Abstract—To deploy successful underwater networks in the face of challenges such as low bandwidth, long propagation delay, half-duplex nature of links, high packet loss and time variability, we require highly optimized network protocols with low overhead and significant cross-layer information sharing. UnetStack is a network stack designed to provide a good balance between separation of concern, and information sharing. By replacing a traditional layered stack architecture by an agent-based architecture, we provide additional flexibility that allows novel protocols to be easily implemented, deployed and tested. In discrete-event simulation mode, UnetStack can be used on desktop/laptop computers or computing clusters to simulate underwater networks and test protocol performance. In real-time simulation mode, it can be used to interactively debug protocol implementations, and test deployment scenarios prior to an experiment. Once tested, the protocols can simply be copied to an underwater modem with UnetStack support, and deployed in the field. The stack implementation has been extensively tested, not only through carefully calibrated simulations, but also in several field experiments. We provide an overview of UnetStack and briefly discuss a few deployments to illustrate some of its key features.

I. INTRODUCTION

Commonly cited challenges in underwater networks include low bandwidth, long propagation delay, half-duplex nature of the links, high packet loss, and time-variability [1–3]. To deploy successful networks in the face of such challenges, it is important to use highly optimized protocols that are especially designed for use in such networks. Specifically, cross-layer information sharing, low-bandwidth design and accurate transmission/reception timing can be critical in these protocols. Traditional layered network stacks provide good separation of concern, but result in sub-optimal protocols. Cross-layer optimization initiatives address this shortcoming by allowing direct interaction between layers [4, 5]. In UnetStack, we take a somewhat different approach. The stack consists of a collection of software agents that provide well-defined services. This approach, often referred to as service-oriented architecture [6], provides good separation of concern while allowing information to be shared, services to be provided, and behaviors to be negotiated between different agents. The resulting network stack is flexible and allows software-defined underwater networks to be rapidly designed, simulated, tested and deployed.

The idea of software-in-the-loop underwater network stack simulation was introduced in [7], and later adopted by several underwater network simulators [5, 8]. Such network simulators allow the same code to be run in simulation and in underwater modems. Since this removes the need to port a protocol code from simulation to a field-deployable modem, considerable time and effort is saved, and subtle differences often introduced during the porting phase are avoided. UnetStack takes this approach one step further. By supporting implementation on a portable platform such as a Java virtual machine (JVM), UnetStack allows the exact same compiled binary to be used during simulation and later deployed in underwater modems.

To allow researchers to easily develop test scripts in the field, and to modify and tune their protocols without the need to recompile their code, UnetStack supports dynamic execution of Groovy scripts. Groovy [9] is an agile and dynamic language for the JVM. It builds upon the strengths of Java but has additional power features inspired by languages like Python, Ruby and Smalltalk. This provides a perfect balance between ease of learning and powerful features for development of protocols. Groovy supports the development of domain-specific languages (DSL). UnetStack, in network simulation mode, uses this support to enable an underwater network simulation DSL for researchers to describe simulation scenarios in English-like human-readable form.

UnetStack has been actively developed and extensively used for simulation and numerous field experiments over the past 3–4 years. A community version of UnetStack is available online. Bindings for UnetStack for several popular modems including the ARL UNET-II modem [9], Subnero modem [10] and Evologics WISE-edition modem [11] are currently available.

Several other underwater network stacks and simulators have also been developed over the past few years (e.g. [5, 10–12]). Many of these initiatives are based on the ns2 network simulator [13] that is very popular for simulation of
terrestrial networks. The key advantage of this approach is that many traditional communication networks researchers are already familiar with the simulator. While there is extensive documentation available for a new researcher to learn ns2, the learning curve is quite steep. Moreover, most of the documentation is written with terrestrial networks in mind. As ns2 is not originally designed with cross-layer collaboration in mind, an extension known as miracle is adopted in DESERT and SUNSET to facilitate cross-layer interaction. Since ns2 is primarily a discrete-event simulator, both initiatives had to make significant changes to it to run in real-time mode for field deployments. Even then, moving a simulated protocol to field operations requires additional non-trivial steps such as cross-compilation.

