Physical Communication

A Framework for Residual Energy Model in UnetStack Simulator for Underwater Sensor Networks

--Manuscript Draft--

Manuscript Number:	PHYCOM-D-20-00108
Article Type:	Full Length Article
Keywords:	Energy model framework; Network lifetime; Energy depletion; UnetStack
Abstract:	In recent years, Underwater Acoustic Sensor Networks (UASN) has gained much attention from researchers because of its diverse applications. UASNs face several issues and challenges like limited bandwidth, high propagation delay, 3D topology, media access control, routing, resource utilization, and energy constraints. Unlike the nodes in terrestrial wireless sensor networks (TWSNs), UASNs suffer from energy constraints, which severely affects the network lifetime as well as throughput. Simulation of UASNs is a common aspect in researchers as it facilitates analysis of the working and performance of a UASN before it is implemented and deployed, which incurs substantial time and cost. Among the different simulation platforms available for simulating UASNs, UnetStack is one, which is an efficient and well-known tool available for simulating UASN, with significant benefits. But, the present UnetStack does not provide direct functionality for monitoring the energy of nodes during simulations, which is crucial. This paper presents the design and implementation of the residual energy model framework in UnetStack. Additionally, through the experimental simulations, the number of frames transmitted & received, and the depletion of node energy over time is presented. Further, the implemented energy model framework able the researchers in the design of energy-aware routing protocols and load balancing methods.

A Framework for Residual Energy Model in UnetStack Simulator for Underwater Sensor Networks

B. R. Chandavarkar[a], Akhilraj V. Gadagkar[a]

[a] *Department of Computer Science and Engineering, National Institute of Technology Karnataka, Surathkal, Mangalore, India*

Abstract

In recent years, Underwater Acoustic Sensor Networks (UASN) has gained much attention from researchers because of its diverse applications. UASNs face several issues and challenges like limited bandwidth, high propagation delay, 3D topology, media access control, routing, resource utilization, and energy constraints. Unlike the nodes in terrestrial wireless sensor networks (TWSNs), UASNs suffer from energy constraints, which severely affects the network lifetime as well as throughput. Simulation of UASNs is a common aspect in researchers as it facilitates analysis of the working and performance of a UASN before it is implemented and deployed, which incurs substantial time and cost. Among the different simulation platforms available for simulating UASNs, UnetStack is one, which is an efficient and well-known tool available for simulating UASN, with significant benefits. But, the present UnetStack does not provide direct functionality for monitoring the energy of nodes during simulations, which is crucial. This paper presents the design and implementation of the residual energy model framework in UnetStack. Additionally, through the experimental simulations, the number of frames transmitted & received, and the depletion of node energy over time is presented. Further, the implemented energy model framework able the researchers in the design of energy-aware routing protocols and load balancing methods.

Email addresses: brcnitk@gmail.com (B. R. Chandavarkar), gadagkar.akhil@gmail.com (Akhilraj V. Gadagkar)

Keywords: Energy model framework, Network lifetime, Energy depletion, UnetStack
2010 MSC: 00-01, 99-00

1. Introduction

As the demand for marine resource exploitation is increasing, there is increased awareness of applying underwater sensor technologies for oceanic monitoring. Underwater Acoustic Sensor Networks (UASNs) have been widely suggested as promising solutions to enable a variety of marine applications, e.g., pollution monitoring, offshore exploration, navigation assistance, and mine recognition [1]. UASNs face several issues and challenges like limited bandwidth, high propagation delay, 3D topology, media access control, routing, resource utilization, and energy constraints [2]. Typical UASN is composed of several sensor nodes anchored to the ocean bottom and interconnected wirelessly with one or more underwater gateways. The underwater gateways are the specific nodes equipped with both vertical and horizontal transceivers. The vertical transceiver is used to send commands & configuration data to the sensor nodes, and further to obtain the gathered data from it. Whereas, horizontal transceiver is employed to relay the monitored data to the sea surface station [3]. The data is usually relayed within this sensor network from the bottom to the sea surface station through multi-hop paths.

One of the vital factors of any communication networks with battery-operated devices is the life-time. Higher the life-time, the better the system. In comparison with the traditional Terrestrial Wireless Sensor Networks(TWSNs), UASN pose severe challenges in-terms of manpower and budget in manually extending the life-time of the networks. Unlike TWSNs, nodes in UASNs drain their energy due to various reasons. Some of the reasons for the reduced life-time of networks are, environment, high energy consumption by the sensors hardware and poorly designed protocols. Hence, UASN nodes require more energy as compared to TWSNs because acoustic signals cover a long distance, and imple-

ments more complex signal techniques. Further, multi-path routing in UASNs affects the node energy where a node is involved in forwarding data of another node. Additionally, the other major issue that affects node energy is the node mobility. In UASN, nodes are not static like in TWSNs, instead, nodes move due to different activities and circumstances of the underwater environment, usually 2-3 m/sec with water currents [2]. Except surface nodes (sink), most of the deployed nodes in UASNs are energy-constrained and non-rechargeable. However, solar charging power source or regularly replacing drained battery is not an feasible in an underwater environment [1].

Simulation of UASNs is a common aspect in researches as it facilitates the cost-effective and less time consuming approach of the analysis of working and performance of a UASN before it is implemented and deployed. There are several simulation platforms available for simulating UASNs, but not all are open-source. Some of the open-source simulation tools actively available and freely downloadable are: UnetStack[1], Aqua-Sim[2], DESERT[3] and SUNSET[4]. In comparison with Aqua-Sim, DESERT, and SUNSET, UnetStack supports a seamless transition from simulation to real field level deployment without requiring changes in code and design. Hence the compiled binary simulation code can be ported directly to any UnetStack-compatible modem for use in field-level or lab-level testing without requiring any additional cross-compilation [4]. In terms of efficiency, UnetStack supports both discrete-event and real-time operation modes. Thus UnetStack becomes an ideal choice for researchers to conduct UASN simulations and then to transit for real field level deployment.

MATLAB and NS-2 based simulations of UASNs are also quite popular in literature. In case of MATLAB based simulations, most of these are application specific. Though the MATLAB can be used for conducting an in depth simulation, there is no facility to define custom topologies, power models. Also there

[1] https://unetstack.net/
[2] https://github.com/rmartin5/aqua-sim-ng
[3] http://desert-underwater.dei.unipd.it/
[4] http://reti.dsi.uniroma1.it/

is no provision to define methods to monitor factors like packet transmissions, losses and collisions that might interest the research community and might even affect the performance of underwater network. Further, in MATLAB simulations, support for any routing protocols is unavailable. Another important issue to be considered is the mobility of nodes, and the ability to simulate node mobility is also unavailable in this simulator [5].

NS-2 is an open source discrete event simulator. NS2 is based on C++ and Object-Tcl(OTcl) based scripting. The latest version available is NS-2.36. NS2 does not provide any built-in packages supporting UASN simulation except in NS-2.30. Also, very few tutorials and code samples are available on NS-2.30 as it is an earlier version. Further simulation scripts written in versions above NS-2.30 is difficult to run. Therefore simulating UASN in NS-2.35 and above requires to model all underwater channel characteristics and propagation model [6].

Similar to NS-2, NS-3 is also an open source, discrete event network simulator. It was built using C++ and Python with scripting facility and supports visualization [7]. The NS-3 UAN framework facilitates the modeling of underwater network scenarios [6]. The UAN model has four main components: The channel, PHY, MAC, and Autonomous Underwater Vehicle (AUV) models. The framework intends to simulate behaviour of AUVs. The communications stack, associated with the AUV, can be modified depending on simulation needs. Typically, the default underwater stack comprises a half duplex acoustic modem, an Aloha MAC protocol, and a generic physical layer.

However, UnetStack does not provide direct functionality for monitoring the energy of nodes during the simulations. This paper presents the design and implementation of the residual energy model framework in UnetStack. Additionally, through the experimental simulations, the number of frames transmitted & received, and the depletion of node energy over time is presented. Further, the implemented energy model framework in UnetStack able the researchers in the design of UASN's energy-aware routing protocols and load balancing.

