Hindawi Publishing Corporation
International Journal of Distributed Sensor Networks
Volume 2014, Article ID 950973, 18 pages
http://dx.doi.org/10.1155/2014/950973

Research Article

End-to-End Message Exchange in a Deployable Marine Environment Hierarchical Wireless Sensor Network

César Ortega-Corral,[1,2] Luis E. Palafox,[1] J. Antonio García-Macías,[3] Jaime Sánchez-García,[4] and Leocundo Aguilar[1]

[1] *Facultad de Ciencias Químicas e Ingeniería, Universidad Autónoma de Baja California, Calzada Tecnológico 14418, Mesa de Otay, 22390 Tijuana, BC, Mexico*
[2] *Tecnologías de la Información y Comunicación, Universidad Tecnológica de Tijuana Km. 10 Carretera Libre Tijuana-Tecate, Fraccionamiento El Refugio, Quintas Campestre, 22650 Tijuana, BC, Mexico*
[3] *Departamento de Electrónica y Telecomunicaciones, Centro de Investigación Científica y de Educación Superior de Ensenada, Carretera Ensenada–Tijuana No. 3918, Zona Playitas, 22860 Ensenada, BC, Mexico*
[4] *Departamento de Ciencias de la Computación, Centro de Investigación Científica y de Educación Superior de Ensenada, Carretera Ensenada–Tijuana No. 3918, Zona Playitas, 22860 Ensenada, BC, Mexico*

Correspondence should be addressed to César Ortega-Corral; cesar.ortega@uabc.edu.mx

Received 21 March 2013; Revised 7 October 2013; Accepted 7 October 2013; Published 27 January 2014

Academic Editor: Tai-hoon Kim

We present a pragmatic view of different approaches used to guarantee data delivery in a deployable marine habitat monitoring system, composed of a two-tier dual frequency (2.4 GHz/900 MHz) hierarchical wireless sensor network (WSN). We cover end-to-end application layer aspects. At the lower tier, we preconfigured endpoint (EP) transceivers for automatic data acquisition and wireless transfer using their native Application Program Interface (API) framework. These endpoints communicate with a more powerful intermediate cluster-head (CLH) system. At the upper tier, we deployed a modified low level 8-bit "Lighter" version of the well-known web application protocol called JavaScript Object Notation (JSON, or in our case LJSON) for back and forth CLH to BS validated message exchange. These LJSON messages are converted by the BS to 16-bit JSON and vice versa, for remote Internet interaction. And finally, the BS software establishes Internet Protocol (IP) client socket connections with a remote custom JSON service, in charge of marine habitat sensor data reception, verification, and nonvolatile database storage.

1. Introduction

During the last decade, wireless sensor networks (WSN) have been at the forefront in many ubiquitous sensor oriented research and application developments. A large amount of WSN publications have appeared [1–3] due in part to the constantly growing application space (smart homes, assisted living, precision agriculture, habitat monitoring, etc.), pointing out that the most successful WSN endeavors have been *application-centric* deployments as discussed in [4–6].

In most monitoring WSN applications, sensor nodes have fixed locations. Another important issue is distance, which influences the decision of what general architecture is more practical for a WSN [7]. Most data routing protocols can be classified according to the general WSN structure as flat, hierarchical, or position-based [8, 9]. A flat structure is usually applied for dense networks where many wireless nodes are close together. On the other hand, for long range sparse WSN applications a hierarchical structure may be more practical than a mesh network. And the third option is the position based architecture, which can be used either way in short range or long range WSN with power budget requirements and extended GPS (global positioning system) hardware, in order to update the nodes position coordinates. With the implicit restriction that it can only be deployed in "open sky" applications, it can get a "lock" on the relative position of line of sight satellites flying overhead.

In the application presented in this paper, we chose a hierarchical WSN architecture. This hierarchy establishes two wireless sensor network layers or tiers. Most or all sensor data

are forwarded towards the BS located at the top of the upper tier, which provides Internet connections for DBS sensor data storage. This is done using a web application protocol implemented for point-to-point long range communications.

From the available web application protocols, the most popular is the HyperText Transfer Protocol (HTTP) that accepts client connections and responds with web pages coded with HyperText Markup Language (HTML) [10]. Another important application message protocol that deals particularly with data structure oriented representations is called eXtensible Markup Language (XML) [11]. It does not use "tags" for visual design as HTML, rather XML uses them for data structure messaging and presentation. Another message transmission protocol is called Simple Object Access Protocol (SOAP), and it is a lightweight XML-based messaging protocol for encoding information in request and response messages. And similar to XML, another more compact data representation protocol called JavaScript Object Notation (JSON) was standardized and it is used in browser-server-browser application layer processing [12]. JSON is now being deployed in a more sophisticated manner through the so-called Remote Procedure Call (RPC) services [13].

Recently, standards such as the 6LowPAN published in [14, 15] and the IPv6 Routing Protocol for Lossy (RPL) networks published in [16] promote the use of Internet enabled embedded systems with constrained resources. Nowadays, evolving WSN have to deal with interacting with some kind of web services and application oriented protocols with even lighter-weight restrictions, such as the ones imposed in the JSON syntax. One recent approach is group-based web services in the so-called Representational State Transfer (REST) architecture as stated in [17]. These types of architectures rely on sets of services that can be shared and reused. In REST a resource is an abstraction handled by the service according to a Universal Resource Identifier (URI). These resources can be represented by any format such as XML or JSON and are processed by an application protocol that works under a client/server request/response paradigm. Although REST is not limited to a specific application protocol, most deployments interact with HTTP servers, which handle resources through their native methods: GET, POST, PUT, DELETE, and so forth. Be it the Internet of Things (IoT) or machine-to-machine (M2M) architectures, REST permits developing applications using web services that enable transparent communication with a REST/HTTP compliant WSN agent. On the other hand, others have proposed alternatives to using HTTP. For example, the Internet Engineering Task Force (IETF) organized the Constrained RESTful Environments (CoRE) Working Group, released the Constrained Application Protocol (CoAP) [18] that has functionalities similar to HTTP but thought out for embedded devices such as wireless sensors and actuators. A major difference between CoAP and HTTP is that it uses User Datagram Protocol (UDP) at the transport layer instead of the reliable Transport Control Protocol (TCP).

