acoustic Modems in the Werbellin lake, Landkreis Barnim, Brandenburg, Germany. (Best viewed in color.)

Protocol (UDP) and, for the network layer, the SUN protocol presented in Section II. For the data link layer, instead, the Media Access Control (MAC) actually used is D-MAC, implemented as part of the modem firmware as described in Section III.

Each deployed node could act as a transmitter, relay or sink; changing the roles of the six nodes of the underwater network in Fig. 2, we ran these four experiments:

1) *Relay failure*, which aims at testing SUN when a route must be recovered after the failure of a relay node;
2) *Sink failure*, which aims at testing SUN when a route must be recovered after the failure of a sink node, and when multiple sink nodes exist;
3) *Sink detection*, which aims at testing the behavior of SUN when an additional sink node, placed in a position that the adopted routing metric of SUN considers more convenient, is discovered;
4) *Mobile sink*, which aims at testing SUN when routes are disrupted because of the movement of a mobile sink.

In all the above experiments SUN used the lowest hop count metric to choose the best route; also its ARQ mechanism (see Section II) has been enabled to enhance point to point performance, allowing retransmission of lost packets (the maximum number has been fixed to one for all experiments, i.e., each data packet can be sent at most twice over a given channel link before being disregarded). A more detailed description of the performed experiments and the corresponding results are presented in the following.

A. Experiment 1: Relay Failure

In this experiment we consider one transmitter (node 1), one sink (node 6) and four relay nodes (nodes 2 to 5). At first, we force SUN to route packets through the path 1-3-4-6, as illustrated in Fig. 3; practically, to do so we both verified the connectivity throughout the desired path and masked other possible existing connections using ban-lists, i.e., by dropping packets coming from undesired sources[3] at each node. Successively, we cause the failure of node 3 (thus disrupting the route already established by SUN) and make a new route available, i.e., the route 1-2-5-6 as illustrated in Fig. 4. This experiment allows us to observe the behavior of SUN in case of relay failure: in Fig. 5 we indicate with circles packets that have been correctly sent from the transmitter to the receiver, both before node 3 fails (white-filled circles) and after the recovery of a new route (blue-filled circles); a vertical red-dashed line indicates when the relay failure occurs. Both before and after the relay failure, we have that packets are correctly delivered to the sink with a delay that is between 9 and 40 seconds depending on the variable channel conditions and the corresponding retransmissions required per link. Furthermore, along the x-axis, we mark with crosses the cases in which data packets are lost over the channel (i.e., the maximum number of transmissions has been reached without success in one link of the chosen routing path) and with diamonds data packets dropped by the SUN protocol because of buffering timeouts (see Section II). As a matter of fact, immediately after the node failure, we can observe a packet loss and three packet droppings: in this phase, in fact, the used path 1-3-4-6 has been broken causing the packet loss; SUN reacted to adaptively discover a new route and this action required time to exchange control messages (i.e., *Path Establishment-Request* packets) that caused three buffering timeouts and corresponding dropping of packets. Overall, to recover from the relay failure SUN needed about 77.6 s. Note that we have lost and dropped packets also in other three circumstances: 1) at the beginning of the experiment we have one dropped packet, caused by the buffering timeout due to the time needed to create a valid route for the first time ($\sim$ 22 s); 2) throughout all the experiment, channel conditions cause both packet losses and route disruptions which, in turn, lead to dropped packets[4]; 3) at the end of the experiment, where the bad channel conditions do

[3]This mechanism based on ban-lists is implemented in the `uwmphy_modem` interface and, once the physical connectivity among the nodes in the network is verified, allows us to have full control on the logical network connectivity during all the performed experiments.

[4]In detail, it may happen, for instance, that a neighbor node of the sink is unable to hear the sink probes for some time. If this time exceeds a given threshold, a *Route Error* packet will be generated and at the transmitter the established route will be considered no longer valid. Consequently, a new path discovery mechanism will be initiated by the transmitter and this may lead to buffer timeouts and corresponding dropped packets.

