Unified Simulation and Implementation Software Framework for Underwater MAC Protocol Development

Shiraz Shahabudeen[1], Mandar Chitre[1], Mehul Motani[2], Alan Low Yong Siah[3]

[1] ARL, National University of Singapore
[2] ECE, National University of Singapore
[3] Singapore Technologies Electronics

shiraz@nus.edu.sg, mandar@arl.nus.edu.sg, motani@nus.edu.sg, alanlow@stee.stengg.com

***Abstract*—MAC protocol development for underwater acoustic modem has been an active research area for many years. The primary mode of investigation is through network simulations. Subsequent implementation in modems for testing in sea trials typically involves porting of simulation code into the appropriate programming language and software environment in a modem. Correct porting is critical since, if the modem implementation differs from the simulation code, comparison of sea trial results with simulation results become misleading and the performance observed in simulations might be compromised. One of the challenges associated with the above porting process is to maintain exact algorithmic match in the modem implementation and the simulation code as in many protocols, minor variations in the logic can have significant effects on the protocol behavior. Apart from programming errors in porting, this task can be made difficult if the modem hardware have additional complexities which make one-to-one translation difficult.**

We have developed a software framework to address this. In this structured framework, identical C code for the MAC protocol runs in both the simulator and the modem. The simulator captures the essential behavior of the modem and uses the same software interfaces as the modem. Since the same code is used with no change, performance comparisons between simulations and sea trials become meaningful. We present a case study using the ARL OFDM modem operating in the 31.25 kHz centre frequency, where one MAC protocol is simulated and also operated with no code changes in the modem. The authors aim to make this framework publically available, whereby other researchers can develop MAC protocols that will run in an underwater modem with no modifications. This framework makes a modem trial an easy step after simulation study. The aim of this paper is to provide an overview of the key aspects of this framework.

***Index Terms*—Underwater, medium access control, MAC, standardization, simulation, implementation**

I. INTRODUCTION

There are many commercial and university based acoustic modems in use around the world, and they use many forms of MAC protocols - both scheduled (such as TDMA) and random access. Improved underwater MAC protocols for varying deployment scenarios are constantly being investigated in many research institutes and commercial entities. The primary mode of investigation is through simulations using network simulators such as NS2 etc, with proprietary modifications to model the underwater channel. Simulations are used since the design/debug/validate cycle is less costly. Some of the promising protocols can then be implemented in modems for sea trials. This typically involves porting of simulation code into the appropriate programming language and software environment in a modem. There are many challenges associated with such a porting process. One of them is to maintain exact algorithmic match in the modem implementation and the simulation code since minor variations in the protocol logic can have significant effects on the protocol behavior. If coded separately based on simulation code from another environment, there could be significant costs in terms of time and effort to develop it.

Modem hardware could have additional complexities absent in the simulator and the original simulation code might need significant modifications to cater to such differences. This in turn can alter protocol behavior and performance as seen in simulations and comparison of sea trial results with simulation results could become misleading. To give an example, the ARL acoustic OFDM modem has a High Power Amplifier (HPA) that needs to be turned on before transmissions and turned off for receptions. By design, the control of this feature is delegated to the Datalink Layer (DLL) together with MAC functionality, for optimum use. A standard simulation program for a MAC protocol might not take into account such aspects as the HPA control, and during the porting process this could lead to difficulties in maintaining algorithmic integrity.

We have developed a software framework to address this situation. It is built upon an earlier proposal for a unified framework called Underwater Network Architecture (UNA) [1] for a structured inter layer communications. In this framework, identical C code for the MAC protocol runs in both the simulator and the modem. The simulator captures the essential behavior of the modem and uses the same software interfaces as the modem. The MAC code thus developed will

run on any UNA compliant modem.

Being able to use the same MAC code in simulator and in the modem is of immense value. Algorithm debugging is possible within the simulation environment itself which can then be run in the modem with virtually no further issues. Since the same code is used without changes, performance comparisons between simulations and actual sea trials become meaningful. The simulator also becomes a stronger and trustable tool, once there is such an identical code linkage with the modem.

There have been some papers on underwater simulators such as [2]. A very closely related work [3] looks at unifying interfaces to Physical Layer etc to facilitate portable MAC development, and have some overlaps with the ideas in this paper. But the concept of identical code for simulation and implementation is not discussed.

The main contribution of this paper is the presentation of this unified software framework for MAC protocol development for underwater networks, and a case study where one MAC protocol is simulated and also operated in the modem. The authors will make this framework publically available, whereby other researchers can develop realistic MAC protocols that will run in any UNA compliant modem without requiring modification. This could help make MAC protocol results from different groups around the world easily comparable on equal terms. Typically, simulations are performed to test out MAC protocol ideas and get some indicative results. Very rarely are these translated into real acoustic modem trials and compared with simulation results. This framework makes a modem trial an easy step after simulation study. The need for such a globally accepted framework with standardized interfaces for MAC simulation study was recently emphasized in [4] and we hope that this paper is a step toward such a standard.

II. FAPI AND UNA

This software framework is based on the proposal for a unified framework called Underwater Network Architecture (UNA) published earlier [1]. In that proposal, a structured inter-layer message passing mechanism was defined as shown in Fig. 1. It uses a request (REQ), response (RSP), and notification (NTF) paradigm for inter-layer communications. The higher layer sends a REQ to which it expects a RSP. NTFs are unsolicited messages from the lower layer. This simple model suffices for all the interactions between layers in the acoustic network stack.