In this paper, we propose and describe the deployment of a modified version of the JSON application layer messaging protocol for a hierarchical WSN applied to long range marine habitat monitoring [19]. From a bottom-up point of view, on the lower WSN tier, the first protocols we deal with are the well-known XBee transceiver attention (AT) protocol and the Application Program Interface formats owned by Digi Corporation [20, 21]. These XBees are the core systems of our EP nodes. The other application protocol is the aforementioned modified JSON that we adapted to our needs. It operates at the CLH-BS upper tier; particularly, it is an 8-bit "lighter" JSON (or LJSON) for conveying marine habitat aggregated sensor data and network link quality assessments. In our design, the CLH systems are in charge of bidirectional LJSON messaging for back and forth communications with the BS and database web service. In summary, from end to end, the EP nodes send sensor data API frames to the CLH, which extract pertinent information and validate and aggregate it to CLH outgoing LJSON messages that are transmitted to the overall Internet enabled BS system. The BS as it receives CLH LJSON messages decodes them and extracts certain data (for its operational purposes) and then augments and converts these messages to standard 16-bit JSON. Lastly, the BS software opens an Internet socket connection and sends the application layer message to a remote service that decodes the JSON message, extracts sensor digital data, does final data validation, and stores it in a database system (DBS).

This paper is organized as follows. In Section 1.1 ocean monitoring issues are presented, where an itemized list points out physical layer problems that have to be overcome for a successful deployment in a harsh environment such as the sea. In Section 1.2, a discussion about published marine WSN case studies is presented. It is comprised of short descriptions of their particular application. In Section 2, the proposed hierarchical WSN architecture and every type of device that participates are described. In Section 3, the EP application program interface (API) scheme is explained, for configuring their automatic behavior and remote host interactions. In Section 4, we present the way cluster data gathering and aggregation is done by the CLH systems and how they convey LJSON messages destined for BS translation (at the upper tier level). In Section 5, both sides of the BS operations are presented. This includes the role that our experimental TCP JSON enabled server plays in the final WSN data verification and storage. And in Section 6, prototype testing is presented and weeklong experimental results are discussed. Finally, in Section 7, a qualitative comparison of popular web oriented application layer WSN protocols is presented with the goal of pointing out the benefits of our proposal.

1.1. WSN Ocean Monitoring Deployment Issues. Common variables measured by ocean WSN deployments are temperature, pH, salinity, dissolved oxygen concentration, depth-pressure, turbidity, tide direction, and flow rate. Other systems require measuring the amount and type of plankton, suggesting imaging hardware on observation endpoints to visually inspect marine habitat images of remote water samples [22, 23]. In order to overcome long distances, many sensor network applications use repeaters and/or deploy hierarchical architectures. Up to date WSN projects include hybrid topologies that combine hierarchical architectures with mesh topologies at different levels of the overall WSN

structure. The most widely used radio frequencies for wireless links are within the well-known industrial-scientific-medical bands (or ISM frequencies bands at 400 MHz, 868 MHz—part of Europe—900 MHz, and 2.4 GHz) [24].

Once an oceanographic sensor network is in place, other issues entail a maintenance strategy aimed at minimizing attendant costs while preserving proper functionality. For this, three aspects have to be covered: (1) maintenance of sensorization and communication elements—cables and antennas—to prevent deterioration due to the harsh environment—algae fouling, calibration, orientation, and so forth, (2) power supply maintenance, solar panel antifouling by seabirds, and (3) network infrastructure and topology maintenance.

Grouping implementation issues, WSN sea deployments, are determined by

(i) monitored area dimensions and number of deployable nodes,

(ii) network architecture and topology selection according to the prior issue,

(iii) selected radio frequency ranges,

(iv) communications devices and protocols used,

(v) maintenance precautions and power source procurement,

(vi) flotation and/or mooring systems used for ocean surface positioning,

(vii) types of oceanographic sensors or data loggers used.

Some common issues to all WSN implementations such as topology and type of architecture are present in the previous list. Regarding frequency range, for long distance communications, wave length is determinant, and it can be shown that 900 MHz signals travel farther than 2.4 GHz frequency signals with the same level of transmission power, specially over seawater due to additional 2.4 GHz energy absorption loss [25]. One important aspect to consider in marine telemetry systems is the maintenance strategy and the required physical infrastructure. For deep sea monitoring devices, floating devices have to be deployed. Other scenarios include anchor mechanisms as part of the underwater sensor system or when a buoy is set adrift with a counter weight in order to keep it from flipping over.

1.2. Related Work: Marine and Coastal WSN Case Studies. WSN applications at sea are especially difficult to deploy because of the obvious harsh conditions. Some efforts have been done to deploy seaworthy WSN near the coast. One such endeavor was called Self-Organizing Collegiate Sensor (SECOAS) project. It originally deployed sensor networks that monitored low depth undersea sedimentation processes at the base of wind farm towers at eight different low depth locations within an area known as Scroby Sands at the coast of Norfolk, England [26]. This deployment was done to prove that the wireless sensing nodes (called Pods) could be placed and enabled to operate for extended periods of time. From 2003 to 2005, three trials were done, with the last test run lasting two months.