The rest of the paper is organized as follows: Section 2 presents related work;

Section 3 describes the residual energy model for UASN; Section 4 gives details on implementation of energy model in UnetStack simulator; Section 5 presents the results and analysis. Finally, conclusions and future scope are drawn in Section 6.

2. Related work

This section of the paper presents the significance of energy model implementation in UnetStack from the perspective of different research aspects of underwater communication. Further, need for the energy model implementation in UnetStack platform. Mandar Chitre et al. [8], developer of UnetStack presented the detailed introduction and overview of UnetStack architecture. Additionally, details on different services available and the set of pre-defined agents offering the services are also presented by the authors in their contribution.

2.1. Significance of energy model in underwater research

As discussed in Section 1 monitoring node energy in UASN is important in various aspects of underwater research like energy consumption analysis, design of energy-based routing protocols, data collection strategies, mitigating energy holes, and load balancing. This subsection describes significance of energy monitoring in the above mentioned research problems.

Guangjie Han et al. [1] proposed an Asymmetric link-based reverse routing protocol (AREP) to ensure bi-directional data communication between source and destination nodes. The authors explored the impact of the directional beam width of underwater nodes on communication links. Further the authors have conducted three case studies of communication links. These case studies indicate, for directional antennas of fixed beam width, the change in a relative position of two geographically adjacent nodes is prone to generate asymmetric links. Performance of AREP is compared against Link-state based Adaptive Feedback Routing (LAFR). Simulation results show that, in terms of energy consumption AREP consumes less energy than LAFR. Also AREP provides

low transmission delay and high packet delivery in an underwater environment. In simulating AREP, the paper presents mathematical equations to calculate the energy consumed in transmitting and receiving a m-bit packet.

Mari Carmen Domingo and Rui Prior [9] presented a mathematical analysis of total energy consumption in underwater acoustic networks in shallow and deep water scenarios. The authors calculated the total energy consumption via direct links or via relaying. Relaying or a routing protocol based clustering for both shallow water and deep water scenarios were studied. Finally the authors conclude routing protocols based on the clustering scheme save more energy, and they show a better performance in shallow water.

Guanglin Xing et al. [10] deployed a UASN on named data networking (NDN) architecture and explored the energy consumption of the NDN-based UASN under shallow water and deep water conditions based on the relay network topology. Simulations were conducted in NS-3 and MATLAB to analyze the results of energy consumption models of NDN-based relay UASNs in shallow water and deep water. Results obtained through MATLAB simulations may not actually model real UASN. MATLAB does not provide any method to define custom topologies or methods to monitor factors like packet transmissions, collisions and losses.

Jing Yan et al. [11], in their paper on the underwater cyber-physical system (UCPS) proposed a new two stage solution for energy-efficient data collection for UCPS over autonomous underwater vehicle (AUV) assisted underwater acoustic sensor network. In the first stage, sensor nodes are deployed in a rigid graph-based topology and relay physical data to a short range data collector through multihop acoustic communication. Then in the second stage, data collectors are visited periodically by an AUV to retrieve data through high-speed visible light communication. Simulations are conducted in MATLAB 2016b and shown that the topology optimization scheme can prolong the network lifetime. Though the MATLAB can be used to conduct in-depth simulations, there is no support for any routing protocol.Moreover, with AUV's the mobility is also very important, but the ability to simulate node mobility is not supported in MATLAB

Anwar Khan et al. [12] presented a localization-free interference and energy holes minimization (LF-IEHM) routing protocol for underwater wireless sensor networks (UWSNs). The authors proposed an algorithm to overcome interference during data packet forwarding by defining a unique packet holding time for every sensor node. A localization-free energy holes mitigation is done using a variable transmission range of sensor nodes. Here a node can increase its transmission range to include one or more live forwarder nodes, in a case when no node is available within its transmission range. Results are presented for simulations performed in MATLAB and performance metrics considered include total energy consumption, dead nodes, live nodes.

In many research works, the evaluations are done either by performing numerical analysis or by conducting simulations using tools like MATLAB or NS-3. Though the results of these prove the performance of proposed methods, the same cannot be used for real-time implementations. However, in the UnetStack simulator, it allows protocols developed to be ported directly to software-defined modems. Hence the results obtained through UnetStack can be used to prove not only in simulations but also in a real-time deployment. Thus the energy model in UnetStack implemented in this paper demonstrates its necessity.

2.2. Need for energy model implementation in UnetStack

As discussed in Section 1 though there are numerous simulation and experimentation tools available for simulating UASNs. The UnetStack, Aqua-Sim, SUNSET and DESERT are few such tools which are actively available and freely downloadable. The Aqua-Sim [13] built on top of the most popular NS-2 simulator is a packet level simulation platform. Similar to NS-2, Aqua-Sim also uses the object oriented design style and provides abundant underwater protocols for researchers. However, currently, in Aqua-Sim there is no support for a seamless transition from simulation to field level deployment as it focuses only on simulation and emulation. DESERT [14] and SUNSET [15] are the simulation, emulation and experimental tools, built on top of NS-2 and NS2-Miracle. DESERT provides seamless transition flexibility among simulation, emulation,

and field-level testing. SUNSET, as similar to DESERT, provides facility for seamless transition support among simulation, emulation, and field-level testing.

In terms of efficiency, both DESERT and SUNSET are extended from NS-2. Hence they are also constrained with discrete-event characteristics. For example, both platforms have a main single-threaded process, and the events are scheduled to execute sequentially by a strict event scheduler that is sensitive to time-restricted events. On the other hand, UnetStack supports both discrete-event and real-time operation modes. So the compiled binary simulation code can be ported directly to any UnetStack-compliant modem (e.g., Subnero) for use in field-level or lab-level testing without requiring any additional cross-compilation [4].

Further, in the case of a seamless transition from simulation to emulation or real field level deployment, both SUNSET and DESERT are built on top of Ns-2 and Ns2-Miracle, which are primarily discrete-event simulators. Hence, significant changes may be required in code and design, while transiting from discrete-event-based simulation to distributed real-time emulation on these platforms, which may cause additional problems. For instance, when simulating a certain application, if it uses centralized global network information, then while conducting emulation of the same application, it requires special caution to be taken when transferring the code from the purpose of simulation to emulation. For example, it is challenging to identify event-timing related problems in the simulation. So, while exporting the code written for the simulation to the emulation, many consistency-related issues may happen in this context. Also, in the DESERT, the packet conversion method is not that convenient, which may lead to higher packet conversion overhead [4].

UnetStack uses agent-based architecture and supports real-time simulation. Hence, the same compiled binary code used in the simulation can be ported directly to UnetStack-compatible underwater modems without requiring any cross-compilation for emulation or field-level testing. Thus the UnetStack becomes an ideal choice for researchers for conducting their experiments, and hence the implementation of the energy model, which is currently unavailable

in UnetStack, is essential.

3. Residual energy model for UWSN

The main purpose of the proposed work is to provide a residual energy model framework to further investigate energy dependent algorithms in UnetStack. The residual energy deals with the left out energy at the node after the deduction of consumed energy during transmission, reception, etc. In this proposed work, a residual energy module presented by the authors Guangjie Hana et al. [1] is implemented in UnetStack. A detailed implementation of the existing energy model framework presented in the following Section 4 can be further extended/modified as per the specific needs of the researchers or industry. In this proposed work, the initial energy of a node is deducted for every transmission and reception of a packet. As presented by the authors Guangjie Hana et al. [1] Equations. (1) and (4) indicates the energy consumed during transmission and reception of m-bit packet.

3.1. Energy consumption during the transmission of m-bit packet

The following Equation (1) indicate the energy consumption during the transmission of m-bit packet.

$$E_{tx}(m,l) = m * E_{elec} + m * T_b * C * H * l * e^{\vartheta(f)*l} \tag{1}$$

Where,

- E_{elec} - the energy consumed by the transmitter electronics to process one bit of data,
- l - the transmission distance,
- T_b - bit duration,
- H - water depth,

- C - an empirical constant calculated using Equation. (2)

$$C = 2\pi * 0.67 * 10^{-9.5} \tag{2}$$

- $\vartheta(f)$ - a frequency dependent medium absorption coefficient (in db/km) calculated using Equation. (3)

$$\vartheta(f) = 0.036 * f^{3/2} \tag{3}$$

  Where, f is frequency of sound wave (in kHz) underwater.