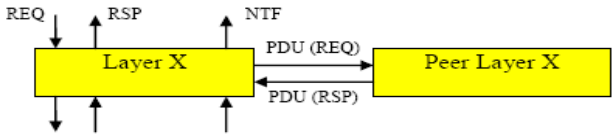

Fig. 1. Message nomenclature in UNA (adapted from [1])

The Datalink Layer is defined to provide single-hop data transmission within which the main sub function is Media Access Control (MAC). DLL can also provide reliability optionally (through retransmissions). The UNA based system expects mainly the following REQ, RSP and NTFs from DLL: "Send Packet REQ", "Packet Transmitted RSP", "Error RSP" and "Incoming Packet NTF".

The UNA based Physical Layer is defined to provide half duplex transmission and reception of data. It provides modulation and error correction capability and orthogonal channels may be selected via parameters. It uses the following REQ, RSP and NTFs: "Send Packet REQ", "Packet Transmitted RSP", "Error RSP" and "Incoming Packet NTF". Other extensions to provide functionalities such as carrier sense will be added in the next version of UNA.

Though the above REQ/RSP/NTF model for Physical Layer is the same in nomenclature as the DLL, the functionality is different. For example, when the DLL operates in reliable transmission mode, "Packet Transmitted RSP" indicates that a packet has been reliably delivered through re-transmissions, whereas in the Physical Layer, it indicates an un-acknowledged unilateral transmission.

Additionally, both DLL and Physical Layer use parameter setting and retrieval messages "Set Parameter REQ" and "Get Parameter REQ" to set various layer related parameters.

A Framework API (FAPI) was also previously proposed [1] to provide many essential services for a network stack. It is a C API to abstract hardware & OS functionality and handles layer registration, message queues for inter-layer communications and timers. Implementation of FAPI requires a single threaded model, though multi-threaded implementations are possible. The API is shown in Appendix VIII.A.

DLL implementations adhering to FAPI can easily be ported from one system to another as long as the light weight FAPI interface is supported. For systems that do not have FAPI, the interface can easily be added. This framework was identified recently as a potential framework for MAC standardization by an independent group [4].

III. ARL MODEM

Here we briefly look at the acoustic modem where this framework has been implemented. This modem is used for the sea trials described later on in this paper.

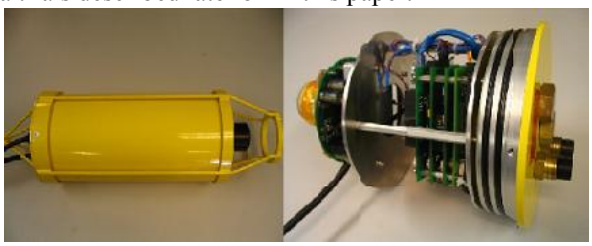

Fig. 2. The ARL modem used in the tests

The ARL modem employs OFDM modulation and operates at 31.25 kHz centre frequency with 12.5 kHz bandwidth. It has a maximum range of 3 km and has up to 7.5 kbps data rate. It supports repetition, convolutional, Golay and LDPC coding. OFDM packets consist of two parts, a detection preamble and a data modulated signal. The detection preamble is a DSSS

sequence to provide reliable detections and accurate frame synchronization. The modem Physical Layer implements the FAPI interface. DLL code can be developed to use this FAPI interface to send and receive data. Various parameters also can be set using the FAPI API.

IV. THE SIMULATOR

The ARL modem simulator was developed using the discrete event simulation package Omnet [5]. It's written in C++ and provides flexible and fast simulation of discrete-event simulations. Each of the layers in the protocol stack is implemented as an Omnet module. The underwater acoustic channel is thus implemented as an Omnet module. For MAC simulation study, three modules – Network, Datalink and Physical layers form a logical node module (represents a single modem). Each node is linked to the channel module.

Accuracy of the simulator have been verified through analysis of timing logs of packets send and received, graphical representation of protocol behavior as well as numerical accounting of packets send, received and lost due to various means like Bit Errors (BER) and collisions. This simulator has been in use for a number of years and were used in studies such as [6] and all subsequent random access protocol studies at ARL.

A. The unified simulator and modem software model

FAPI was designed to be a C language API for the modem so that it is easily implemented in any hardware system. The C++ based Omnet system for the simulator was independently chosen for being freely available for academic use and its simplicity and ease of use. This is linked to a FAPI framework as shown in Fig. 3.