UnetStack approaches the problem differently. Rather than using an existing discrete-event network simulator as a foundation, we adopt the open-source fjåge lightweight agent framework designed to support discrete-event simulations as well as real-time operation. To avoid being constrained by traditional layered network stack architecture, we embrace a service-oriented agent architecture that has cross-layer (cross-agent) interaction at its core. This enables us to add functionality that is typically not provided by a traditional network stack (e.g., acoustic ranging, software-defined modem) seamlessly. By choosing JVM technology as a platform, we ensure portability across operating systems and hardware, for simulation and for embedded operation during field deployments. Through the use of Groovy and a custom-designed DSL, we shorten the learning curve for a new researcher. While we provide default agents implementing a full set of underwater networking protocols, we also allow the researcher to reorganize the stack as needed, and introduce new protocols and functionality as desired. While we acknowledge that an existing researcher familiar with ns2 might have to spend a few days to learn UnetStack, we believe that the effort would be more than compensated for by a shorter development and testing cycle, and much greater architectural flexibility of this approach.

The rest of this paper is organized as follows. In section II, we provide an overview of the UnetStack architecture. In section III, we show how UnetStack can be used for network simulation. We briefly discuss two underwater networking experiments using UnetStack in section IV. Finally, we present some concluding remarks in section V.

II. UnetStack Overview

The UnetStack architecture (depicted in Fig. 1) defines a set of software agents that work together to provide a complete underwater networking solution. Agents play the role that layers play in traditional network stacks. However, as the agents are not organized in any enforced hierarchy, they are free to interact in any way suitable to meet application needs. This promotes low-overhead protocols and cross-layer information sharing. Multiple agents providing similar services may coexist in the modem (e.g., drivers for multiple modems, acoustic and radio links). Each software agent provides some local services and/or implements network protocols that require interaction with agents on other network nodes. The architecture defines the interfaces for commonly needed agents in terms of the services and capabilities that the agent must or may provide. The specifications are extensible, allowing agents to provide additional services that may be used by other agents implementing cross-layer optimized protocols. Although the specifications focus on underwater networks, they allow wired and wireless radio links to be included as part of the network. A basic set of agents to enable a fully functional underwater network are included in the downloadable community version of the UnetStack. Designed for extensibility, UnetStack allows additional agents for optimized protocols to be rapidly developed, tested and deployed.

A. The Basics

A UnetStack agent is a self-contained software component that provides a well-defined functionality. Agents play a similar role as layers in traditional network stacks, but are more flexible in their interactions with other agents. Agents interact with each other through messages. Typical messages include requests, responses and notifications. Responses are always associated with a request, while notifications may be unsolicited. Agents also support parameters that can be used
to configure or monitor the agent. The parameters can be set or queried through appropriate messages. Some agents support multiple indexed parameter sets (e.g., parameters for various logical communication channels). Messages can not only be sent to specific agents, but also can be broadcasted on a topic. All agents subscribing to a topic, receive a message broadcasted on that topic. Unsolicited notifications usually are sent on topics associated with an agent, since an agent does not know a priori which other agent might be interested in that notification. A collection of requests, responses, notifications and parameters that together form a cohesive functionality is known as a service. If an agent provides a service, it advertises the service by registering it with the ‘directory’. An agent requiring a specific service can look up providers in the directory, without having to know a priori the details of the agent that provides the service. Services may define capabilities that represent optional functionality that a service provide may choose to implement. Agents advertise such capabilities for other agents to query.

The Fjøge framework defines a shell service that allows a user to interact with the stack via text commands. It also provides a console shell, a TCP/IP shell and a graphical shell that provide local and remote access to the stack. In addition to the shell service, UnetStack defines a number of services that make up a typical underwater network stack. We provide an overview of the important services next. Detailed specifications are available online.

### B. Datagram Service

Many agents provide the datagram service. This service defines messages and capabilities for transfer of packets of data over the network. A DatagramReq message asks the agent to transmit some data. The agent responds with an AGREE, REFUSE or FAILURE message. When the datagram is received at the peer node, the agent on that node sends out a DatagramNtf on its broadcast topic. Other agents interested in receiving such messages can subscribe to the topic. The maximum size of the datagram supported is defined by the MTU parameter. This parameter may be accessed using the ParameterReq and the ParameterRsp messages.

The datagram service also defines a number of optional capabilities. These can be queried using the CapabilityReq message. On receiving this request, the agent responds with a CONFIRM, DISCONFIRM or CapabilityListRsp message. The optional capabilities include FRAGMENTATION, RELIABILITY, PROGRESS and CANCELLATION.