3.2. Energy consumption during the reception of m-bit packet

The following Equation (4) indicate the energy consumption during the reception of m-bit packet.

$$E_{rx}(m, l) = m * E_{elec} \tag{4}$$

4. Implementation of energy model in UnetStack

This section of the paper presents the implementation of residual energy model framework (ref Section 3) in UnetStack. Initially an overview of UnetStack followed by physical agent with the details on simulated modem used in UWSN simulation is presented. Finally, a detailed implementation of residual energy model framework is presented.

4.1. Overview of UnetStack

UnetStack developed under the Unet project at Acoustic Research lab of National University of Singapore in 2004. UnetStack is an agent-based stack, and it forms the backbone of the underwater network simulator that can be easily used for deployment and testing of underwater networks.

In UnetStack, the default stack, provides a collection of software agents representing different layers of the network stack. These agents offer well-defined

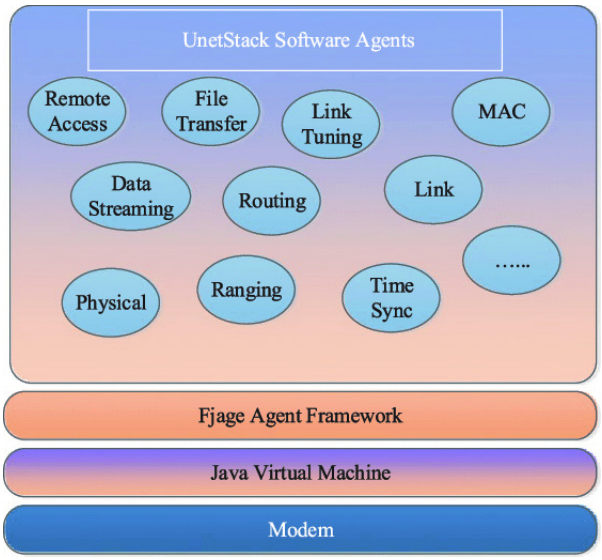

Figure 1: UnetStack architecture [4]

services as of layers in the network stack. This agent-based approach results in having a flexible network stack, with cross-layer based solutions, allowing software-defined underwater networks to be rapidly designed, simulated, tested, and deployed. Further, since the stack is expandable, one can add new agents and services or replace existing agents to meet the developmental requirement.

The architecture of UnetStack as shown in Fig. 1 uses service oriented architecture approach, where the stack defines a collection of software agents that provide well defined services. These agents work with each other to deliver a complete solution for underwater networking. Agents play the role equivalent to the layers in traditional network stack. To achieve this agent based approach, UnetStack uses the open source fjage (Framework for Java and Groovy Agents) lightweight agent framework [5]. The fjage framework provides core implementations which forms the basis for UnetStack. Further UnetStack uses fjage to define agents and their services in its stack. Thus agents are basic building blocks of UnetStack, they exchange messages, provide services and implement

[5] https://github.com/org-arl/fjage

protocols. One can also develop their own agents, for which the UnetStack provides the UnetAgent base class implementing most of the core necessary behaviors of a well behaved agent.

An agent in UnetStack is a self-contained software component that provides a well-defined functionality and have more flexible interactions with other agents. Agents interact with each other through *messages*. Message types used in UnetStack can be classified as *requests*, *responses* and *notifications*. Responses always have an associated request, where as notifications are unsolicited. Messages need not always be sent for a particular agent, but can also be broadcasted on a *topic*. Any agent subscribing to that topic, receives message broadcasted on that topic. Unsolicited notifications usually are sent on topics associated with an agent, since an agent does not know in advance that which other agent is interested in that notification. A well integrated set of requests, responses, and notifications providing a coherent 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 prior information about 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 [8].

The major services defined by UnetStack are Physical service, Datagram service, MAC service, Routing and Route maintenance service, Transport service, Remote Access service, Ranging service, Link service and Node Information service [8].

4.1.1. UnetStack Simulator

UnetStack simulator (aka "UnetSim") is a component of UnetStack. UnetSim is an efficient and well-known simulator available dedicated to underwater network simulations. UnetSim enables to simulate an underwater network of many nodes on a single computer. It can simulate an underwater network in real-time or as a discrete event simulation. UnetSim is easy to install, learn and

use, and once an agent is developed and tested in simulator, it is ready to be deployed and tested in underwater with any UnetStack-compatible modems.

In UnetSim a simulation scenario is describe in a Groovy domain specific language (DSL) script. The script describes location, motion and sets up network stack to be used for node. Simulations can be interactive using shell, or behaviors can be set up to generate network traffic for an automated simulation. Further, if required the script can be made to collect network performance statistics and display them [8].

4.2. Physical agent of UnetStack

In developing energy model, the right place to keep track of residual energy is at the physical layer. In UnetStack it is the physical agent, often abbreviated as `phy`, provides the typical physical layer functionalities of an underwater node. In UnetStack the physical agents are modem drivers and simulated modems [8]. For conducting simulations an implementation of simulated generic modem named as `HalfDuplexModem` is available. The `HalfDuplexModem` simulates the behavior of an underwater modem considering parameters specific to underwater channel, and handles all data transmissions and receptions on a half-duplex channel. All UnetStack simulations uses the `HalfDuplexModem` as default physical agent. The `HalfDuplexModem` supports three different services namely, datagram, physical and baseband service. Each service has set of messages handled by the physical agent, for example, the `DatagramReq` message under the Datagram service requests the agent to send some data.

The implementation of the `HalfDuplexModem` available in org.arl.unet.sim package is represented in two Algo. 1 and Algo. 2. The Algo. 1 depicts working of `HalfDuplexModem` when a request is made to send data using `DatagramReq` message of datagram service. Messages under physical and baseband services are also handled in similar way.

Many agents support `DatagramReq`, with `HalfDuplexModem` a `DatagramReq` asks the physical agent to send some data at the physical layer, As mentioned earlier a request message is associated with a performative response message.

Algorithm 1 UnetStack: `HalfDuplexModem` for sending a datagram

```
1: function processRequest(msg)
2:     if msg instanceof DatagramReq  then
3:         if ¬upadateInfo() ∨ location ≠ null then
4:             req ← TxFrameReq(msg)
5:             handleTxFrameReq(msg, req)
6:             if req.getReliability() then
7:                 data ← req.getData()
8:                 type ← req.getType()
9:                 pro ← req.getProtocol()
10:                if type ≤ 0 ∨ type ≥ FrameLength ∨ pro < 0 ∨ pro > 15 then
11:                    tx ← createTx(Req)
12:                    if tx == null then
13:                        tracer.enqueued(tx, null)
14:                        if OngoingTx ∧ OngoingRx == null then
15:                            startTx(tx)
16:                            platformSend(tx)
17:                            channel.addTx(tx)
18:                            add new WakeBehavior(tx.txDelay)
19:                            tracer.sent(OngoingTx, null)
20:                            sendTxFrameNtf(HalfDuplexModem, this.OngoingTx)
21:                            endTx()
22:                            tx ← (TX)this.queue.poll()
23:                            if tx ≠ null then
24:                                startTx(tx)
25:                            else
26:                                return msg, Performative.AGREE
27:                            end if
28:                        else
29:                            add tx to transmission queue
30:                        end if
31:                    else
32:                        return msg, Performative.FAILURE
33:                    end if
34:                else
35:                    return msg, Performative.REFUSE
36:                end if
37:            else
38:                return msg, Performative.REFUSE
39:            end if
40:        else
41:            return msg, Performative.FAILURE
42:        end if
43:    else
44:        process TxFrameReq or clearReq or TxBasebandReq
45:    end if
46: end function
```

In case of `DatagramReq` message the possible responses are `AGREE`, `REFUSE` and `FAILURE`. When a `DatagramReq` message is received by the `HalfDuplexModem`, it delegates the message instance to 'processRequest()' method to handle it. The method processes the request and returns one of the responses. If request is honoured successfully, returns an `AGREE`, in other cases returns `FAILURE` or a `REFUSE`. The 'processRequest()' in its definition invokes the 'handleTxFrameReq()' to handle transmission of this datagram, which in turn invokes 'createTx()' method to create a new transmission instance, the properties and necessary methods associated for a transmission are defined in a inner class named `Tx` which extends the fjage's `Message` class and implements `Transmission` interface. The 'createTx()' computes and sets the properties and returns the transmission instance, or a null value, then the 'enqueued()' method of `Tracer` class is invoked and an enqueue event is logged in the trace file, thereafter if there is no ongoing transmission or reception, the transmission is beginned calling 'startTx()' where the transmission is added to channel and invkokes 'sent()' of `Tracer` and logs a dequeue event to the trace file and then calls 'sendTxFrameNtf()' to send transmit frame notification after that 'endTx()' is invoked to end the current transmission and poll the queue for next transmission, if exists again calls 'startTx()'.