Another deployment, called the OceanSense project, tested an experimental offshore sensor network, at the coast of the China Sea [27]. Initially, eighteen TelosB motes were encapsulated within buoys fitted with steel rods and underwater counter weights to stabilize flotation. For user interaction, a Web based system was implemented where the gateway node connected to a workstation that stored incoming sensor information using a database and processing system called OsnWeb [28]. Another development effort under the name of Environmental Measuring and Analysis Technologies Project (SEMAT) was done with the collaboration of several universities and government agencies in Australia [29]. In the overall SEMAT implementation, during software development, sensor diversity and management became central issues that were dealt with by a proposed Sensor Abstraction Layer (SAL). Every sensor station incorporated a plug-in-based SAL model where support for new types of sensors was loaded on the running system via plug-in software modules. And yet, another project intended for permanent operation, called the Great Barrier Reef Ocean Observing System (GBROOS), was deployed and is still running at the northeastern costs of Australia. GBROOS is now used to study ocean currents and their impact on the Coral Sea habitat, particularly cool and warm water intrusions on the Great Barrier Reef (GBR) [30]. GBROOS uses a two-tier approach: at the lower tier short range oceanographic buoys acquire sensor data and send it to upper tier systems installed on poles anchored on the coral reefs. The overall result is that GBROOS is now a heterogeneous marine sensor network deployment, with ongoing technology development, looking to substitute legacy instrumentation with newer less expensive WSN technology made for marine deployments.

2. Dual Frequency Hierarchical WSN for Marine Habitat Monitoring

The obvious obstacle in long-range WSN applications is distance. In this case, hierarchical WSN topologies are better suited than modern mesh topologies because they need a lower number of hops in order to complete wireless communications [31–33], and this is why we selected a two-tier hierarchical wireless sensor network architecture for our marine habitat monitoring system.

The actual site where our final system will be placed is within Bahía Falsa at San Quintín, Mexico, located at the western shore of the upper half of the Baja California Peninsula, a satellite image of the site is shown in Figure 1. Bahía Falsa is a shallow body of water that is more like a lagoon than a bay. There is a thriving Oyster Farm industry developing within Bahía Falsa's western side. The oyster farmer community have special interest in deploying this monitoring system because it will give them updated information on the bays habitat conditions, so as to make informed decisions on how to manage their farms and how to anticipate unhealthy conditions for the oyster species, and this can be indirectly determined knowing habitat variables such as temperature, pH, and salinity [34, 35].

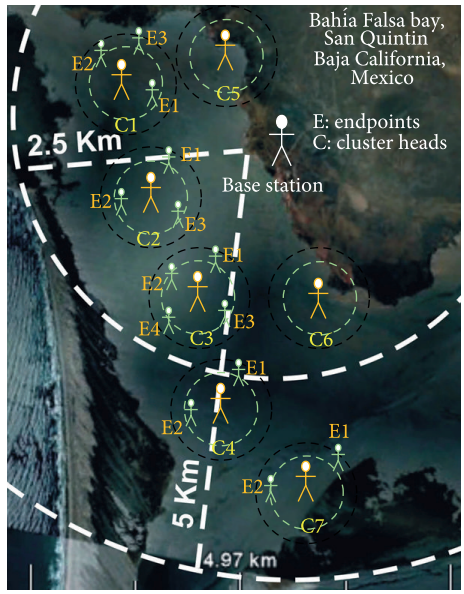

Figure 1: WSN deployment site at Bahía Falsa, Mexico. EP agents are low power endpoints and CH agents are dual frequency cluster heads.

Several potential WSN clusters are shown in Figure 1, and at the center of them are CLH and EP monitoring agents. The distance to the nearest seashore facility, where the gateway operates, exceeds a kilometer anticipating significant signal loss. Considering existing technologies and radio-frequency (RF) standards, two unlicensed frequency bands at the 900 MHz and the 2.4 GHz were considered and later used. In this case, the 900 MHz range bands have a more convenient wavelength for long range communications, reaching greater distance with the same power compared to 2.4 GHz links [36]. However, at 2.4 GHz more short range nodes can coexist due to broader bandwidths and this is why we are using this frequency range for cluster endpoint communications.

A simplified structure of the selected architecture is shown in Figure 2. The WSN hierarchy is composed of two tiers (or levels). At the lower tier, low resource 2.4 GHz EP nodes make up clusters coordinated by a single CLH. These CLH nodes, with more computing resources and power, have two radios onboard: a 2.4 GHz transceiver that enables cluster communications and a second 900 MHz radio through which the upper tier network is created for relaying remote sensor data towards the overall WSN base station.

Most of the habitat sensing is performed at the lower tier or cluster level, and this includes sensing at the EP level as well as at each CLH system. Here we take advantage of "off the shelf" EP transceiver technology called XBee radios that can be configured on the fly with extended data acquisition capabilities and sleep options when being inactive. These XBees operate using a variation of the IEEE802.15.4 medium access control (MAC) wireless sensor protocol operating within 2.4 GHz bands [37]. For sensing purposes, almost all types of XBees have 10-bit analog to digital (A/D) converters onboard and can be configured for automatic acquisition and data transmission (Tx). The XBee protocol stack is shown in Figure 3(a), and in Figure 3(b) our CLH protocol stack is illustrated, noting two radio interfaces onboard every CLH.

Using the drone-type XBee EP sensor node approach (or any other similar transceiver) has the aim of being able to populate a small area of interest with less expensive wireless equipment creating a cluster, which is a coordinated CLH node that has extended resources to fulfill its purpose. On the other hand, the hierarchical approach has two drawbacks: (1) it might imply using more transmission power to cover longer distances and (2) the intermediate CLH nodes have to be fault-tolerant to maintain the system running. In contrast, a benefit of the hierarchical approach is that it requires fewer radio links, which avoids cluttering the radio spectrum with multiple simultaneous wireless communication links interfering with each other.