If the FRAGMENTATION capability is advertised, the agent may choose to fragment/reassemble the datagram in order to support a large MTU. If the RELIABILITY capability is supported and reliability is requested by setting the reliability attribute of the DatagramReq, the agent sends out DatagramDeliveryNtf or DatagramFailureNtf to confirm delivery or failure of datagram. If the PROGRESS capability is advertised, the agent sends out DatagramProgressNtf messages at regular intervals for long datagram transmissions. If the CANCELLATION capability is supported, a datagram queued for transmission can be cancelled using the DatagramCancelReq. If a request is made for a capability that is not supported, the agent replies with a NOT_UNDERSTOOD or REFUSE message.

The important messages, capabilities and parameters in the service are summarized in Figure 2. Additional messages for parameter access, capability check, etc are commonly supported by most agents, and are omitted from the summaries in this paper for brevity.

### C. Physical service

The physical service is typically provided by physical layer agents such as modem drivers and simulated modems. An agent advertising this service must also provide the datagram service.

The main messages in this service are the TxFrameReq and RxFrameNtf – they extend the DatagramReq and DatagramNtf messages to offer additional physical layer options. Additional optional capabilities such as TIMED_TX and TIMESTAMPED_TX allow physical layers to offer accurate control over transmission time. This may be used by other

---

<table>
<thead>
<tr>
<th>Capability</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FRAGMENTATION</td>
<td>Capable of fragmentation/reassembly of datagrams</td>
</tr>
<tr>
<td>RELIABILITY</td>
<td>Capable of link-level reliability</td>
</tr>
<tr>
<td>PROGRESS</td>
<td>Capable of reporting progress via DatagramProgressNtf messages</td>
</tr>
<tr>
<td>CANCELLATION</td>
<td>Capable of cancelling queued datagram transmission</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Request</th>
<th>Possible Responses</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Datagram-Req</td>
<td>AGREE, REFUSE,</td>
<td>Transmit a datagram</td>
</tr>
<tr>
<td></td>
<td>FAILURE</td>
<td></td>
</tr>
<tr>
<td>Datagram-CancelReq</td>
<td>AGREE, REFUSE,</td>
<td>Cancel a datagram</td>
</tr>
<tr>
<td></td>
<td>NOT_UNDERSTOOD</td>
<td>transmission</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Notification</th>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Datagram-Ntf</td>
<td>default</td>
<td>Notification of a received datagram</td>
</tr>
<tr>
<td>Datagram-DeliveryNtf</td>
<td>requester</td>
<td>Notification of successful delivery of reliable datagram</td>
</tr>
<tr>
<td>Datagram-FailureNtf</td>
<td>requester</td>
<td>Notification of unsuccessful delivery of reliable datagram</td>
</tr>
<tr>
<td>Datagram-ProgressNtf</td>
<td>requester / default</td>
<td>Periodic notifications of datagram transfer progress</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>r/w</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTU</td>
<td>rw</td>
<td>Maximum datagram size in bytes</td>
</tr>
</tbody>
</table>

---

6fjøge also supports a distributed deployment where various agents in the network stack can potentially run on different computing platforms connected over a local network.

7Other utility services such as state persistence are defined in UnetStack but not critical to the operation of an underwater network. For conciseness, we do not cover these services in this paper.


---

Fig. 2. A summary of the datagram service.
Capability Description
TIME-STAMPED_TX Transmissions with timestamp encapsulated in frame
TIMED_TX Transmissions of frames at specified time

Request Possible Responses Description
TxFrame-Req AGREE, REFUSE, FAILURE Transmit a physical layer frame
ClearReq AGREE, FAILURE Abort all transmissions/receptions

Notification Topic Description
RxFrame-Ntf default Frame addressed to node arrived
RxFrame-Ntf SNOOP Frame addressed to another node overheard
BadFrame-Ntf default Received frame could not be successfully decoded
Collision-Ntf default Frame detected during reception of another frame

Parameter r/w Description
rxEnable rw True if reception is enabled, false otherwise
propagation-Speed rw Signal propagation speed in m/s
refPower-Level ro Reference power level in dB re µPa @ 1m
timestamped-Tx xDelay ro Delay in seconds to transmit timestamped frames
time ro Current physical layer clock time in µs
busy ro True if modem is busy transmitting/receiving, false if modem is idle