The Algo. 2 depicts the process of `HalfDuplexModem` when receiving a datagram (packet) sent using `DatagramReq` message. As the message arrives the 'processMessage()' method is invoked, the reception of datagram is handled as instance of `RX` class, which is the inner class and extends the `TX` class defining the properties and necessary methods for a reception. the 'processMessage()' creates a new instance of `RX`, representing a new reception and performs calculations as per the channel model. The default channel model used is `ProtocolChannelModel`. The modem also uses methods of channel model to decide whether it can neglect, detect and decode the reception. If a reception is detected and decoded successfully, invokes the 'received()' of `Tracer` and logs a receive event and then invokes 'sendRxFrameNtf()' to send received frame notification. If decoding is unsuccessful invokes the 'dropped()' of `Tracer` to

Algorithm 2 UnetStack: `HalfDuplexModem` for receiving a datagram

```
1: function processMessage(msg)
2:     if msg instanceof TX  then
3:         if ¬upadateInfo() ∨ location ≠ null then
4:             return
5:         else
6:             RX old ← (RX).this.rxPending.get(tx.uuid)
7:             if old ≠ null then
8:                 old.bad ← TRUE
9:                 return
10:            else
11:                RX rx ← new RX(tx,this.address, null)
12:                rx.rxLocation ← this.location
13:                rx.range ← 0.0
14:                if canNeglect() == TRUE then
15:                    return
16:                else
17:                    rxPending.put(rx.uuid,rx)
18:                    add new WakerBehavior(dtms+rx.txDelay)
19:                    HalfDuplexModem.this.startRx(rx)
20:                    this.channel.addArrival(rx)
21:                    if RxEnable == TRUE then
22:                        return
23:                    else
24:                        if getBusy() then
25:                            if ¬Txongoing ∧ detectRx then
26:                                sendCollisionNtf(rx)
27:                                tracer.dropped(address, rx, "COLLISION")
28:                            else
29:                                return
30:                            end if
31:                        else
32:                            if detectRx() then
33:                                set OngoingRX with current rx
34:                                add new WakerBehavior(dtms + rx.txDelay)
35:                                HalfDuplexModem.this.endRx(rx)
36:                                xxx ← this.channel.decodeRx(rx)
37:                                if xxx == 0.0F ∧ ¬rx.bad ∧ ... then
38:                                    sendRxFrameNtf(rx)
39:                                    tracer.received(address, rx, null)
40:                                else
41:                                    sendBadFrameNtf(rx)
42:                                    tracer.dropped(address, rx, "BAD FRAME")
43:                                end if
```

Algorithm 2 continued

```
44:                             else
45:                                 tracer.dropped(address, rx, "NOTDETECTED")
46:                             end if
47:                         end if
48:                     end if
49:                 end if
50:             end if
51:         end if
52:     else
53:         return
54:     end if
55: end function
```

log drop event with message indicating the reason.

4.3. Residual energy model in UnetStack

In implementing the energy model the `HalfDuplexModem` class is extended further implementing the class named `EnergyModelModem`, which adds energy monitoring capability. The hierarchy of classes extended is shown in Fig. 2 and Algo. 3 presents the implementations made in extended `EnergyModelModem` class.

The extended `EnergyModelModem` class should be able to handle incoming datagram requests for which it should advertise Physical and Datagram services, by registering with them. Further the extended modem should monitor each transmissions and receptions to calculate the energy consumption for a node. To achieve this the extended modem agent should monitor two physical agent notification messages, which are the transmit frame notification(`TxFrameNtf`) and received frame notification(`RxFrameNtf`) the base (`HalfDuplexModem`) class send these notification messages upon handling a nodes data transmission and reception respectively. To send these messages the `HalfDuplexModem` class invokes the send() defined in fjage's `Agent` and passes the instance of message as parameter, since the `HalfDuplexModem` is extension of `UnetAgent` class and in turn `UnetAgent` is extension of fjage's `Agent` class, the 'send()' is invoked from `Agent` class, which is at topmost in the hierarchy. Thus in the extended

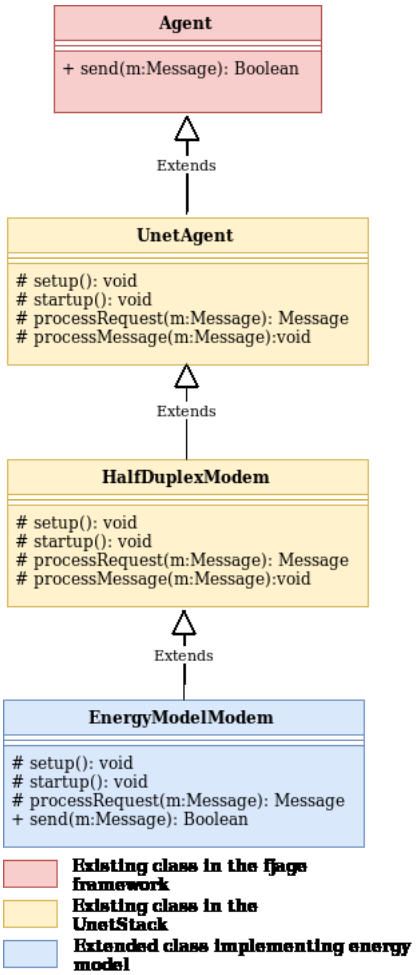

Figure 2: UnetStack: Energy model class diagram

`EnergyModelModem` which comes at last level in this hierarchy, this 'send()' is overriden and defined to check whether the message instance received as parameter is an instance of `TxFrameNtf` or `RxFrameNtf` message, the Energy consumed for data transmission is calculated when parameter is instance of `TxFrameNtf` message, and reception if it is of `RxFrameNtf` message, as our energy model design for transmission energy calculation requires computing size of data and distance between sender and receiver nodes, we need to obtain data being sent from the node and location of node,these are fetched by handling the datagram

Algorithm 3 UnetStack: Energy model in the extended class

```
1: function SETUP
2:     register(Services.PHYSICAL)
3:     register(Services.DATAGRAM)
4: end function
5: function STARTUP
6:     nodeInfo ← agentForService(Services.NODE_INFO)
7:     addr ← nodeInfo.address
8:     depth ← nodeInfo.location[2]
9:     map.put(addr, nodeInfo.location)
10: end function
11: function PROCESSREQUEST(msg)
12:     if msg instanceof DatagramReq then
13:         req ← new TxFrameReq((DatagramReq)msg)
14:         data ← msg.getData()
15:         return new Message(msg, Performative.AGREE)
16:     end if
17:     return null
18: end function
19: function SEND(msg)
20:     if msg instanceof TxFrameNtf then
21:         loc1 ← map.get(addr)
22:         loc2 ← map.get(req.getTo())
23:         x ← loc1[0] - loc2[0]
24:         y ← loc1[1] - loc2[1]
25:         z ← |loc1[2]- loc2[2]|
26:         distance ← √(x^2 + y^2 + z^2)
27:         bits ← 32
28:         sbits ← bits * sdata.size()
29:         dist ← distance/1000.0
30:         Tx ← sbits*50e^-9 + sbits*(0.001)*dist*(depth*−0.001)*C*E^(d*dist)
31:         initenergy ← initenergy − Tx
32:     end if
33:     if msg instanceof RxFrameNtf then
34:         rdata ← msg.getData()
35:         bits ← 32
36:         rbits ← bits * rdata.size()
37:         Rx ← rbits * 50e^-9
38:         initenergy ← initenergy − Rx
39:     end if
40: end function
```

request message which is generated when node request for data transmission.