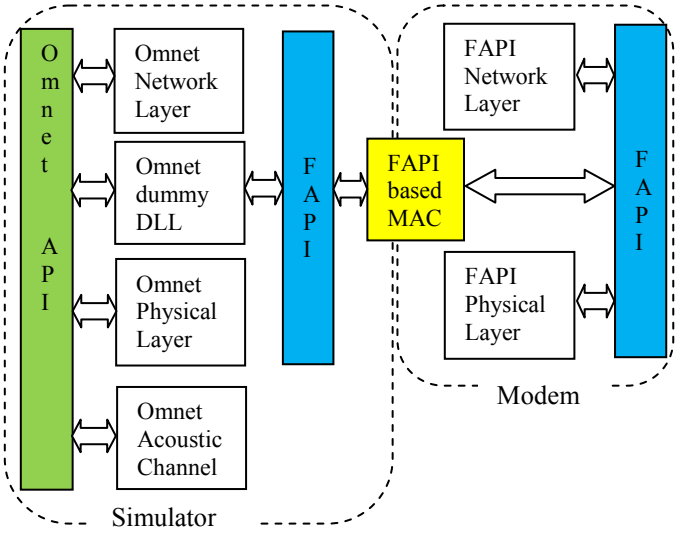

Fig. 3. Simulator and modem software model

Both simulator and modem have FAPI implementations. The simulator implementation of FAPI uses a callback mechanism to interact with Omnet discrete event API through dummy Omnet layer. The same DLL/MAC module, written using FAPI interface, is shown to exist for both modem and simulator. The FAPI based DLL/MAC module in the simulator does not interact with Omnet API directly. This also shows a possible mechanism that can be used by other existing underwater simulators to adopt our proposal to provide a common FAPI interface for the DLL/MAC.

In the simulation, the Omnet Network Layer acts as the data generator for the Datalink Layer below. The data for the Physical Layer from DLL/MAC layer goes to the Omnet Physical Layer which emulates the modem, which in turn is send across the channel module that models collisions and BER.

B. Simulator Physical Layer details

The Physical Layer is assumed to be a half duplex system as is usually the case in most commercial acoustic modems. BER and collisions are simulated at the Physical Layer. A simplified state diagram for the Physical Layer is shown in Fig. 4. (Adapted from [7])

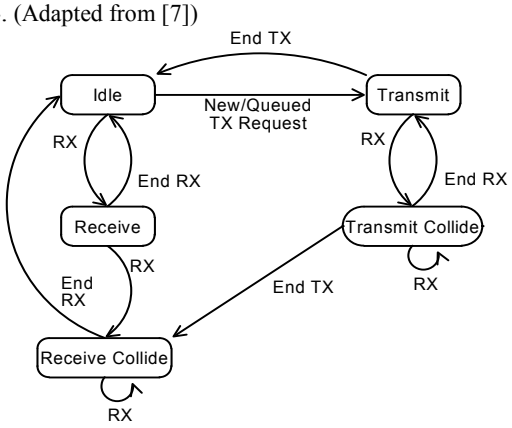

Fig. 4. Simplified Physical Layer State Diagram (Adapted from [7])

RX refers to reception of packets from other nodes and TX refers to transmission. Receive collisions that occur during transmissions do not interrupt the transmission, but the incoming packets will be lost due to the half duplex model. All collided RX packets are considered lost. In a modem operating at sea, there is a possibility that partial collisions could be tolerated and hence in this regard the simulator will give more conservative results in terms of collisions.

In the ARL modem simulator, detection and BER based errors are handled by the simulator's Physical Layer module. All packets are modeled as consisting of two parts, a detection preamble and a data modulated portion following that. Detection is characterized by detection probability P_d and decoding success probability by P. Packet loss probability P_{loss} is then

$$P_{loss} = 1 - P_d P \tag{1}$$

The simplest option to model detection and decoding is to assign fixed values P_d and P to match experimental conditions (see section VI). For greater realism, it is possible to use *SNR* based BER and P calculations in the simulator. The *SNR* is computed as follows where N_0 represents the ambient noise power spectral density.

$$SNR = \frac{\text{Received Power}}{N_0 \times \text{Bandwidth}} \tag{2}$$

The E_b/N_0 ratio is then computed as

$$\frac{E_b}{N_0} = SNR \times \frac{\text{Bit Rate}}{\text{Bandwidth}} \quad (3)$$

Associated BER and corresponding P can then be computed based on a given modulation type such as QPSK using analytical or empirical models. The overall packet loss probability can be then computed as in (1).

We can also use other methods such as range dependent BER profile of particular modems as used in some simulations [6] on moving AUVs as shown in Fig. 5.

We note that in many published simulation results on underwater networks, detection and decoding errors are not considered (only collisions), which would undermine their validity in field trials, since in most modem sea trials, packet BER and detection losses can be a significant factor.

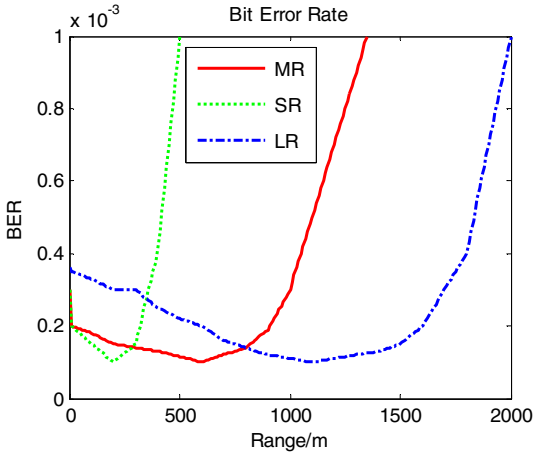

Fig. 5. Coded BER curves for different modem types (MR, SR, LR). The BER has a rapid increase after their maximum range in this model. At lower range the BER increases slightly as compared to each channel's optimum range (for more details see [6]).

C. Channel model details

A simple channel model is used. The channel propagates all packets to all nodes except the sender and accounts for the propagation delay and path loss. The path loss model used is the spherical spreading model:

$$path\ \ loss\ \ in\ \ dB\ = 20\log_{10}(range) \quad (4)$$

More complicated models (e.g. [8]) could be implemented, but the above is deemed sufficient as a first order approximation for the scope of most simulations.