Indexed parameters – index: CONTROL (0), DATA (1)
Parameter r/w Description
MTU ro Maximum frame size in bytes
frame-Duration ro Frame duration in seconds
powerLevel rw Transmission power level in dB re reference level
maxPower-Level ro Maximum allowable transmission power in dB re reference level
minPower-Level ro Minimum allowable transmission power in dB re reference level
error-Detection ro Number of bytes used for error detection (CRC/Checksum)
frame-Length ro Frame length in bytes
maxFrame-Length ro Maximum allowable frame length in bytes
fec rw Forward error correction (FEC) code
decList ro List of supported FEC codes
dataRate ro Effective data rate in bits/second

Fig. 3. A summary of the physical service.

other nodes.
The physical service defines two logical communication channels – CONTROL and DATA. The CONTROL channel is typically a low-rate but robust communication link that is used for control information and link negotiation. The DATA channel may be an adaptively tuned high-rate communication link for large data transfer. Drivers for modems that do not support such differentiation may simply treat both channels identically.

The key messages, capabilities and parameters of the physical service are summarized in Fig. 3.

D. Ranging Service

Agents offering the ranging service provide time synchronization and ranging functionalities between pairs of nodes. Such agents usually require a physical service provider that supports the TIMESTAMPED_TX capability.

The ranging service provides support for two-way travel-time (TWTT) as well as one-way travel-time (OWTT) range estimation. For OWTT to be used, synchronization information has to be first obtained between nodes. If this is not available a priori, it may be obtained through a TWTT exchange. The lifetime or validity of the synchronization information depends on the accuracy/drift of the clocks used in the modems.

TWTT ranging is initiated via the RangeReq message, and eventually leads to a RangeNtf notification on the initiating node. OWTT ranging is initiated via the BeaconReq message, and leads to a RangeNtf notification on all other nodes that synchronization with the initiating node.

The key messages, capabilities and parameters of the ranging service are summarized in Fig. 4.

Fig. 4. A summary of the ranging service.
E. Link Service

Agents offering the link service provide single-hop communication. Single-hop here refers to a logical single hop in the UnetStack network. For example, a link may be provided over wireless radio network that has multiple physical hops (e.g., using UDP/IP). However, as long as the link does not pass through multiple UnetStack nodes, it is considered a single-hop link.

All agents supporting this service must provide the datagram service. Agents offering a reliable link advertise it using the RELIABILITY capability.

F. Medium Access Control Service

Agents offering the medium access control (MAC) service provide some implementation of a MAC protocol. The basic MAC functionality is accessed by making a ReservationReq request and waiting for the corresponding ReservationStatusNtf message before using the channel. Before a request is granted, if the client agent determines that the channel is no longer required, it may send a ReservationCancelReq message.

Some MAC protocols involve control frame exchanges between nodes. Such frames may carry additional data such as acknowledgments (ACK). This is supported through the optional RELIABILITY capability, ackPayloadSize parameter and TxAckReq and TxAckNtf messages. In some cases, the control frames can carry additional payload data from other agents. This is advertised through the reservationPayloadSize parameter and accessed using the payload data in the ReservationReq and the ReservationAcceptReq messages.

The key messages, capabilities and parameters of the MAC service are summarized in Fig. 5.

G. Routing and Route Maintenance Services

Agents offering the routing service provide multi-hop communication for datagram messages. These agents accept datagram messages and route them to their destination based on supported underlying routing algorithms. Such algorithms are often based on routing tables, which may be maintained by providers of the route maintenance service. All agents supporting the routing service must support the datagram service.

Agents offering the route maintenance service generate route discovery/change notifications to allow routing agents to maintain routing tables. They also provide the ability to initiate discovery or trace of a network route. The key messages of the route maintenance service are summarized in Fig. 6.

H. Transport Service

Agents offering the transport service provide end-to-end reliability and fragmentation/reassembly for large datagrams. They may also support connection-oriented services for data streaming. Agents providing this service typically use the routing service for multi-hop delivery of data. All agents supporting this service must support the datagram service, along with the RELIABILITY and FRAGMENTATION capabilities. It is also recommended that they support the CANCELLATION and PROGRESS capabilities, since datagrams at this level are likely to be large.