Finally, in simulation, the `EnergyModelModem`, customized for residual en-

Table 1: Parameters for energy model

Parameter	**Value**
Channel center frequency (f)	10 kHz
Empirical constant (C)	1.3312e-9
Thorp's constant ($\vartheta(f)$)	$0.036 * f^{1.5}$
Bit duration (T_b)	0.001 sec

ergy monitoring is added by configuring the modem settings, thereby it is added as physical agent to the container for use by nodes running in simulation. The parameters for used in implemented energy model with reference to Equation (1) and (4) in Section 3 is shown in Table 1. Two other parameters Water depth (H) and Transmission distance (l) which are simulation dependent are shown in Table 2

Table 2:
Simulation parameters

Parameter	**Value**
Number of nodes	7
Communication range	1000 m
Simulation time	1 min., 1hr.
Initial energy	10 J
Water depth (H) (Node 1 through 7)	0, 800, 1500, 2200, 2800, 600, 2000
Transmission distance (l)	Euclidean distance between nodes, as shown in Fig. 3

5. Results and analysis

This section of the paper presents the simulation topology, parameters and the different scenarios adopted to validate/demonstrate the implemented residual energy model in UnetStack. Further, the topology is simulated for the duration of 1 min. (Case-I) and 1 hr. (Case-II). Additionally, Case-I and Case-II are simulated for with and without acknowledgment (ACK). The successful implementation of the residual energy model in UnetStack is demonstrated through the parameters such as, total number of packets sent/forwarded and received for all active nodes with their relative energy depletion. Case-I provides the

in-depth simulation of the implemented module, whereas, Case-II assures the robustness of the implemented module with the exhaustive simulation.

The simulation with hop-to-hop ACK ensure the reliable delivery of packets. In UnetStack simulations, The `uwlink` agent of type `ReliableLink` available in the default stack, provides link layer service with segmentation/reassembly and link level reliability. When `uwlink` is used in sending data, the hop-to-hop ACK can be enabled for reliable delivery. This can be done using the `reliability` field set to true. In case when `router` agent used for sending data, the `uwlink` is the default link agent. Where the hop-to-hop ACK feature is available by default, or it can be explicitly controlled while adding the routes, using the `reliability` field set either to true or false. The `router` agent provides routing service based on routing table.

5.1. Simulation topology and parameters

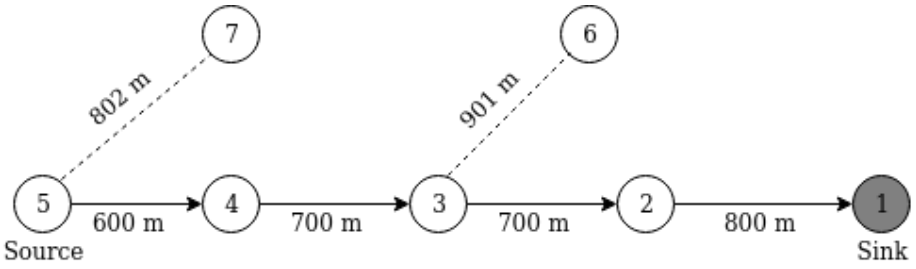

Figure 3: Simulation topology

This subsection presents the simulation topology with its parameters and the values related to the implemented residual energy model in UnetStack. Fig. 3 presents the simulation topology consisting of 7 nodes. As shown in the Fig. 3, circles represents the node, with the dotted line connecting the nodes as neighbours to each other. Further, solid line with the arrow indicate the flow of data and the value alongside the line indicate the Euclidean distance (l) between the connecting nodes. The simulation parameters and the parameters configured for energy model H and l with reference to Equation (1) and (4) in Section 3 is shown in Table 2.

5.2. Case I:

This subsection presents the simulation of the topology as shown in Fig. 3 for the duration of 1 min., with and without ACK. Further, Node-5 and Node-1 are configured as a source and sink respectively.

5.2.1. Total packets sent/forwarded

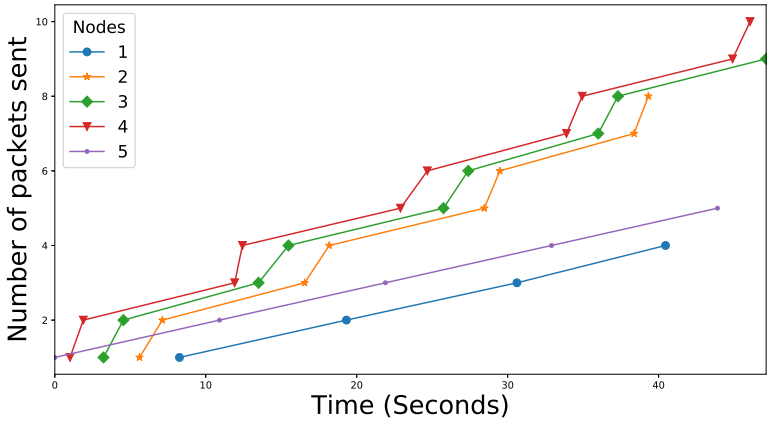

Figure 4: Total packets sent/forwarded with ACK

The multi-line graph in Fig. 4 depicts the total number of packets sent/forwarded by all the nodes with hop-to-hop ACK set in the simulation. As the Node-1 is sink, it sends only ACK packets. Packets sent by Node-4 include, packets to forward to Node-3, and ACK's sent back to Node-5. Similarly packets sent from Node-3 and -2 include packets forwarded to Node-2 and -1 respectively and ACK's sent back. Lastly Node-5 being the leaf node, has its own packets sent.

As shown in Fig. 4, every marker on a graph indicate either packet forwarded or ACK sent event except Node-5 which involved only in packet sent and Node-1 ACK sent. Further, as shown in Fig. 4, 5 packets are sent by Node-5 and accordingly, 10 sent event (5 packets + 5 ACKs) is recorded at Node-4 and -3. Whereas, at Node-2 9 sent event (5 packets + 4 ACKs) is recorded and

accordingly 4 ACKs sent event is recorded at Node-1. Additionally, delay in packet forward and ACK sent event can also be observed in Fig. 4.

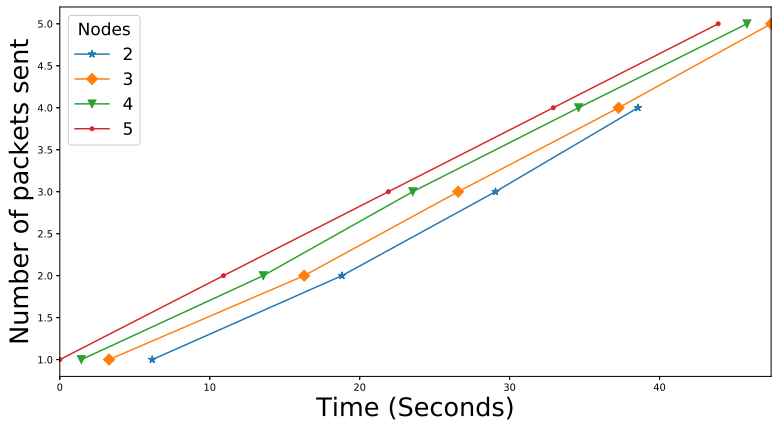

Figure 5: Total packets sent/forwarded without ACK

Fig. 5 depicts the total number of packets sent by all the nodes without ACK set in the simulation. It can be noticed that, there is no graph for Node-1, as it is the sink node and here it does not send ACK's too. Packets sent by Node-4 include packets to forward to Node-3. Similarly packets sent from Node-3 and -2 include packets forwarded to Node-2 and -1 respectively. Lastly Node-5 being the leaf node, has its own packets sent.

As shown in Fig. 5 without ACK, number of packets sent by Node-5 matches with the number of packet forwarded by the intermediate Node-4 and -3. Whereas, at Node-2, packet forwarded count is one less than the previous node because of end of simulation.