We typically use additive white Gaussian noise (AWGN), although channels vary in noise characteristics such as the warm shallow water channels with impulsive noise. Node motion etc also can be modeled as described in our earlier work [6].

D. Simulator Limitations

A simulator needs to capture essential behavior of the underwater acoustic channel. And for testing a MAC protocol, the Physical Layer emulation has to capture essential behavior of a modem to a reasonable accuracy. For a protocol such as the MACA based protocol illustrated in this paper later on, important characteristics required from the combination of Physical Layer and the underlying channel are packet detection and loss emulation (due to various factors such as path loss, ambient noise, packet collisions) as well as propagation delays. These are currently captured accurately by the simulator.

However, the simulator does not currently model reverberations and multi-path associated with real sea environments. Such characteristics were not deemed necessary to evaluate the performance of the MAC protocol being evaluated at ARL to be used over the ARL OFDM modem. One of the reasons is that OFDM handles multipath effectively by the use of cyclic prefix in each of the packets. The higher MAC layer does not see multi-path effects directly. Such multi-path effects would translate into packet detection and decoding losses at the Physical Layer and this is indeed captured by the current simulator.

However for some type of MAC protocols, such characteristics cannot be ignored in the simulator. For example T-LOHI [9] protocol relies on tone transmissions for channel capture. A tone by itself cannot carry source node or other information and the protocol relies on detection of tones to perform channel capture. If there are reverberations, a node will be unable to distinguish other nodes' tones from its own, and thereby the main mechanism in the protocol will fail. If reverberations are not modeled in the simulator used for testing such a protocol, such problems will not arise and results will not meaningfully predict performance of such a protocol operating in a modem at sea.

The current simulator also does not support different types of "packets" with differences such as coding level or different detection, BER and collision characteristics. This could be useful to model dynamic coding, with control packets such as RTS or CTS using high level of FEC coding where as DATA packets could utilize a lower level of coding (decided via RTS/CTS handshake) to enhance data rate. Such characteristics can be added quite easily in the ARL simulator, as required by the nature of the MAC protocol being tested.

Thus, we wish to highlight that there are simplifying assumptions in all MAC simulators and none can capture the effects of a modem operating at sea completely. There is no validation as important as testing the protocol at sea using a modem. This is the main motivation behind this paper's concept of unifying simulation and implementation framework, which provides MAC protocol developers a way to translate simulation studies to sea trials with little added effort.

V. Writing MAC Code

The MAC is a sub-function of Datalink Layer. MAC function is responsible for deciding when packets are sent and received. The Datalink Layer has other functions such as link adaption (e.g., choosing and setting Physical Layer parameters for optimal performance). This link adaptation could include power control also, since transmission power setting can be viewed as a link parameter. There is inter-play between these sub-functions, for example to do power control, the MAC

control packets might carry transmit power and recommended power information etc. There could also be a queuing sub-system that decides the data to be sent according to priorities, models such as FIFO, lifetime and expiry etc. These functions are represented in Fig. 6.

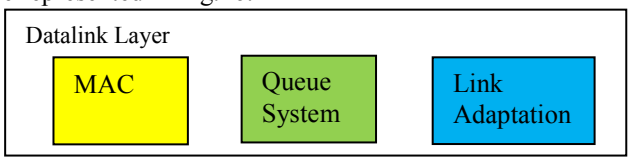

Fig. 6. Key sub-functions of Datalink Layer

In this paper we look only at the MAC function and see how it is implemented in our framework. Appendix VIII outlines the UNA messages, FAPI interfaces and some utility functions provided by the DLL to the MAC function. A sample MAC implementation for a simplistic ALOHA-like protocol is presented in Appendix VIII.D. This code sample shows that it is very simple and similar to the way most MAC protocols are written in a simulation program. And yet since interfaces are standardized across the simulator and the modem, with no changes to the code it runs on both and avoids porting issues and effort. Correct porting is important in order to achieve the performance gains seen in simulations and to help cross verify simulations with experiments. In general, simulator and modem software interfaces could have differences that make porting difficult.

The framework also ensures that the simulation code for the MAC protocol handles all the required complexities in a modem. For example when a MAC protocol requests the Physical Layer to send a packet, the Physical Layer in a modem might be unable to send it at that point in time and choose to return an error. A simple simulation program might not take into account such situations and would require modifications during modem implementation, and it could have potential impact on the algorithm. How the framework handles this aspect is shown in Appendix VIII.D. There could be other important peculiarities in a modem. For example, the ARL modem uses a high power amplifier (HPA) for transmission and it takes about 300ms to settle after turn on. And when HPA is turned on, no receptions are possible. If in a certain MAC protocol, it needs to transmit a control packet right away, it would realize that it has to wait about 300ms before it can actually transmit. For example, in position based collision avoidance protocols, it could significantly affect the main protocol mechanisms as 300ms translates to roughly 500meters. So the HPA might need to be switched on 300ms before the intended transmission to avoid such problems. But once HPA is switched on, receptions are not possible during that period, and this again can have repercussions. Therefore, all such modem limitations and aspects should ideally be captured into simulations in order to develop realizable protocols for a modem implementation and to get accurate assessment of protocol performance through simulations. How the framework handles the HPA aspect is shown in Appendix VIII.C and VIII.D.