I. Remote Access Service

Agents offering the remote access service provide control over remote nodes. This includes querying/setting parameters, delivering text messages, transferring files and running scripts remotely. At present, no authentication or security is offered, but we expect to extend this service to provide both in the future. The primary messages defined by the remote access service are summarized in Fig. 7.

J. Node Information Service

An agent offering the node information service manages and maintains a node’s attributes such as address, location, speed etc, in systems where such information is available. The agent often integrate with the host system (e.g., using ROS).
The baseband functionality is accessed through TxBasebandSignalReq and RecordBasebandSignalReq requests, and RxBasebandSignalNtf notification. Additionally an optional time-triggered transmission and recording ability may be advertised using the TIMED_BBTX and TIMED_BBREC capabilities. The key messages, capabilities and parameters in the service are summarized in Figure 10.

M. The Default Stack

The community version of the UnetStack available for download has one or more default implementations for each of the services. The stack therefore can be used for simulation and deployment (with additional modem drivers) of a fully functional underwater network. Since the stack is extensible, researchers can easily replace the default agents or add new agents and services. If some of the agents are not required, they can be disabled to yield a leaner stack for highly resource-constrained embedded devices.

We briefly describe the agents in the default stack. The NodeInfo agent provides the node information service by

### Baseband Service

The baseband service is designed to enable researchers to access low-level signal transmission and reception capability of a modem. This not only allows development of software-defined modems, but also enables numerous other applications. Agents offering the baseband service are most commonly modem drivers and modem simulators.

#### L. Baseband Service

The baseband functionality is accessed through TxBasebandSignalReq and RecordBasebandSignalReq requests, and RxBasebandSignalNtf notification. Additionally an optional time-triggered transmission and recording ability may be advertised using the TIMED_BBTX and TIMED_BBREC capabilities. The key messages, capabilities and parameters in the service are summarized in Figure 10.

### M. The Default Stack

The community version of the UnetStack available for download has one or more default implementations for each of the services. The stack therefore can be used for simulation and deployment (with additional modem drivers) of a fully functional underwater network. Since the stack is extensible, researchers can easily replace the default agents or add new agents and services. If some of the agents are not required, they can be disabled to yield a leaner stack for highly resource-constrained embedded devices.

We briefly describe the agents in the default stack. The NodeInfo agent provides the node information service by

### MOOS [16], DSAAV [17], etc) to obtain this information. The information may be used by agents implementing highly optimized network protocols. The set of parameters supported by a node information service agent is shown in Fig. 8.

#### K. Address Resolution Service

In some small networks, all network nodes have a priori known addresses. However, in other networks, addresses may be assigned dynamically and discovered using node names. The address resolution service defines the messages required for address allocation and name-to-address resolution. These messages are shown in Fig. 9.

### L. Baseband Service

The baseband service is designed to enable researchers to access low-level signal transmission and reception capability of a modem. This not only allows development of software-defined modems, but also enables numerous other applications [14]. Agents offering the baseband service are most commonly modem drivers and modem simulators.
serving as a central repository where the relevant information can be deposited. The AddressResolution agent implements the address resolution service using a simple hashing mechanism to map names to addresses. Since dynamic conflict resolution is not provided, this serves well for a small network but needs to be replaced by a more sophisticated protocol in larger networks. The Ranging agent provides OWTT and TWTT ranging as well as time-synchronization as defined by the ranging service. The ReliableLink agent offers a link service with fragmentation/reassembly and link-level reliability. The UdpLink agent uses UDP/IP to provide a link service over wired or radio links. The AlohaACS agent is the default MAC service provider. It implements a carrier-sensing flavor of Aloha with adaptive backoff based on network load. An alternate Maca agent can be used as the MAC service provider if desired. This agent implements the popular MACA protocol with reliability, early-ACK and multi-ACK options [18]. The Router agent implements the routing service based on routing tables. These tables may be statically populated, or dynamically updated on demand using the route maintenance service provided by the RouteDiscoveryProtocol agent. The SWTransport agent offers a transport service using stop-and-wait ARQ-based end-to-end reliability. The RemoteControl agent offers the remote access service to allow nodes to be reconfigured and updated remotely.

The physical service is provided by all modem drivers and a simulated generic modem (HalfDuplexModem). The ARL UNET-II modem driver, Subnero modem driver and the HalfDuplexModem also provide the baseband service.