The numerical values shown in Fig. 4 and 5 justify the proper functioning of the implemented energy model in UnetStack for packet sent/forward event.

5.2.2. Total packets received

The multi-line graph in Fig. 6 depicts the total number of packets received by all the nodes with hop-to-hop ACK. As the Node-1 is sink, it receives only data packets forwarded from Node-2. Node-5 which is the leaf node, receives

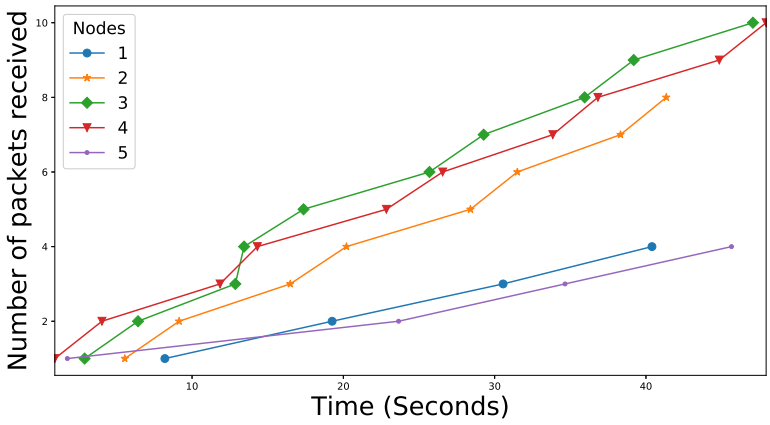

Figure 6: Total packets received with ACK

only ACKs sent from Node-4. Remaining nodes receive packets forwarded from previous hop and ACK's sent from next hop.

As shown in Fig. 6, every marker on a graph indicate either packet or ACK received event except Node-5 which involved only in ACK received and Node-1 packet received. Further, as shown in Fig. 6, 5 packets are sent by Node-5 and accordingly, 10 received event (5 packets + 5 ACKs) is recorded at Node-4 and -3. Whereas, at Node-2, 8 received event (4 packets + 4 ACKs) is recorded and accordingly 4 packets received event is recorded at Node-1. Additionally, packets sent and received events can be mapped in Fig. 4 and 6 respectively.

Fig. 7 depicts the total number of packets received by all the nodes without acknowledgement. It should be noticed that, there is no graph for Node-5, as it is the source node, and does not receive any ACK's. The Node-1 being sink, receives only data packets forwarded from Node-2. Remaining nodes receive packets forwarded from previous node.

As shown in Fig. 7 without ACK, number of packets received by Node-1 matches with the number of packet forwarded by the intermediate Node-2 (Fig. 5. Similarly with Node-3 and -4.

The numerical values shown in Fig. 6 and 7 justify the proper functioning

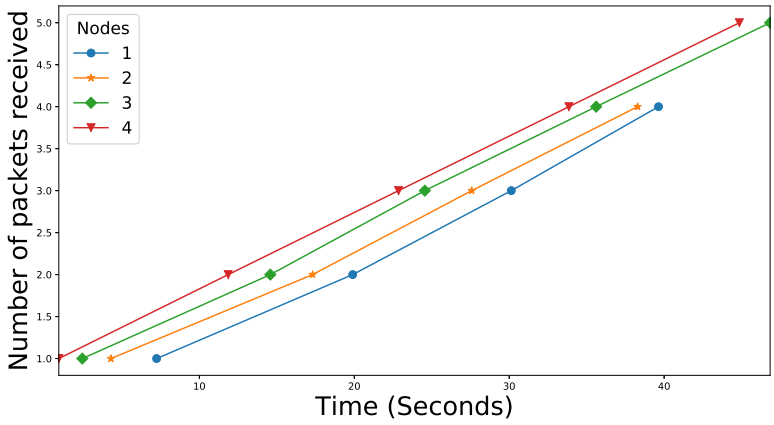

Figure 7: Total packets received without ACK

of the implemented energy model in UnetStack for packet received event.

5.2.3. Depletion of node energy

This subsection demonstrate the successful implementation of primary objective of the proposed work.

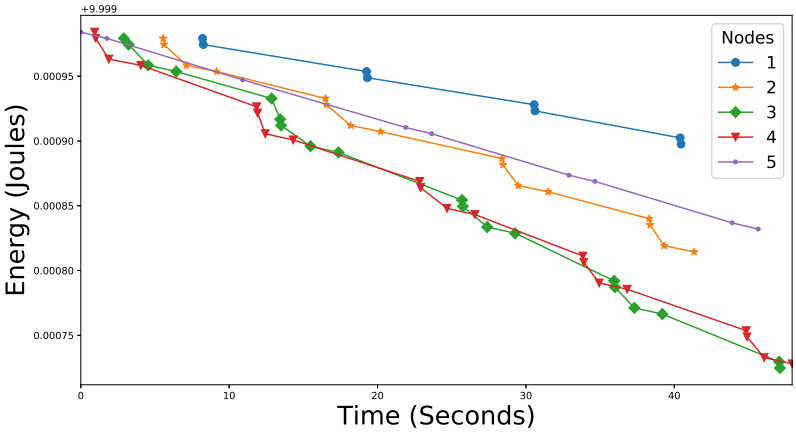

Figure 8: Energy depletion for all active nodes with ACK

The multi-line graphs shown in Fig. 8 and 9 depicts the energy depletion of

all active nodes, for all transmissions and receptions made by the nodes. The graph does not show energy depletion for Node-6 and -7 as these nodes does not forward or receive packets (Fig. 4, 5, 6 and 7) but can sense the transmission (Fig. 3) for which the energy remains same. The markers on the graph indicate a send or receive event occurred at the node.

Fig. 8 shows the graph for depletion of node energy with hop-to-hop ACK. It can be seen in the graph that, Node-1 which is the sink, has lowest energy depleted. As Node-1 is not involved in forwarding packets to next hop, it only receives packets from Node-2 and sends ACK's back to Node-2 for the packets it received. Next the Node-5 being source, has lesser energy depletion, as it only sends packets to Node-4 which is its next hop, and receives ACK's from Node-4. Further the Node-2, -3 and -4 which are the intermediate nodes has higher energy depletion as these nodes receive packets from previous hop, send ACK's back to the previous hop, as well as, forward packets to next hop and receive ACK's from next hop.

As shown in Fig. 8, the number of markers on individual graph of a node is the sum of number of marker on the respective node graph in Fig. 4 and 6. Also, the depletion of energy for packet transmission = packet forward > ACK transmission > packet reception > ACK reception can be observed in Fig. 8.

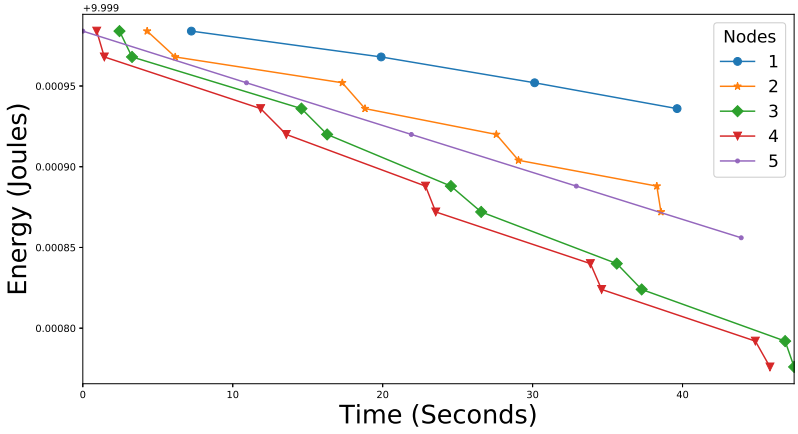

Figure 9: Energy depletion for all active nodes without ACK

The graph in Fig. 9 shows depletion of node energy without ACK. It can be seen, similar to graph in Fig. 8, the Node-1 has lowest energy depletion, as it is involved only in receiving packets. Node-5 energy is depleted, for sending its own packets. Other nodes have their energy lost for receiving packets from previous hop and for forwarding packets to their next hop.

Similar to Fig. 8, the sum of count of markers can be verified between Fig. 9, and Fig. 5 & 7.