VI. MODEM TRIALS AND RESULTS

In this case study, a MAC protocol (essentially based on the original MACA protocol [10] and the popular terrestrial radio wireless 802.11 protocol, and which we evaluated originally in [7]) is implemented and run on both the simulator and in the ARL acoustic modem. The protocol used here is described in detail in [11] and it uses RTS/CTS handshake followed by a batch of DATA packets and an optional ACK packet that indicates the number of packets successfully received. There has been many variations of the same protocol concept for underwater use from early applications such as those reported in the Seaweb project [12], and in the last few years such as Slotted FAMA [13], DACAP [14], PCAP [15] etc.

In the RTS contention algorithm, a node starts with a back-off time, uniformly selected within a fixed window. When the back off timer expires, an RTS is sent. Once RTS is sent, CTS timer starts. If timer expires before reception of CTS, RTS back-off procedure starts again. Once CTS is received, DATA train is sent followed by wait for ACK. If ACK is not received, the RTS cycle repeats. Reception of RTS/CTS packets and a possible DATA frame while waiting to send RTS triggers Virtual Carrier Sense (VCS). Successful DATA transmission for any one node restarts RTS contention cycle for all. This protocol does not use Physical Carrier Sense (PCS). This protocol uses a feature to improve throughput by sending ACK in reply to an RTS (instead of CTS), if the RTS is repeated for the same batch of DATA (i.e. the ACK for a batch not received by DATA transmitter).

During the modem sea trials, packet detection and decoding probability is captured. This is in turn used in the simulations using the simple model of using constant P_d and P mentioned in section IV.B. We show how the throughput performance varies as the batch size is varied. Traffic arrival process is saturated.

A. Sea Trials

Here we look at some results from a preliminary medium range trial in Singapore Coastal waters (south of Pulau Ubin Island) conducted in June 2009. The modems were deployed as shown in Fig. 7 and Fig. 9. Node 2 is in boat "West Squadron", Node 3 is in boat "Dolphin". West Squadron and Dolphin were separated by about 400 meters on both days. Node 1 is in a chase boat that moved about in between the main boats as shown in Fig. 7.

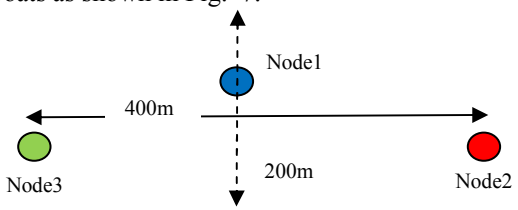

Fig. 7. Node deployment geometry

The separation between Node 2 and 3 was of the order of 400m. The depths were 6 to 12 meters. The modems used were ARL modems. The traffic pattern was Node 1 to Node 2, Node 2 to Node 3 and Node 3 to Node 2. The modems estimated $P_d = 1$ and $P = 0.9$ to 0.95 for the tests shown here.

Thus the primary loss factor is contention and collisions. The packet duration in the modem is 0.6seconds. Also we note that there is a limited PCS present in the modem since the modem hardware does not allow transmission to start if an ongoing reception is in progress. For batch transmission, HPA is turned on/off only at the start and end of the batch only for efficiency. The results are shown in Fig. 8.

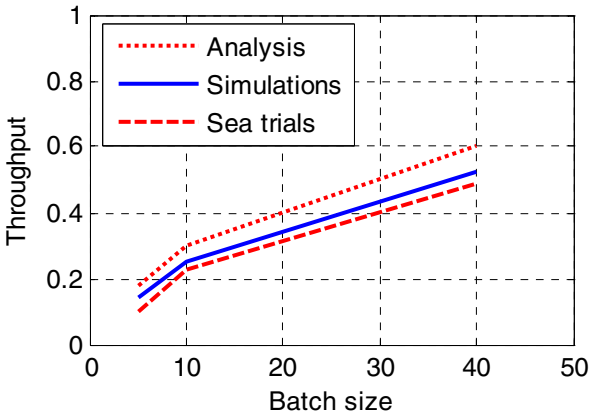

Fig. 8. Sea trials and comparison with simulations

Simulations give a reasonably accurate match. Throughput is seen to improve with batch size, as proven in simulations. As mentioned above, in the high success probability scenario used here, collisions are the most significant source of losses. The analytical results in Fig. 8 are obtained as described in [11]. Note that there are only three actual data points shown in the figure for this particular illustration and the lines are extrapolated. But it is in good agreement with other simulations such as those presented in [11].

Trial results such as above provide a good validation to the simulator's accuracy in modeling propagation delays, packets losses due to BER, collisions etc, and confirm implicitly that it captures the behavior of the modem Physical Layer and the channel that is relevant to the MAC protocol being investigated. Once such as close match with sea-trials is established, the simulator can be relied upon for further investigations for similar protocols.

VII. Conclusion

Through this framework we hope to establish a new paradigm in underwater MAC protocol development. Simulations are used for MAC protocol development since the design/debug/validate cycle is less costly. Being able to use the same code in simulator and in a modem through a published structured software interface is of immense value. Simulation results can meaningfully predict modem performance and aids greatly in MAC protocol development. There is no further porting to be done after a simulation study to conduct a modem trial. Preliminary sea trial results have been provided that show the utility of the framework.

MAC protocol standardization for underwater networks is being attempted by more than one group currently. MAC protocol candidates need to be evaluated on a common platform to provide comparative results [4]. One key aim for the framework proposed in this chapter, is to provide such a platform that has been validated through modem field tests.