In addition to these services, the state persistence service is offered by the StateManager agent. User interactivity is provided through the shell service implemented by the local ConsoleShell, remote TcpShell and the graphical SwingShell agents. Fig. 11 summarizes the available agents in the default stack.

We expect the protocol offerings in the UnetStack to grow over time, as more researchers implement, test and contribute new protocols and agents. We encourage community partici-

Fig. 11. A summary of the agents in the default stack.

Fig. 12. A sample script illustrating the use of the domain-specific language (DSL) used by the simulator.

pation and contribution (in source or binary form) via the UnetStack support forum [9]. Contributed protocol implementations can provide a way for comparative benchmarking of protocols in identical simulation models/scenarios as described in the next section.

III. NETWORK SIMULATION

The UnetStack network simulator (aka “UnetSim”) simulates an underwater network on a single computer (or a cluster of computers) in realtime, or as a discrete-event simulation. UnetSim is easy to install, learn and use, and once an agent is developed and tested using UnetSim, it can simply be copied to any UnetStack-compliant modem for field testing.

The scenario to be simulated is described in a Groovy DSL. A sample simulation script is shown in Fig. 12. While the English-like DSL provides good readability, the simulation script retains the capability to express complex logic in Groovy. The script describes the location and motion of each network node and sets up the network stack at each node. It also sets up behaviors to generate network traffic for automated simulation, or enables an interactive shell for user-driven simulation. If needed, the script may also collect network performance statistics and display them.

Fig. 13 shows the architecture of the simulator. Multiple UnetStacks, one for each node being simulated, are simultaneously instantiated. They interact with each other through a simulated physical layer. The behavior of this simulated physical layer is controlled by a modem model and a channel model.

A. Modem Models

The default modem model is that of a generic underwater half-duplex modem with support for CONTROL and DATA channels, TIMED_TX and TIMESTAMPED_TX capabilities, and the baseband service. Parameters such as data rate,
frame length, carrier frequency, transmit power level, detection preamble duration, etc for the modem can be customized.

Specific modem models for the ARL UNET-II and the Subnero modems have also been developed. These provide a more accurate simulation of the specific modem’s behavior in terms of timing and functionality.

B. Channel Models

Several channel models are available to meet the needs of various kinds of simulation studies. To allow researchers to address their specific needs, the channel model implementations provide extension hooks. In cases where the needs differ significantly from the available models, researchers can provide a custom implementation of the channel model. We describe the currently available models below:

1) Lossy protocol channel model: This is the simplest of the channel models. In this model, every modem has fixed detection range $R_d$, communication range $R_c$ and interference range $R_i$. The power level setting in the modem is ignored. A transmission can be successfully detected with a fixed probability $p_d$ at any range $R \leq R_d$. A detected transmission can be successfully received with a fixed probability $p_c$ at any range $R \leq R_c$. A transmission results in interference (and potentially a collision) at unintended nodes up to range $R \leq R_i$. Although this model is simple, it is often used as a first-order approximation for wireless networks.

2) Basic acoustic channel model: This is a physics-based channel model that provides a good balance between complexity, speed and accuracy. The model is parametrized by the carrier frequency $f$, bandwidth, spreading loss factor $\alpha \in [1, 2]$, temperature, salinity, water depth, noise spectral density, acceptable probability of false detection $p_{fa}$, Rician or Rayleigh fading parameters and irreducible packet loss $p_{min}$. Taking a similar approach as [19], the signal-to-interference-and-noise ratio (SINR) is computed using a transmission loss of $10\alpha \log_{10} R + a(f) R$ dB at a range $R$, where $a(f)$ is an absorption factor from [20] p.10. The probability of detection $p_d$ is then modeled assuming a matched filter for preamble detection operating at the specified $p_{fa}$. Bit errors are simulated assuming a Rician or Rayleigh fading channel. In addition, packet errors are also simulated with a probability $p_{min}$ to model unforeseen short-term events that cause packet loss.

A more comprehensive time-varying physics-based channel model based on statistical characterization of underwater acoustic channels [21] is currently under development.