The numerical values shown in Fig. 8 and 9 justify the proper functioning of the implemented energy model in UnetStack for energy depletion during packet transmission, reception, forwarding, and ACK transmission and reception.

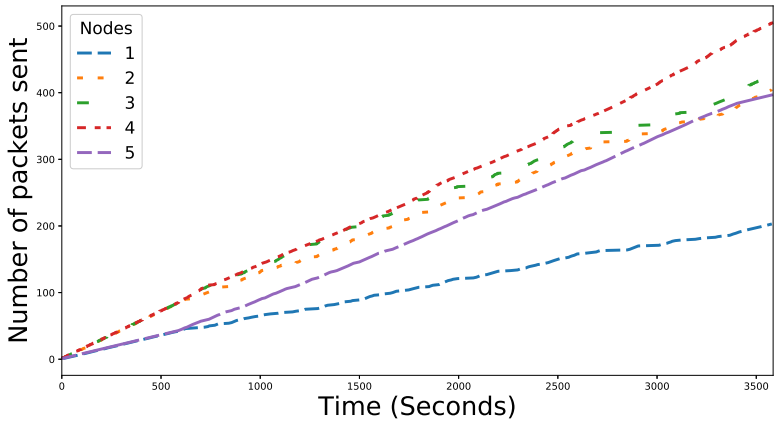

Figure 10: Total packets sent/forwarded with ACK

5.3. Case-II: Extended simulation and analysis

This subsection presents the extensive simulation of the topology as shown in Fig. 3 for the duration of 1 hr., with and without ACK. Further, Node-5 and Node-1 are configured as a source and sink respectively. In comparison with Case-I, the initial energy of the nodes are configured to 0.5J in Case-II.

5.3.1. Total packets sent/forwarded

The multi-line graph in Fig. 10 depicts the total number of packets sent by all the nodes with hop-to-hop ACK. As Node-1 is sink, it sends only ACK's for the packets it received from Node-2. Packets sent from Node-4, -3 and -2 include packets forwarded to their next hop, and ACK's sent to their previous hop.

As shown in Fig. 10, Node-1 has lowest number of packets sent, as Node-1 has sent only ACK's, and it depends on number of packets it receives from Node-2. Node-5 has higher number of packets sent than Node-1, and lesser than other nodes. As Node-5 has to send only packets of its own, or re-transmit some packets when ACK is not received from Node-4. For the intermediate nodes (Node-2, -3 and -4) though they have same task of forwarding packets to their next hop and sending ACK to their previous hop. The number of packets sent is not same, as it depends on number of packets they forward or re-transmit, and number of ACK they send, which again depends on number of packets they receive. In this case, in Fig 10 Node-4 has highest number of packets sent.

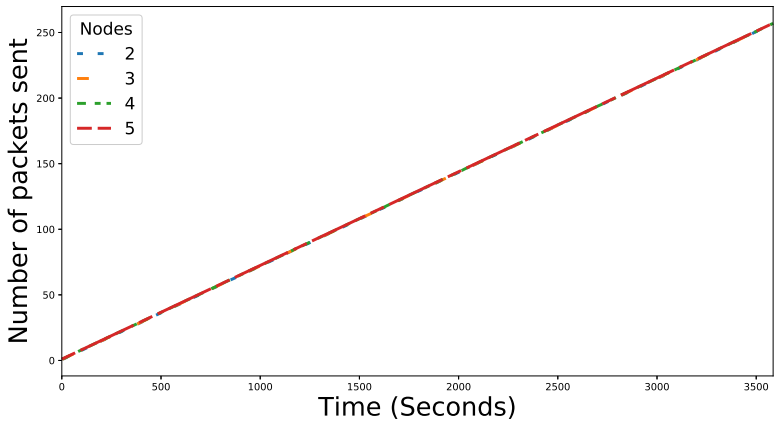

Figure 11: Total packets sent/forwarded without ACK

Fig. 11 depicts the total number of packets sent by all the nodes without ACK. Here except Node-1, the Node-5 which is source, is involved in sending

packets and the intermediate nodes (Node-4, -3 and -2) are involved in forwarding packets. As all the packets sent from Node-5 are forwarded across each intermediate nodes, the amount of energy depletion is same for all these nodes.

As shown in Fig. 11, The Node-1, which is sink node does not send any packets. The number of packets forwarded by intermediate nodes (Node-4,-3 and -2) is same as packets sent from Node-5. Hence in Fig 11 the graph is overlapped for these nodes.

5.3.2. Total packets received

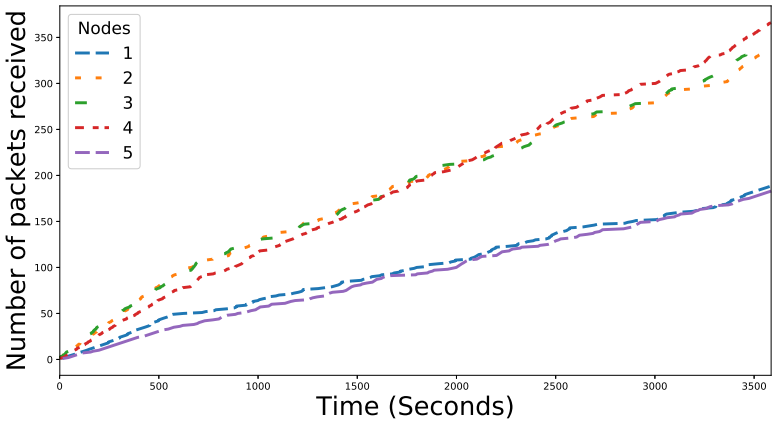

Figure 12: Total packets received with ACK

The multi-line graph in Fig. 12 depicts the total number of packets received by all the nodes with hop-to-hop ACK. As the Node-1 is sink, it receives only data packets forwarded from Node-2. Node-5 which is the source, receives only ACKs sent from Node-4. Remaining nodes receive packets forwarded from previous hop and ACK's sent from next hop.

As shown in Fig. 12 Node-1 and Node-5 has similar number of packets and ACK's received. For the intermediate nodes (Node-4, -3 and -2) the number of packets received depends on number of packets forwarded to them, packets processed at the node,and also re-transmitted packets. In this case, in Fig 12 Node-4 has highest number of packets received, then comes Node-3 and -2.

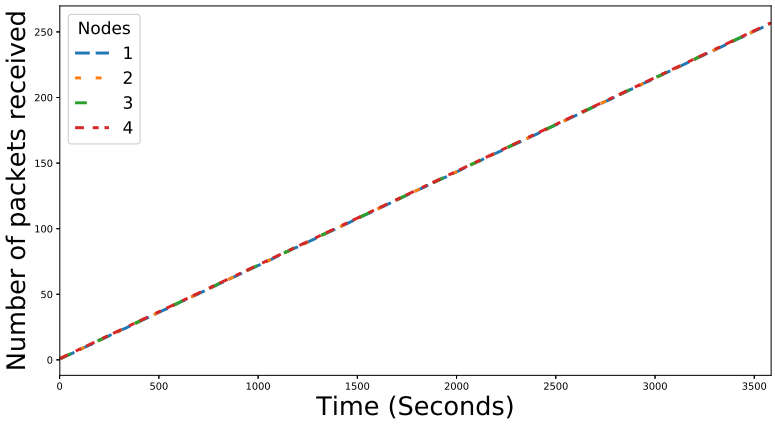

Figure 13: Total packets received without ACK

Fig. 13 depicts the total number of packets received by all the nodes without hop-to-hop ACK. Here except Node-5, The sink (Node-1) and the intermediate nodes (Node-4, -3 and -2) are involved in receiving packets. As all the packets sent from Node-5 are forwarded and received at these nodes, the energy depletion at Node-1 and the intermediate nodes is same.

As shown in Fig. 13 the Node-5 which is source, has no packets received. The intermediate nodes (Node-4,-3 and -2) and Node-1 receive equal number of packets forwarded from their respective previous hop. Hence in Fig. 13 an overlapping can be seen for the number of packets received at these nodes.