Although in a hierarchical point-to-point approach messages have only one route, if a mesh multihop network were to be used instead, message exchange would incur in additional overhead and cross-layer data aggregation at almost every hop. This might make the exchanged messages too long for the mesh middleware to handle. In that situation, the system may not be able to avoid (or at least minimize) message fragmentation, increasing communication algorithm complexity, and processing overhead at different network hops [38, 39]. Similarly, the BS has to maintain much more dense information pertaining to the networks structure and different received signal strength indicator (RSSI) readings, measured along the multihop path. These last remarks justify our hierarchical topology selection for long distance communications. And although routing is not part of this paper, as the WSN structure grows, it impacts the data representation complexity and the whole transfer process, which may compromise its reliability. This reflects the importance of good planning and of deploying appropriate application layer strategies.

In our deployment, in the upper end of the system, the BS protocol stack is composed of two sides, as shown in Figure 4(a). Both upper tier cluster-head communications using LJSON and Internet gateway capabilities using JSON messaging are present at the application layer. In Figure 4(b) our WSN web application database server protocol stack is also shown, and it expects data encapsulated in JSON messages coming from different WSN agents.

In Figure 5, fundamental use cases are shown for the agents involved in our HWSN model, where application level and cross-layer operations are listed.

There are three actors or agents involved: the EP, CLH, and BS. The CLH agent interacts at both lower and higher tier. Interaction between agents depends on the hierarchical nature of this network. Some actions done in the lower tier are not done in the upper level, although a CLH at the upper tier might control lower tier agent processes. One such task is the sleep/wakeup scheduling that takes place at the EP level, pointing out that CLHs do not sleep but they do manage the sleeping process. All automatic sampling sends data to the BS for LJSON/JSON message translation, transference through the gateway, and final custom web service database storage. The conceptual example of automatic sampling and data transfer is shown in Figure 6. The EP uses an API frame

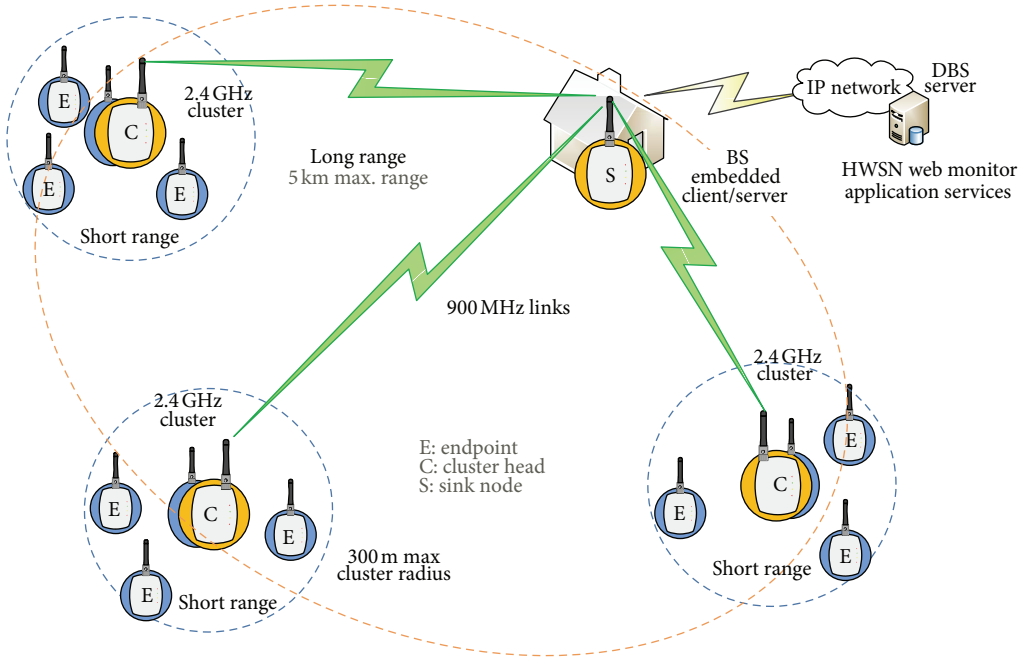

Figure 2: Hierarchical WSN topology for long range marine habitat monitoring.

XBee EP	Cluster head	
API	EP API/LJSON translator	
ZigBee (optional)	ZigBee (optional)	CLH routing
802.15.4 MAC	802.15.4 MAC	WSN MAC
802.15.4 PHY	802.15.4 PHY	WSN PHY

(a) (b)

Figure 3: Hierarchical WSN device protocol stacks: (a) endpoint and (b) cluster head.

HWSN base station		WSN web service
LJSON/JSON translator		JSON/PHP
CLH routing	TCP	TCP
	IP	IP
WSN MAC	Ethernet MAC	MAC
WSN PHY	Ethernet PHY	PHY

(a) (b)

Figure 4: Hierarchical WSN protocol stacks: (a) base station/gateway and (b) JSON enabled web service for WSN data storage.

to send its data to the CLH, which extracts sensor data and constructs the pertinent LJSON message that is translated to JSON and finally received by the web server.

Because the system runs under a modified JSON messaging scheme, the CLH can be expanded to interact with remote web hosts through custom BS services. The BS runs as a proxy that translates messages from LJSON to JSON and vice versa. But most of the sensor data preprocessing is done by the distributed CLH systems in charge of their cluster EP operations. In Figure 7, a web host messaging flow example is shown where a CLH is queried about its status; every exchange ends with an acknowledgement.

3. Endpoint Operation and Cross-Layer Protocols

An XBee transceiver's configuration depends on the values that their internal nonvolatile registers hold, which determine its behavior and indicate their status [40]. These registers are accessed and modified through serial communication interaction with a host computer or embedded system. There are two ways of reading and writing the XBee registers: (1) with AT command strings or (2) by sending it formatted Application Program Interface (API) frames. The problem with issuing serial AT commands is that there is a delayed response that might take a few seconds, because the AT protocol was originally intended for slow human interaction.