3) MISSION 2012a channel model: This is an empirical channel model based on the MISSION 2012 experiment described in section IV. The probability of detection $p_d(i,j)$ and probability of successful reception $p_c(i,j)$ were estimated from a large number of transmissions on each link (from node $i$ to node $j$) in the MISSION 2012 network [22]. The channel model uses these probabilities to model packet reception on each link as a Bernoulli random process. Since the probabilities were measured in a specific 5-node network, the model cannot be applied to arbitrary network geometries. However, it is extremely useful for testing of network protocols, and for comparative benchmarking of network protocols in realistic channel conditions.

4) MISSION 2013a channel model: This is another empirical channel model, based on the MISSION 2013 experiment described in section IV. This model is similar to the MISSION 2012a model, but for a 7-node network and a different geometry.

The MISSION 2012a and 2013a models assume that packet failures on a link are Bernoulli random processes, and independent of failures on other links. They also assume that the link performance does not vary significantly over a short time. In [23], we show that these assumptions are not always accu-
rate. An empirical channel model (MISSION 2013b) relaxing these assumptions is currently also under development.

IV. EXPERIMENTS

UnetStack has been tested in several experimental deployments over the past five years. In this section, we briefly discuss two of the experiments with nodes deployed over several days.

A. The MISSION 2012 Experiment

The MISSION 2012 experiment was held in October 2012 in Singapore waters. During the experiment, a UNET network and a Seaweb network [24] were deployed simultaneously. UnetStack was only deployed on the 5 UNET nodes (Fig. 14), and so we focus our discussion only on these nodes. Node P21 was a surface modem deployed from a barge, while the other 4 nodes were bottom-mounted UNET-PANDA nodes (Fig. 15). The surface modem could be directly accessed from a laptop, and was used to control the network. The bottom-mounted nodes were only accessible acoustically. The experiment tested the physical, baseband, ranging, link, MAC, transport and remote access functionality of UnetStack.

One of the main objectives of this experiment was to measure statistical variability of the communication channel. Over 41,000 transmissions of data frames and channel probe signals were made by the 5 nodes during the experiment. All nodes logged baseband received signals for each reception, enabling post-experiment analysis of channel variability. Some results from this analysis can be found in [22].

B. The MISSION 2013 Experiment

The MISSION 2013 experiment was held in November 2013 in Singapore waters. The experiment was larger than the MISSION 2012 experiment, with more UNET and Seaweb nodes in the water. The experiment included two autonomous underwater vehicles (AUVs) as mobile UNET nodes (Fig. 16), and a gateway node running UnetStack to allow data to flow between the UNET and Seaweb networks. Seven static UNET nodes were deployed as shown in Fig. 17. Node 21 was a surface modem deployed from a barge, while all other nodes were bottom-mounted UNET-PANDA nodes and only accessible acoustically. While channel variability measurements were also made during this experiment [23], the experiment was primarily aimed at testing various application scenarios that required multi-hop underwater networks. Specific tests were designed to test each agent in the UnetStack during this experiment. The routing and route management services in UnetStack were used to dynamically communicate with the AUVs as they moved across the network. Time synchronization and OWTT ranging was used to localize and track the AUVs in realtime.

V. CONCLUDING REMARKS

With years of development and testing, and valuable feedback from numerous researchers and users, UnetStack has evolved to become a robust and flexible network stack for underwater networks. Not only is it well suited for field deployment, but also provides an excellent platform for network simulation studies. Once protocol implementations are tested in the UnetStack simulator, they can be deployed to UnetStack-compatible modems for field deployment without the need for porting or recompilation. In addition, several utility classes are built into UnetStack to enable researchers to rapidly translate their protocol ideas into working implementations.

We urge researchers to contribute reference implementations of their protocols and channel models on UnetStack in source or binary form. This will allow other researchers to benchmark their protocol performance against reference implementations of published protocols, and in various simulated underwater channels.
MISSION 2013 experiment.

Fig. 17. Seven static network nodes deployed in Singapore waters during the MISSION 2013 experiment.

Fig. 16. The STARFISH AUV being deployed as a mobile network node during the MISSION 2013 experiment.

ACKNOWLEDGEMENT

We wish to thank several researchers from ARL, and our research collaborators from Subnero, Evologics, and Northeastern University for valuable feedback that has helped improve the network stack. We also wish to thank our STARFISH AUV team, engineers and lab/field support staff at ARL who have supported numerous experiments that tested the stack in the field.

REFERENCES