5.3.3. Depletion of node energy

The multi-line graphs in Fig. 14 and 15 depicts depletion of energy overtime for all active nodes. Similar to the graph shown in Fig. 8, here, the Fig. 14 shows the graph for depletion of node energy with hop-to-hop ACK. The Node-1 which is sink, has lowest energy depleted. As Node-1 is not involved in forwarding packets to next hop, it only receives packets forwarded by Node-2, and sends ACK's back to Node-2. Node-4 and -3 has higher depletion as these nodes have most number of send and receive events. Further, Node-5 which is source, has depletion higher than Node-2 for sending and re-transmitting

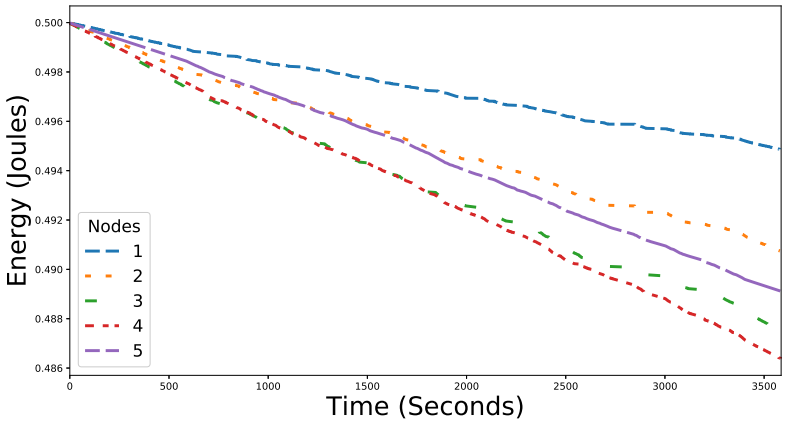

Figure 14: Energy depletion for all active nodes with ACK

packets and receiving ACK's.

The graphs in Fig. 10 and 12 can be used in verifying the energy depletion shown in Fig. 14 for energy depletion during packets sent, forwarded, re-transmitted, ACK's sent and packets, ACK's received.

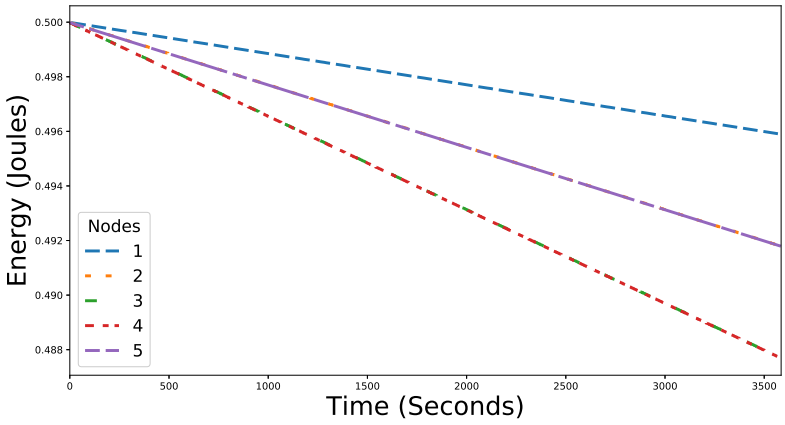

Figure 15: Energy depletion for all active nodes without ACK

Fig. 15 shows graph for depletion of node energy without ACK. Here, as Node-1 is sink, it only receives packets forwarded by Node-2. Thus has the lowest

energy depletion. The intermediate nodes (Node-4, -3 and -2) are overlapped as the amount of energy depletion across these nodes are same. As these nodes are involved in both sending and receiving packets, and also all the packets were forwarded without packet drop due to collision. Node-5 which is the source, involved only in sending packets to Node-4, which is its next hop. Since energy consumed for a transmission is always more than for a reception, it has higher depletion than Node-1 and lesser than other intermediate nodes.

The graphs in Fig. 11 and 13 can be used in verifying the energy depletion shown in Fig. 15 for energy depletion during packets sent, forwarded and received.

The values obtained for energy depletion in Fig. 14 and Fig. 15 justify the proper functioning of the implemented energy model in UnetStack for an exhaustive simulation.

6. Conclusions and future work

UnetStack being one of the most popularly used underwater network simulation tool and the non-availability of residual energy model motivated to design and implement the same. The implemented module extended the existing `HalfDuplexModem` implementation of UnetStack for customized residual energy calculation. Through the basic and exhaustive simulations the correctness of the implemented residual energy model is demonstrated for number of packets sent and received, and energy depletion at all nodes. The calculation of energy still can be made much accurate, considering other parameters in underwater network which is future scope of this work.

Acknowledgment

The authors thank the Science and Engineering Research Board (SERB), Govt. of India for providing financial support (ref. no.EEQ/2018/001036). Further, authors are also thankful to Mr. Sourabh M Zambre (student intern) for his valuable contributions.

References

[1] G. Han, L. Liu, N. Bao, J. Jiang, W. Zhang, J. J. Rodrigues, AREP: An asymmetric link-based reverse routing protocol for underwater acoustic sensor networks, Journal of Network and Computer Applications 92 (2017) 51–58.

[2] K. M. Awan, P. A. Shah, K. Iqbal, S. Gillani, W. Ahmad, Y. Nam, Underwater wireless sensor networks: A review of recent issues and challenges, Wireless Communications and Mobile Computing 2019.

[3] M. Jouhari, K. Ibrahimi, H. Tembine, J. Ben-Othman, Underwater wireless sensor networks: A survey on enabling technologies, localization protocols, and internet of underwater things, IEEE Access 7 (2019) 96879–96899.

[4] H. Luo, K. Wu, R. Ruby, F. Hong, Z. Guo, L. M. Ni, Simulation and experimentation platforms for underwater acoustic sensor networks: Advancements and challenges, ACM Computing Surveys (CSUR) 50 (2) (2017) 28.

[5] A. Sehgal, Analysis & simulation of the deep sea acoustic channel for sensor networks.

[6] A. P. Das, S. M. Thampi, Simulation tools for underwater sensor networks: a survey, Simulation 8 (4).

[7] J. Fall, The ns manual, http://www. isi. edu/nsnam/ns/ns-documentation.

[8] M. Chitre, R. Bhatnagar, W.-S. Soh, Unetstack: An agent-based software stack and simulator for underwater networks, in: 2014 Oceans-St. John's, IEEE, 2014, pp. 1–10.

[9] M. C. Domingo, R. Prior, Energy analysis of routing protocols for underwater wireless sensor networks, Computer communications 31 (6) (2008) 1227–1238.

[10] G. Xing, Y. Chen, L. He, W. Su, R. Hou, W. Li, C. Zhang, X. Chen, Energy consumption in relay underwater acoustic sensor networks for ndn, IEEE Access 7 (2019) 42694–42702.

[11] J. Yan, X. Yang, X. Luo, C. Chen, Energy-efficient data collection over auv-assisted underwater acoustic sensor network, IEEE Systems Journal 12 (4) (2018) 3519–3530.

[12] A. Khan, I. Ahmedy, M. Anisi, N. Javaid, I. Ali, N. Khan, M. Alsaqer, H. Mahmood, A localization-free interference and energy holes minimization routing for underwater wireless sensor networks, Sensors 18 (1) (2018) 165.

[13] P. Xie, Z. Zhou, Z. Peng, H. Yan, T. Hu, J.-H. Cui, Z. Shi, Y. Fei, S. Zhou, Aqua-sim: An ns-2 based simulator for underwater sensor networks, in: OCEANS 2009, IEEE, 2009, pp. 1–7.

[14] R. Masiero, S. Azad, F. Favaro, M. Petrani, G. Toso, F. Guerra, P. Casari, M. Zorzi, Desert underwater: an ns-miracle-based framework to design, simulate, emulate and realize test-beds for underwater network protocols, in: 2012 Oceans-Yeosu, IEEE, 2012, pp. 1–10.

[15] C. Petrioli, R. Petroccia, D. Spaccini, Sunset version 2.0: Enhanced framework for simulation, emulation and real-life testing of underwater wireless sensor networks, in: Proceedings of the Eighth ACM International Conference on Underwater Networks and Systems, ACM, 2013, p. 43.