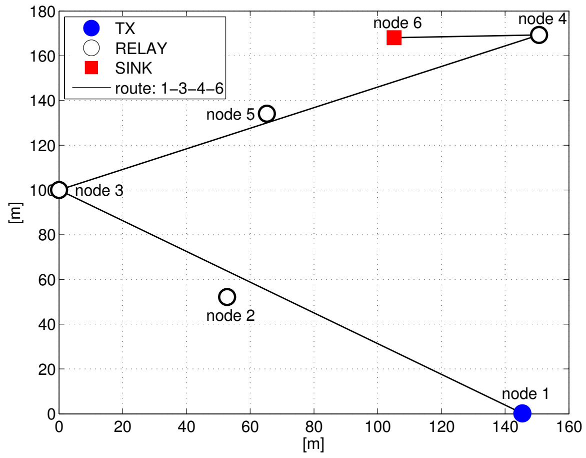

Fig. 3. Experiment 1: Relay Failure. Illustration of the initial route.

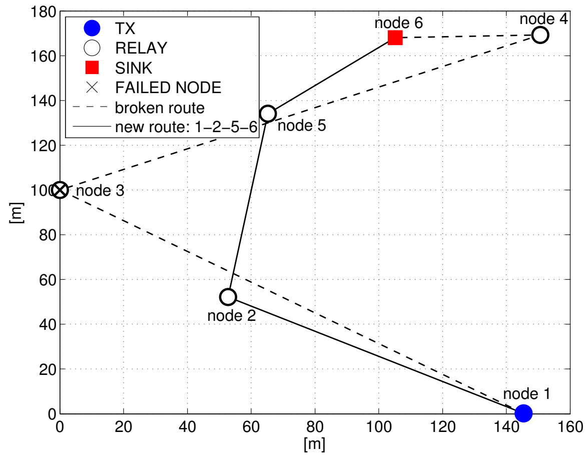

Fig. 4. Experiment 1: Relay Failure. Illustration of the node failure and the new route.

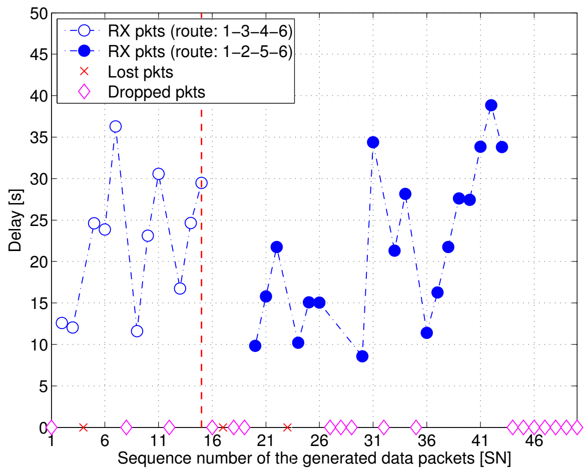

Fig. 5. Experiment 1: Relay Failure. Delay of the received packets, lost and dropped packets.

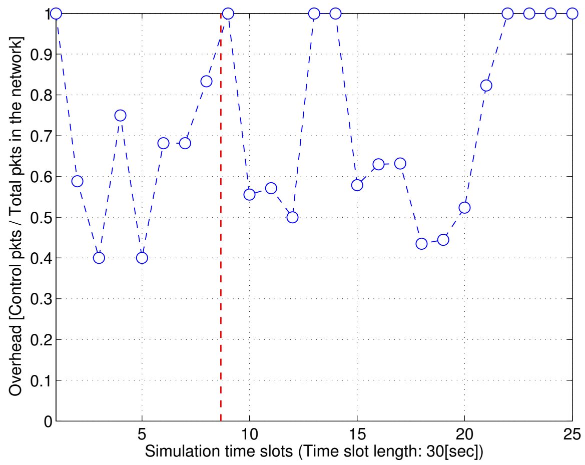

Fig. 6. Experiment 1: Relay Failure. SUN protocol overhead observed along the experiment.

not allow the transmitter to recover a valid route before completely emptying its buffer, therefore the simulation concludes with a bundle of dropped packets. The overall Packet Error Rate (PER) measured in this experiment is 0.42 (0.12 without considering dropped packets).