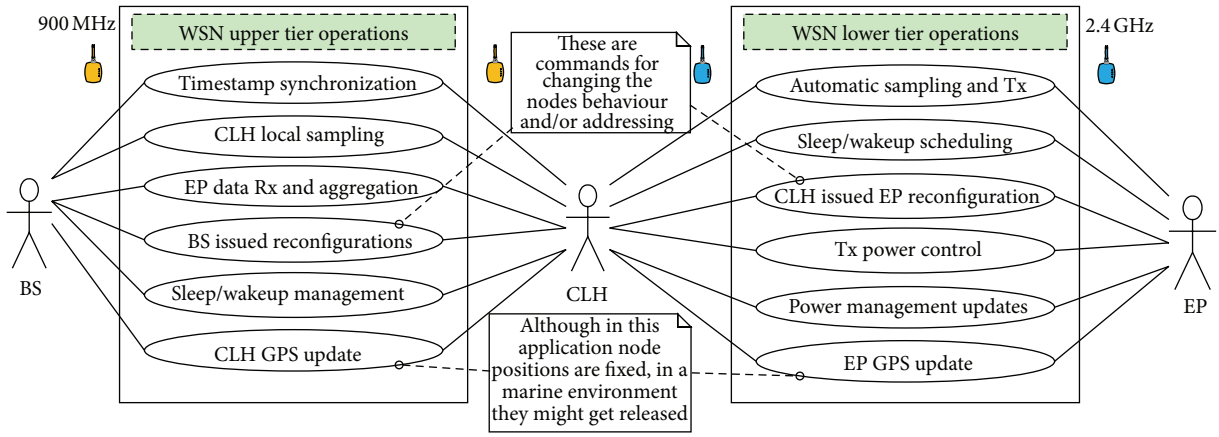

Figure 5: Hierarchical WSN agent uses cases.

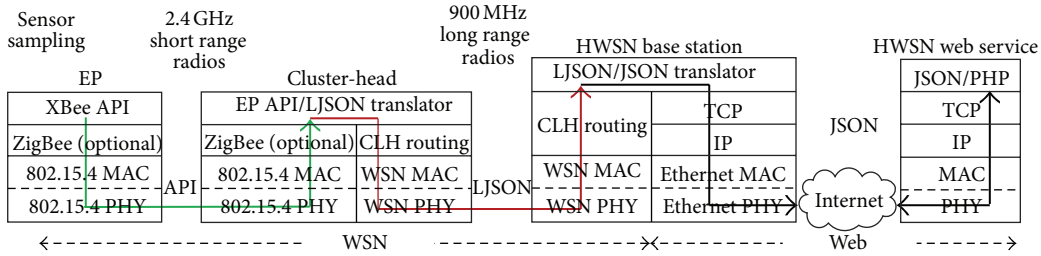

Figure 6: Sensor data traversing the different WSN agents layers.

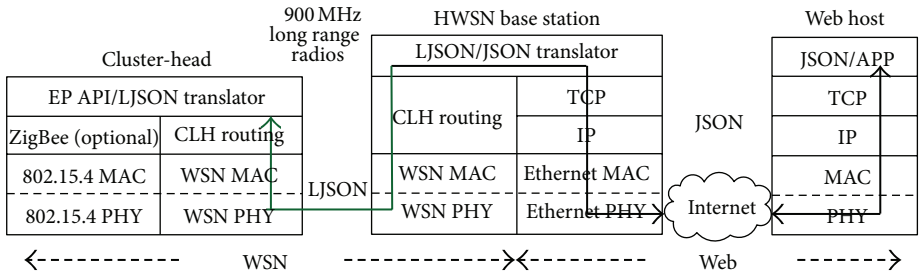

Figure 7: Communications flow while a web host queries a CLH node.

Due to this drawback, the XBee manufacturer programmed an alternative for embedded system interaction through the API framework. This now permits other capabilities such as remote AT configurations encapsulated within API frames that are instantaneously executed by the XBee. The generic XBee API frame structure is shown in Figure 8 [41]. Also, three other particular API command frames pertaining to this application are shown in Figures 9, 10, and 11.

In Figure 9, the specific API frame corresponds to a remote XBee I/O automatic response identified by the 0x83 API identifier (cmdID), which in our case is an XBee 16 bit addressed input signal frame sent automatically by an EP and received by a destination CLH node. A 0x83 frame payload may hold A/D samples coming from as much as six analog inputs of the remote source EP, among other input detecting features. On the other hand, in Figures 10 and 11, the remote AT command request frame structure (cmdID = 0x17)

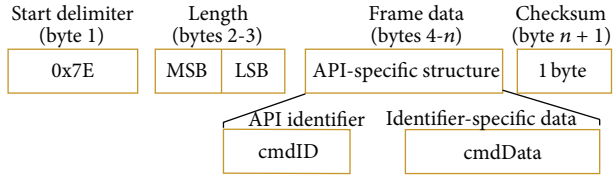

Figure 8: XBee API frame model.

and its corresponding remote AT command response frame (cmdID = 0x97) are shown. The response frame confirms the correct configurations and registers updates, or it flags if an error occurred during this operation due to malformed request frame.

In Figure 12, remote XBee EP interaction with the XBee-CLH system is shown. The CLH has its own XBee onboard

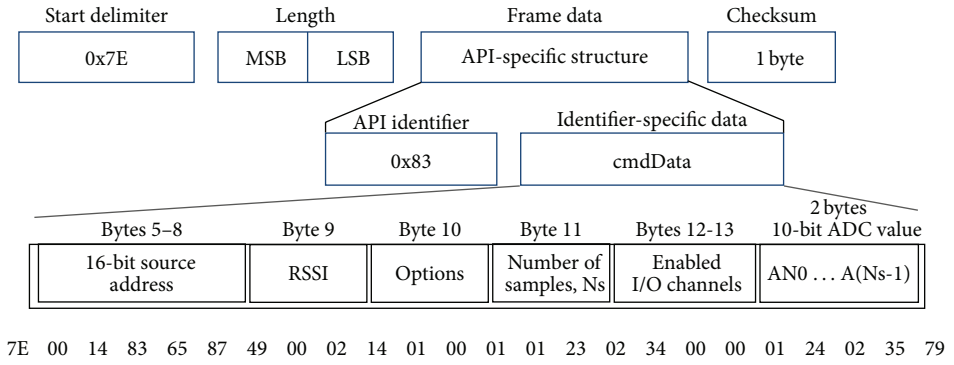

Figure 9: XBee API automatic ADC sampling frame model.