There is certainly more work to be done in collaboration with interested researchers who would like to contribute to such initiatives. Current simulator has some limitations as mentioned in section IV.D that needs to be addressed. A technical report will be published online on the ARL website, to provide all the necessary information for other researchers to make use of this framework. As mentioned earlier, there is at least one very closely related project [3] that is looking at unifying interfaces to modem Physical Layer etc to facilitate portable MAC software development. Perhaps in the next phase, it would be prudent to link such parallel initiatives and develop a globally acceptable framework.

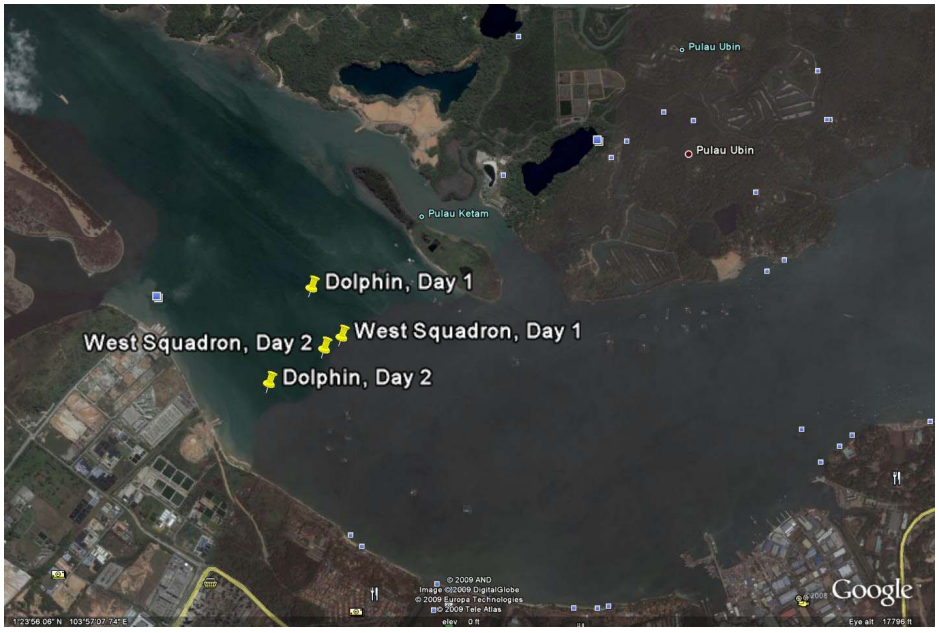

Fig. 9. Medium range sea trial location off the north of Singapore and below Pulau Ubin island. Boats were separated by about 400 to 500 meters during trials. Trial area had very shallow waters of about 6 to 12 meters depth, severe multipath and high ambient noise.

VIII. APPENDIX

A. UNA Messages

This is a short summary of UNA inter-layer messages [1] used in the illustrations and discussion in this paper. We omit parameter set and get messages etc for brevity. Also there are general error responses: ERROR_RSP

1) Key Physical Layer messages

Send a packet	PHY_SEND_PKT_REQ
Packet Transmitted Response	PHY_PKT_XMIT_RSP
Notification to DLL for incoming packet	PHY_INCOMING_PKT_NTF

2) Key Datalink Layer messages

Send a packet	DLL_SEND_PKT_REQ
Notification to Network Layer for incoming packet	DLL_INCOMING_PKT_NTF
Packet Transmitted Response	DLL_PKT_XMIT_RSP

B. Framework API

Here we show some of the key API of FAPI. There has been some changes and enhancements over the proposal in [1]. In the rest of the appendix, C code or related text will be presented in text boxes for clarity.

All layers implement the following message handler to receive messages from other layers.

```
/* "msg" is the incoming message to be handled, "sender" is the sending layer, "hX" is a handle to an arbitrary object to be used by the caller (used my multi- node simulations in the simulator for example) */
void (*MessageHandler)(Message* msg, int sender, UserData hX);
```

The API for sending messages is as follows.

```
/* "msg" is the message, "me" is the sender layer , "destination" is the destination layer, "hX" is as decribed above */
int FAPI_sendMessage(void* msg, int me, int destination, UserData hX); /*
```

Timer set functionality

```
/* "ref" is a reference for the caller to identify the timer, "timeout" is the timeout in millisecs */
int FAPI_setTimer(int me, unsigned int ref, long timeout, UserData hX);
```

Timer cancellation

```
int FAPI_cancelTimer(int me, unsigned int ref, void *hX);
```

To get the current time in milliseconds

```
long FAPI_getTime(UserData hX);
```

C. Datalink Layer Utility Functions

Here we summarize some of the important utility functions provided by the Datalink Layer to the MAC sub-function. We omit parameter set and get utility function etc for brevity. These DLL utilityfunctions isolate the MAC implementation from complexities such as HPA on/off control etc.