The protocol overhead observed during the experiment is reported in Figure 6 where we divided the time axis in slots of 30 s and for each time slot we compute the ratio between the control packets and the total packets observed in the network. In this figure, the overhead is equal to one when only control packets are sent over the network, i.e., i) at the beginning of the experiment, when the routing path must be built for the first time, ii) in correspondence of the node failure (event marked with a red-dashed vertical line) when a completely new route must be recovered and iii) at the end of the simulation, when only the sink node remains active sending probe messages. During the rest of the experiment, the number of control packets are roughly the same as the data packet (note that *Acknowledgment* packets are counted as control packets, and at each *Data* packet sent ideally corresponds one *Acknowledgment* packet): the overhead only increases when the SUN protocol has to react to bad channel conditions to effectively maintain a valid routing path.

B. Experiments 2 and 3: Sink Failure and Detection

The experiments presented in this section allow us to test the SUN protocol when: 1) the source node has to choose among multiple available routes according to the metric in use (i.e., the lowest hop count metric in our case); 2) multiple sink nodes exist and they can either

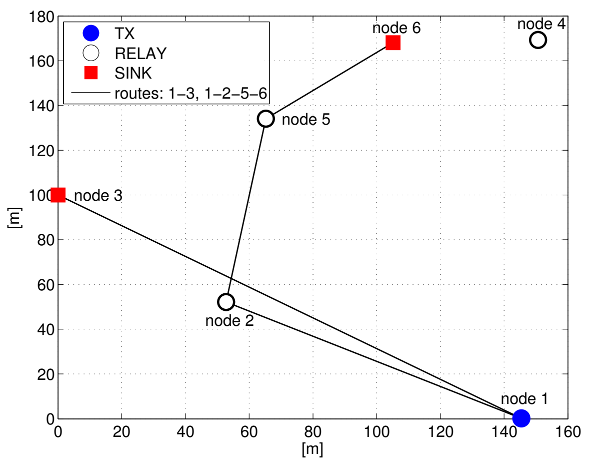

Fig. 7. Initially available routes in Experiment 2 (Sink Failure) before sink 3 fails; the routes are the same available at the end of Experiment 3 (Sink Detection), after sink 3 joins the network.

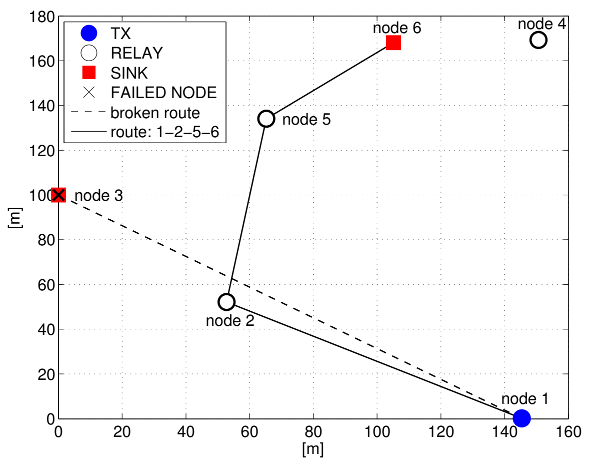

Fig. 8. Experiment 2: Sink Failure. Illustration of the node failure.

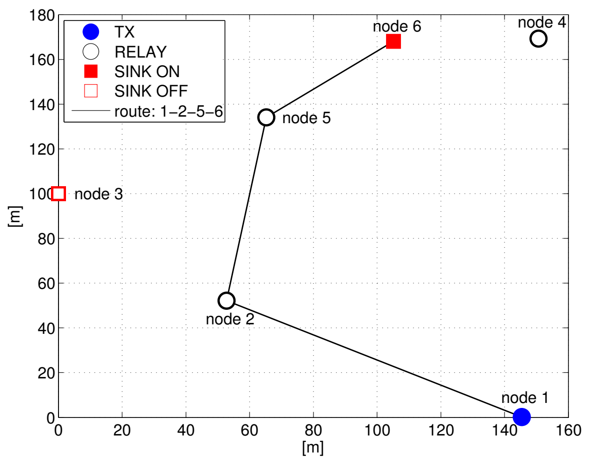

Fig. 9. Experiment 3: Sink Detection. Illustration of the initial conditions.

fail disrupting a valid routing path or appear enabling the creation of a more convenient route. For these experiments, we consider one transmitter (node 1), two sinks (nodes 3 and 6) and three relays nodes (nodes 2, 4 and 5). Fig. 7 illustrates the routes that are set up through ban-lists in Experiment 2, before one of the two sinks fails as illustrated in Fig. 8. The same routes are available in Experiment 3 when node 3 joins the network, whilst before only one sink is switched on as illustrated in Fig. 9.