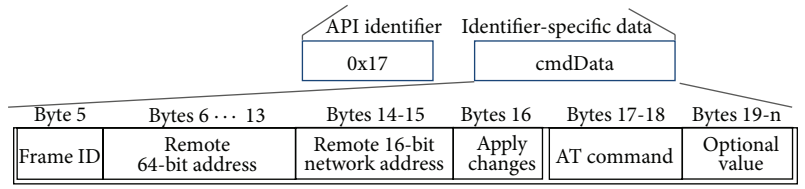

Figure 10: Remote AT command Request.

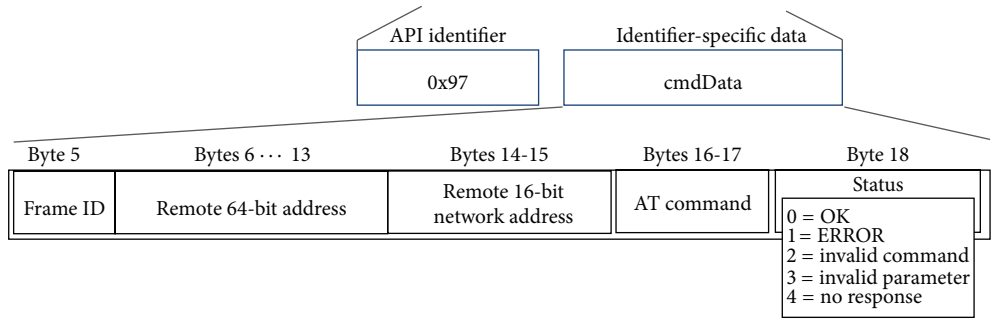

Figure 11: Remote AT command response.

which accepts API requests and responds to them as well. The example shows the three prior API frame descriptions: 0x83 automatic remote I/O frame, 0x17 remote AT command request, and the 0x97 remote AT command response interactions. In this application, the 0x17 frames mostly request remote sampling rate changes and enable/disable EP sleep scheduling configurations. Other remote AT requests pertain to Tx power level reassignment to avoid wasting battery power during data transmissions, and this becomes necessary when the incoming wireless received signal strength indicator, or RSSI, is exceedingly strong.

4. Cluster-Head Data Gathering: Aggregated Messages and "Lighter" JSON

CLH devices have more resources compared to EP systems. In this implementation, the main CLH controller module is the Arduino-Mega controller [42]. We fitted the CLH with a custom stack shield, such as the one shown in Figure 13, which accommodates two radios on board: an XBee Pro transceiver and an AC4490LR1000 long range digital radio, and a second shield on top for sensor conditioning and data acquisition. In general, the CLH system is in charge of five main tasks: (1) runs and has access to an updated real-time clock for time-stamping and event logging; (2) at the 2.4 GHz lower layer the CLH receives and processes remote sensor data API frames coming from the EP network; (3) also the CLH acquires, validates, and buffers local sensor data that is considered part of the lower tier; (4) the CLH gathers and time-stamps other relevant data as well, like power management values, cross-layer information, and RSSI statistics; lastly (5) the CLH constructs an application layer message encapsulating all aggregated data in application layer messages that are sent through its serial interface, which is connected to the upper tier AC4490 900 MHz wireless link radiating towards a long range sink node attached to the BS.

A very important task, often overlooked in theoretical WSN discussions, is the need of having an up-to-date time clock for accurate sensor data acquisition timestamping.

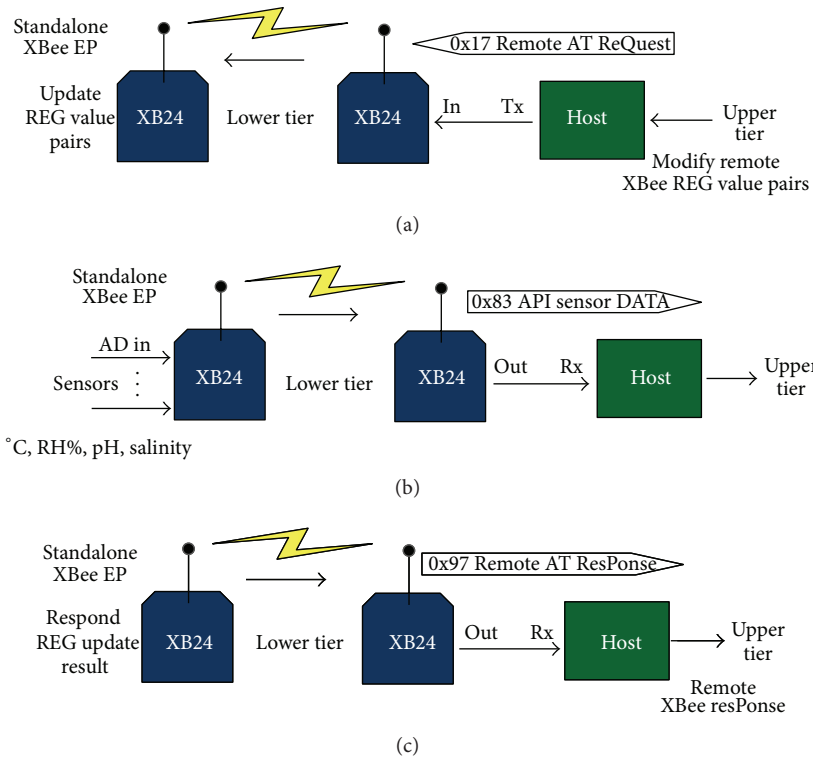

Figure 12: XBee API frame exchange with an attached host.