1) Reception handling

The following function switches reception off.

```
/* "pStateDLL" is the state information storage of the MAC module - described further in the sample MAC code section later on, "hX" is as decribed above */
void switchOffRx(stateDLL *pStateDLL, void *hX);
```

The following function switches reception on.

```
void switchOnRx(stateDLL *pStateDLL, void *hX);
```

2) HPA control

Though FAPI_sendMessage() function (see section B) can be used directly to send a packet to the Physical Layer for transmission (using UNA message PHY_SEND_PKT_REQ defined in Appendix VIII.A.1), there are complexities such as HPA on/off control etc that needs to be controlled directly by the DLL. In order to isolate MAC module from such complexities, the MAC module uses the following function to transmit a packet.

```
/* "mode" set to 1 if HPA needs to be kept on afer transmission, for example in batch transmission. Else HPA will be turned off after transmission and reception switched on*/
int txPacket(stateDLL *pStateDLL, void *msg, int mode, void *hX);
```

The above reception or transmission handling requires interaction with the Physical Layer. Hence response events will come from Physical Layer and needs to be handled, and that is done by the following function. Appendix VIII.D on sample MAC code shows how this is done.

```
int Tx_Rx_Handler(stateDLL *pStateDLL, Message *msg, void *hX);
```

3) PDU handlers

In order to create a Datalink PDU, MAC can utilize the following function. This isolates the MAC protocol from the packet byte structure. The PDU length must be within the allowed maximum size in the modem, which is related to the level of FEC and other link parameters.

```
/* "type" the type of payload such as 'RTS' or 'CTS', "UID" is unique identifier, "src" and "dest" are node addresses, "param1" an arbitrary parameter, "data" is the MAC data and "datalen" is length of data in bytes*/
DatalinkPDU *generatePDU(BYTE type, int UID, int src, int dest, BYTE param1, BYTE *data, int datalen);
```

There are other PDU related utility functions to get and set fields, duplicate PDUs etc, which are omitted from this appendix for brevity. Its not essential for the following

illustration of how to write a typical MAC module in this framework.

D. Sample MAC Code

Here we present a sample DLL message handler to give a flavor of MAC layer implementation using the framework. The above FAPI API and Datalink utility functions are used to write a typical MAC module. Here we illustrate a trivial "ALOHA" like simple MAC protocol. The code is not meant to compile and meant for illustration only.

1) The main handler interface

This is the main DLL message handler. All events for the Datalink layer come here. "pStateDLL" input parameter, is used to pass in the current state information of the DLL. During processing, the state can be updated and stored until next event. As mentioned earlier (appendix VIII.C.2) , transmission and reception handling will generate responses from the Physical Layer. All events are handled first by the Tx_Rx_Handler() function as shown below to capture such responses, before an event is passed to the actual MAC code.

```
/* "pStateDLL" is the state information
  storage of the MAC module. other parameters
  as defined earlier */
void DLL_handleMessage(stateDLL *pStateDLL,
Message* msg, int sender, void *hX) {
  //input checks
  if (msg != NULL) {
    /*handles tx/rx/HPA related behavior
        before doing processing */
    if (Tx_Rx_Handler(pStateDLL, msg, hX) < 0)
    {
      //insert << MAIN_MAC_HANDLER >>
    }
  }
}
```

2) MAIN_MAC_HANDLER

This is the main MAC handler's overall structure (this fits into the label "*MAIN_MAC_HANDLER*" above) which handles messages from higher and lower layers as well as timers and takes appropriate action.

```
if ((sender == LAYER_NETWORK) ||
      (sender == LAYER_APPLICATION)) {
  //Note the sender layer for future use
  pStateDLL->senderLayer = sender;
  if (msg->msgid == DLL_SEND_PKT_REQ) {
    //insert << DLL_SEND_PKT_REQ >>
  }
}
else if (sender == LAYER_PHYSICAL) {
  if (msg->msgtype == MSGTYPE_RSP) {
    if (msg->msgid == ERROR_RSP) {
      /*Handle the error as appropriate
      For example this could be in response to
        transmission request */
    }
    else if (msg->msgid == PHY_PKT_XMIT_RSP){
      //insert << PHY_PKT_XMIT_RSP >>
    }
  }
  else if (msg->msgtype == MSGTYPE_NTF) {
    if (msg->msgid == PHY_INCOMING_PKT_NTF){
      //insert << PHY_INCOMING_PKT_NTF >>
    }
  }
}
else if (msg->msgid == FAPI_TIMER_EXPIRED_NTF)
{
  //insert << FAPI_TIMER_EXPIRED_NTF >>
}
```

3) DLL_SEND_PKT_REQ

A new packet send request from higher layer comes in here. The actions will depend on the protocol, and in this example we illustrate a random back-off before transmission.

```
DllSendPktReq* dllmsg;
dllmsg = (DllSendPktReq*)msg;
//cancel previous back-off timer
FAPI_cancelTimer(LAYER_DATALINK,
  ref_BACKOFF_TIMER,hX);

/*remove previous message. No queuing in this
example*/
if (pStateDLL->DATAmsg != NULL)
  free(pStateDLL->DATAmsg);

/* pStateDLL->DATAmsg is a pointer to type
DatalinkPDU */
pStateDLL->DATAmsg = generatePDU(_DATA, 0,
  pStateDLL->ownMAC, dllmsg->dstaddr,0, NULL,
  SIZE_DATALINK_PDU-DatalinkHeaderLen) ;

//Do a back off
FAPI_setTimer(LAYER_DATALINK,
  ref_BACKOFF_TIMER, (long)(wait), hX);
```

4) FAPI_TIMER_EXPIRED_NTF

In this example, when the timer expires, we use the txPacket() function to transmit the stored packet. We do not need to handle HPA control etc explicitly. After this action we wait for either PHY_PKT_XMIT_RSP or ERROR_RSP from Physical layer.

```
if (((FapiTimerExpiredNtf *)msg)->ref ==
  ref_BACKOFF_TIMER) {
  /*transmit packet and switch on reception
after transmission*/
  txPacket(pStateDLL, phyreq, 0, hX);
}
```

5) PHY_PKT_XMIT_RSP

This is where we handle transmission success responses from Physical Layer. In this example, we pass a DLL_PKT_XMIT_RSP back to the higher layer. Note that in this example the MAC protocol does a random back off before transmission, i.e. the packet is transmitted only when the back-off timer expires as handled above.

```
//inform higher layer
rsp = FAPI_createMessage(MSGTYPE_RSP,
  DLL_PKT_XMIT_RSP, SIZE_MESSAGE);
FAPI_sendMessage(rsp, LAYER_DATALINK,
  pStateDLL->senderLayer, hX);
```

As mentioned earlier, some transmission requests could meet with error responses which are handled as indicated

earlier (section 0.