As expected, in both cases, when the two sinks are simultaneously available, SUN picks the most convenient route (i.e., the one with fewest hops) forwarding data packets directly to node 3 as illustrated in Figs. 10 and 11 (white-filled circles). From these pictures we see that picking the route with fewest hops causes, in both experiments, a drastic reduction of the data packet delivery time. Moreover, in Experiment 2, the failure of the sink (event indicated by the red-dashed vertical line in Figs. 10) that is gathering the data causes the loss of a packet; however, SUN immediately recovers from such failure by exploiting the alternative available path towards node 6 (note that no dropped packets have been recorded in this case). Differently, in Experiment 3, when node 3 joins the network (the red-dashed vertical line in Figs. 11 indicates when this event happens), the reception of probe messages from this new sink and the time required to determine that the new detected path is more convenient lead to four buffer timeouts and corresponding dropped packets[5]. In any case, the PER measured in Experiment 2 and 3 is 0.26 and 0.12, respectively (0.1 and 0.02 without considering dropped packets).

[5]Future extensions of SUN will include a solution to resolve this side effect of the buffing mechanism currently implemented.

C. *Experiment 4: Mobile Sink*

With this last experiment we can observe the behaviour of SUN in a hybrid network (namely, made of both fixed and mobile nodes, see, e.g., [13]) by reproducing via ban-lists the route disruptions as if the sink actually moved. In detail, considering Fig. 12, we make the sink (node 6) move counter-clockwise around the other deployed nodes. Accordingly, the first available path from the transmitter (node 1) to the mobile sink is through the relay node 5, then through node 3; when the sink approaches the transmitter, instead, a direct path exists and when the sink goes away again the two-hop available path uses first the relay node 2 and, finally, node 4. In Fig. 13 we show the delay of the received packets, as well as the lost and dropped packets during Experiment 4. All the routing changes realized via ban-lists updates are indicated with red-dashed vertical lines. In order, SUN recovers from each route disruption after $\sim$ 116, 14, 74

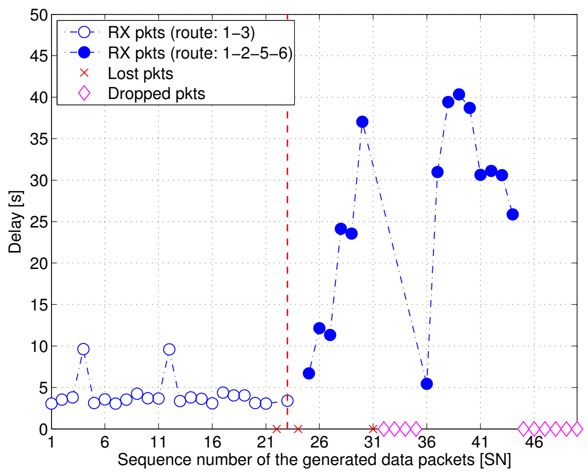

Fig. 10. Experiment 2: Sink Failure. Delay of the received packets, lost and dropped packets.

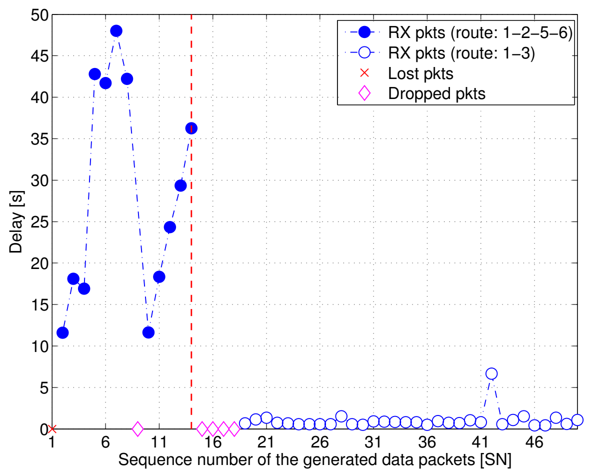

Fig. 11. Experiment 3: Sink Detection. Delay of the received packets, lost and dropped packets.