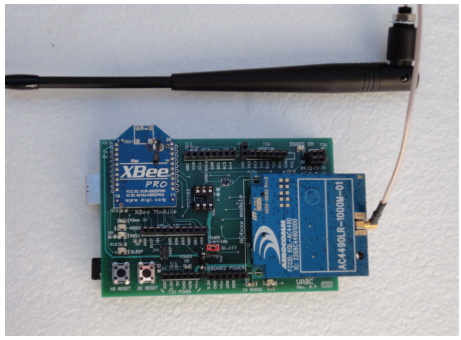

Figure 13: Cluster-head communications shield on top of an ArduinoMega.

The sampling timestamp may be generated locally or done at the final server repository. The latter option has the drawback that a substantial delay may occur, from the time the samples are acquired until they are timestamped and stored in a DBS. In our case, the simplest EP systems lack the resources for having a real-time clock (RTC) or such, its sampling is a matter of a direct A/D conversion process done by the EP firmware, and after a few milliseconds the acquired data is placed on the TX buffer and then sent-off to the associated CLH node. At the destination EP, sensor data is received encapsulated within 0x83 coded API frames. Worth mentioning is that, for practical purposes, we chose to use the UNIX ten digit time-stamp, which is a standard format that represents the number of seconds elapsed since 01/01/1970 (or the first of January of 1970) until the present.

When a CLH node receives remote sample API frames, the CLH software uses an Arduino C++ open source XBee API function library [43] with which it first identifies the 0x83 command ID byte, then inspects the frames content, and extracts two byte sensor data values. At this point, every CLH extraction process involves saving EP data in its associated timestamped buffer for forthcoming aggregation and subsequent message relay towards the BS.

Regarding local CLH sensor sampling, in this deployment the CLH controller has at its disposal several 10-bit A/D input lines. In our case we initially used four of the Arduino-Mega boards analog inputs, so as to acquire air and seawater temperature, air relative humidity, and remaining battery voltage. Other habitat variables being considered are the seawater pH and salinity levels. In most habitat monitoring systems, the sampling period usually is equal to several seconds or minutes. This means that most habitat monitoring applications are not data intensive, and long intervals of inactivity can be used for other tasks or just simply for putting the electronics to sleep, to conserve battery charge in the meantime. All data gathering processes in this project done by the CLH are destined to be encoded into messages that are to be sent to the BS which is discussed next.

4.1. Long Range Upper Tier AC4490 Laird Transceiver. We used the Laird technologies AC4490LR1000 transceiver as the radio that establishes upper tier communications, and it

0x83	Payload data length	Source MAC	Payload data

Figure 14: Laird API Receive packet.

0x81	Payload data length	Res.	Transmit retries	Destination MAC	Payload data

Figure 15: Laird Transmit API packet model.

0x82	Reserved	Last ACK RSSI	Failure/success

Figure 16: Laird API Send Data Complete packet.

0x81	Payload data length	Res.	RSSI	Source MAC	Payload data

Figure 17: Laird Enhanced API Receive packet.

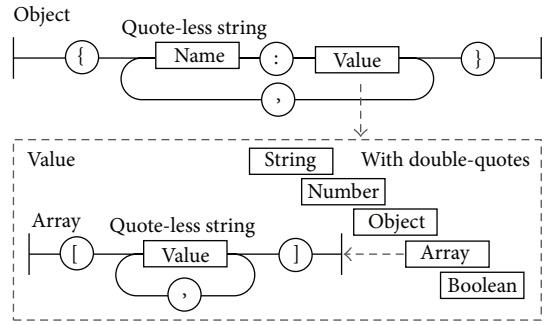

Figure 18: L-JSON object syntax.

operates within 900 MHz ISM frequency bands and transmits at a fixed 76.8 Kbps data rate. It uses Frequency Shift Keying (FSK) combined with Frequency Hoping Spread Spectrum (FHSS) collision avoidance [44]. These modules have adjustable Tx power levels that go from 5 mW to 1000 mW using 3 dBi antennas. When a data frame arrives, the transceiver places the information on its serial output line so an attached host system gets the forthcoming data. The AC4490 transceiver operates in a master-slave configuration, or what the manufacturer calls a server-client scheme with the aim of synchronizing multiple clients' turn to send data through the wireless channel. The server radio is in charge of sending a synchronizing channel beacon. The drawback is that with this scheme the server transceiver has to consume much more power than a client transceiver, which is why it is usually placed as the sink transceiver at the BS where it is more certain that there will be enough available energy.

These transceivers can be configured on the fly using their own version of AT commands, through which a host user can access and change the AC4490 internal EEPROM initialization and operational registers [45]. Upon resetting the transceiver, the new configuration values startup the systems operation. The Laird AC4490 can operate as a transparent line or can be configured for API communication, which is meant for receiving and transmitting data in a more reliable way. This API is based on a framing structure that the host writes or reads from a preconfigured serially attached AC4490. Four kinds of API packet exchanges can be enabled: *Receive*, *Transmit*, *Send Data Complete*, and *Enhanced Receive*. When a host is in this type of communication mode with the AC4490, specific packet structures are expected, and the general API formats are shown in Figures 14, 15, 16, and 17.

In this deployment, the AC4490 radios are configured to operate in Transmit API mode and in Enhance API Receive mode. The Transmit API mode is useful because with it the BS can specify the destination CLH address on-the-fly without writing to the transceivers EEPROM, taking into account that an EEPROM has a limited amount of memory recording cycles, this will avoid shortening its lifespan. Likewise, the Enhanced API Receive feature is convenient because it holds cross-layer information such as the RSSI value and the MAC address identifying the source node.