6) PHY_INCOMING_PKT_NTF

Packets from other nodes will be handled here. First we extract the PDU from PHY_INCOMING_PKT_NTF UNA message. In this example we just pass up the incoming packet to the higher layer. In MAC protocols with control packets such as RTS, CTS, appropriate next steps will be taken here.

```
/* This MAC utility function extracts the pdu
contained in the Message struct. This function
is written by the MAC developer and not part
of the DLL utility framework */
pdu = checkPacket(pStateDLL, msg, check, 0);

if (pdu != NULL) {
  //send incoming message to the higher layer
  ntf = FAPI_createMessage(MSGTYPE_NTF,
            DLL_INCOMING_PKT_NTF,
SIZE_MESSAGE);
  FAPI_sendMessage(ntf, LAYER_DATALINK,
    LAYER_NETWORK, hX);
  free(pdu);
}
```

The details of this framework will be made available on ARL's website. We hope that this Appendix provides an overview of how the framework is used to implement a MAC protocol that seamlessly works in a simulator and modem.

ACKNOWLEDGMENT

The authors thank the Defence Science and Technology Agency (DSTA) of Singapore who funded the Underwater Acoustic Networks program at ARL. We also acknowledge the developers of Omnet for the academic version of their software that was used for the simulations.

REFERENCES

[1] M. Chitre, L. Freitag, E. Sozer, S. Shahabudeen, M. Stojanovic, and J. Potter, "An Architecture for Underwater Networks," in *OCEANS'06 Asia Pacific IEEE* Singapore, 2006.

[2] E. A. Carlson, P. P. Beaujean, and E. An, "Simulating communication during multiple AUV operations," in *IEEE/OES Autonomous Underwater Vehicles*, 2004, pp. 76-82.

[3] J. Shusta, L. Freitag, and J. Partan, "A modular data link layer for underwater networks," in *OCEANS 2008*, 2008, pp. 1-5.

[4] R. Otnes, T. Jenserud, J. E. Voldhaug, and C. Solberg, "A Roadmap to Ubiquitous Underwater Acoustic Communications and Networking," in *Underwater Acoustic Measurements: Technologies & Results*, Nafplion, Greece, 2009, pp. 557-564.

[5] http://www.omnetpp.org/, "Omnet++, Discrete Event Simulation System."

[6] S. Shahabudeen, M. Chitre, and M. Motani, "A multi-channel MAC protocol for AUV networks," in *IEEE Oceans' 07* Aberdeen, Scotland, 2007.

[7] S. Shahabudeen and M. A. Chitre, "Design of networking protocols for shallow water peer-to-peer acoustic networks," in *Oceans 2005 - Europe*, 2005, pp. 628-633 Vol. 1.

[8] K. Raysin, J. Rice, E. Dorman, and S. Matheny, "Telesonar network modeling and simulation," in *MTS/IEEE OCEANS '99*, 1999, pp. 747-752 vol.2.

[9] A. Syed, Y. Wei, and J. Heidemann, "Comparison and Evaluation of the T-Lohi MAC for Underwater Acoustic Sensor Networks," *Selected Areas in Communications, IEEE Journal on,* vol. 26, pp. 1731-1743, 2008.

[10] P. Karn, "MACA - A new channel access method for packet radio," in *ARRL/CRRL Amateur Radio 9th computer Networking Conference*, 1990, pp. 134-140.

[11] S. Shahabudeen and M. Motani, "Modeling and Performance Analysis of MACA based Protocols for Adhoc Underwater Networks," in *WUWNet'09* Berkeley, California, 2009.

[12] J. Rice, B. Creber, C. Fletcher, P. Baxley, K. Rogers, K. McDonald, D. Rees, M. Wolf, S. Merriam, R. Mehio, J. Proakis, K. Scussel, D. Porta, J. Baker, J. Hardiman, and D. Green, "Evolution of Seaweb underwater acoustic networking," in *MTS/IEEE OCEANS 2000*, 2000, pp. 2007-2017 vol.3.

[13] M. Molins and M. Stojanovic, "Slotted FAMA: a MAC protocol for underwater acoustic networks," in *MTS/IEEE OCEANS'06*, 2006.

[14] B. Peleato and M. Stojanovic, "Distance aware collision avoidance protocol for ad-hoc underwater acoustic sensor networks," *Communications Letters, IEEE,* vol. 11, pp. 1025-1027, 2007.

[15] X. Guo, M. R. Frater, and M. J. Ryan, "Design of a Propagation-Delay-Tolerant MAC Protocol for Underwater Acoustic Sensor Networks," *Oceanic Engineering, IEEE Journal of,* vol. 34, April 2009.