and 63 s. The PER measured in this experiment is 0.21 (0.1 without considering dropped packets). Fig. 14 and Fig. 15 illustrate, respectively, the protocol overhead and the packet traffic (classified in probe, path request, path answer, data, ack and path error packets) observed during the experiment, dividing the time axis into slots of 30 s. This experiment allowed us to asses the good adaptability of SUN also in a dynamic network.

VI. Conclusions

In this paper we exploited the integration of the NS-Miracle network simulator with the S2C acoustic underwater modems to evaluate the SUN protocol in a real-world scenario. First, we presented SUN, a dynamic, reactive Source routing protocol for Underwater Network and the S2C modem hardware. Then, we illustrate the DESERT Underwater library designed to interface the NS-Miracle simulator with real modems.

Thanks to the proposed approach, we have been able to conduct several field experiments to test SUN in a real-world environment. The collected results show that SUN promises to be a robust and reliable tool for underwater routing. Even more important, the performed real-world experiments provided us with the necessary feedback for further enhancements of the proposed protocol. This kind of feedback can hardly be obtained through simulation and observing the protocol operation in real-world scenarios provides invaluable insights.

We believe that this work represents an important achievement to spur both academia and industry towards continuing their joint activity for integrating research solutions on actual devices, with the objective of realizing prototypes and, eventually, useful and reliable products.

VII. Acknowledgments

This work has been partially supported by the European Commission under the Seventh Framework Programme (Grant Agreement 258359 – CLAM), and by the Italian Institute of Technology within the Project SEED framework (NAUTILUS project).

References

[1] "The Network Simulator - *NS-Miracle*," Last time accessed: August 2012. [Online]. Available: http://dgt.dei.unipd.it/download
[2] "The Network Simulator - *ns-2*," Last time accessed: August 2012. [Online]. Available: http://nsnam.isi.edu/nsnam/index.php/User_Information
[3] "Evologics," Last time accessed: August 2012. [Online]. Available: http://www.evologics.de/
[4] 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 *Proc. of IEEE/OES Oceans*, Yeosu, Korea, Jun. 2012.
[5] "The DESERT Underwater libraries - *DESERT*," Last time accessed: August 2012. [Online]. Available: http://nautilus.dei.unipd.it/desert-underwater
[6] D. B. Johnson, D. A. Maltz, and J. Broch, "DSR: The dynamic source routing protocol for multi-hop wireless ad hoc networks," in *In Ad Hoc Networking, edited by Charles E. Perkins, Chapter 5*. Addison-Wesley, 2001, pp. 139–172.
[7] "The World Ocean Simulation System - *WOSS*," Last time accessed: August 2012. [Online]. Available: http://telecom.dei.unipd.it/ns/woss/
[8] C. Petrioli, R. Petroccia, and J. Potter, "Performance evaluation of underwater MAC protocols: From simulation to at-sea testing," in *Proc. of OES/IEEE Oceans*, Santander, Spain, Jun. 2011.
[9] "The Sapienza University Networking framework for underwater Simulation, Emulation and real-life Testing - *SUNSET*," Last time accessed: August 2012. [Online]. Available: http://reti.dsi.uniroma1.it/UWSN_Group/index.php?page=sunset

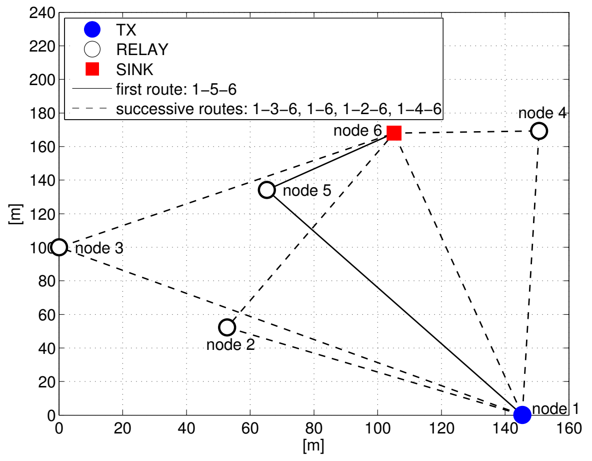

Fig. 12. Experiment 4: Mobile Sink. Illustration of the different routing paths used sequentially.

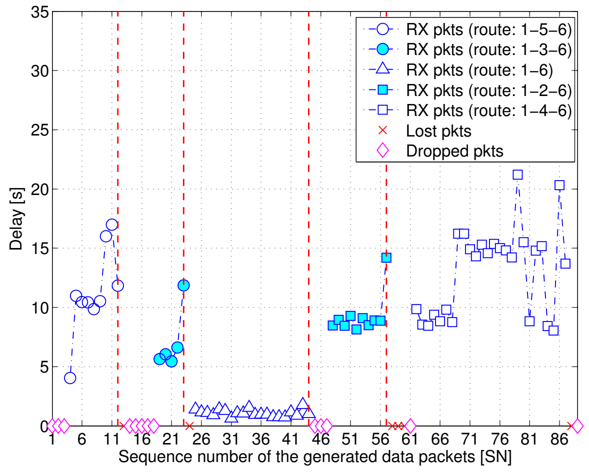

Fig. 13. Experiment 4: Mobile Sink. Delay of the received packets, lost and dropped packets.

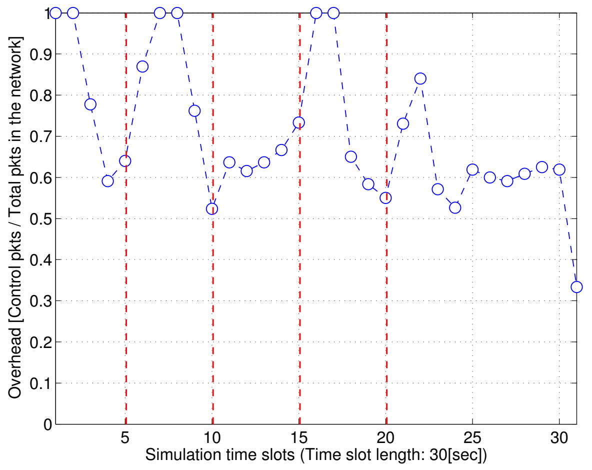

Fig. 14. Experiment 4: Mobile Sink. SUN protocol overhead observed during the experiment.

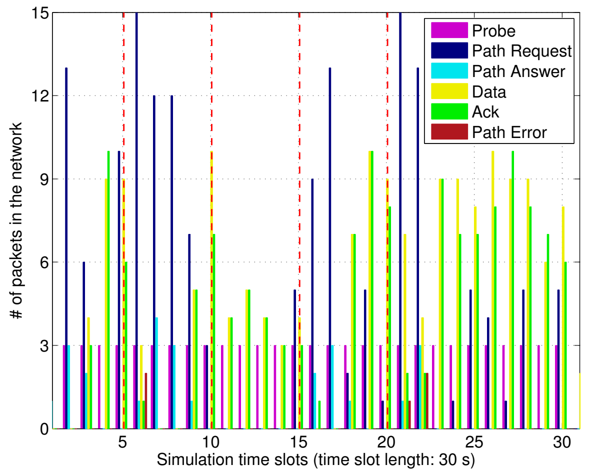

Fig. 15. Experiment 4: Mobile Sink. Network traffic generated during the experiment.

[10] O. Kebkal, “On the use of interwoven order of oncoming packets for reliable underwater acoustic data transfer,” in *Proc. of IEEE OCEANS*, Bremen, Germany, May 2009.

[11] O. Kebkal, M. Komar, K. Kebkal, and R. Bannasch, “D-MAC: Media access control architecture for underwater acoustic sensor networks,” in *OES/IEEE Oceans*, Santander, Spain, Jun. 2011.

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

[13] R. Masiero, P. Casari, and M. Zorzi, “The NAUTILUS project: Physical parameters, architectures and network scenarios,” in *MTS/IEEE Oceans*, Kona, Hawaii, US, Sep. 2011.