4.2. LJSON for Reliable Sensor Data Transfer. It is said that JSON is a lightweight web application data protocol, similar in functionality to eXtensible Markup Language (XML) but with a simpler syntax compared to conventional web application protocol notation styles [46]. Noting that instead of using "less than" < or "greater than" > symbols to open and close tags, JSON uses colons ":" to associate name, value pairs, and commas to separate multiple variable assignments. For grouping array members JSON uses brackets and for object representations it uses braces for grouping properties and methods.

Taking syntax rules from the standard 16-bit JSON protocol, here we propose a "Lighter" JSON or LJSON as an 8-bit JSON type scheme for representing complicated data structures and procedures within ASCII messages, while being easily read by humans. All JSON names of objects, arrays, or variables are strings and are invariably enclosed between double quote symbols " ". In our lighter implementation of JSON we chose not to use double quotes to achieve shorter messages and this is why we also call LJSON a "quote-less" JSON. Thus, LJSON may represent groups of variables, arrays, and objects with an even simpler syntax. In Figure 18 the syntactical JSON rules are shown and in general apply to 8-bit LJSON with double quote suppression for name casting, and the exception is for implicit string values where double quotes are permitted to avoid confusion when parsing comes around.

LJSON conveys hierarchical sensor network information which may become highly structured. In general, all WSN L-JSON messages are represented as objects that hold variables/properties and arrays that group sensor samples taken at different locations within the network. This modified "Lighter" JSON only specifies syntax rules, and it does not explain the way to go about coding or decoding messages. First off, we are calling "tokens" all JSON standard characters such as the following `{ : " , [ ] }`. Also, another couple of tokens are considered, and these are the single quote ' for character representations and the question mark ? for

Table 1: Simplified general LJSON names for a hierarchical WSN.

Symbol	Name	Description
T	Timestamp	UNIX timestamp
I	Identification	Object or array particular identification number
Av	Sample array	Set of analog samples taken by either the CLH or EP
An	Number of samples	Amount of samples that holds the associated sample value array
C, c	Cluster-head object	Specifies a CLH object and holds pertinent member variables, arrays, or other objects, such as EP objects
E, e	Endpoint object	Specifies an EP object and holds pertinent member variables and arrays, such as sample arrays
En	Number of EP	It is specified within a CLH object to know the amount of objects that hold the EP array of objects
R	XBee RSSI	RSSI that reads a CLH when receiving EP data
R	AC4490 RSSI	The AC4490 RSSI level that reads a CLH when it receives a BS messages
M	AC4490 MAC address	Used for creating and/or maintaining routing tables
A	XBee 64 bit address	Used in some remote API configuration commands
A	AC4490 transceiver object	Groups together properties and methods associated with an AC4490 radio
X	XBee transceiver object	Groups together properties and methods associated with an XBee radio
P	AC4490 Tx power level	Long range radio transmission power level for optimizing power consumption
Rg	AC4490 EEPROM register address	AC4490 configuration register address
Rv	AC4490 EEPROM register value	AC4490 configuration register address
P	XBee Tx power level	XBee EP radio transmission power level
Rg	XBee register name	Holds an XBee register identifier to query or update
Rv	XBee register value	Holds an XBee register query response or updated value

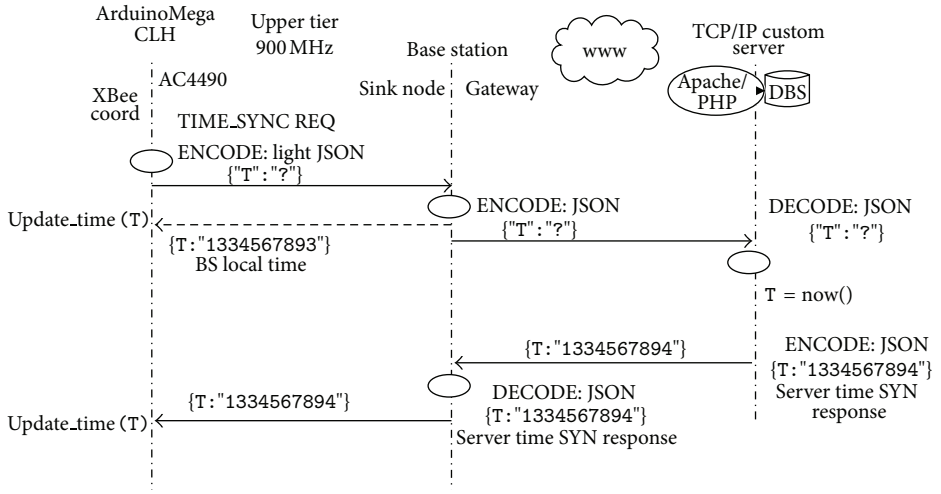

Figure 19: Endpoint time synchronization sequence diagram.

straightforward queries such as Time Synchronization update requests. Furthermore, we decided to specify global simplified names for common object, array, and variable names, and these are listed in Table 1. An example of the simplest LJSON deployment is when a CLH sends towards the BS a time synchronization request message, which is needed to assure updated timestamps [32]. A common sense LJSON message for this request is {t :?} which with ANSI standard JSON would be {"t" : "?"}. In Figure 19, a complete CLH time synchronization request/response sequence diagram is shown.

A standard ANSI character JSON example where its contents are CLH gathered EP sensor samples is shown as follows:

{"C" : 1,"En" : 2,"e" : [{"i" : 3,"T" : "1352244343",
"r" : 68, "An" : 4,"Av" : [235, 9E8, 81C, 430]},
{"i" : 4, "T" : "1352244344", "r" : 72, "An" : 4,
"Av" : [235, 9A6, 82C, 422]}]}.

This message conveys information gathered from two endpoints, and the EP data makes up an array coded within

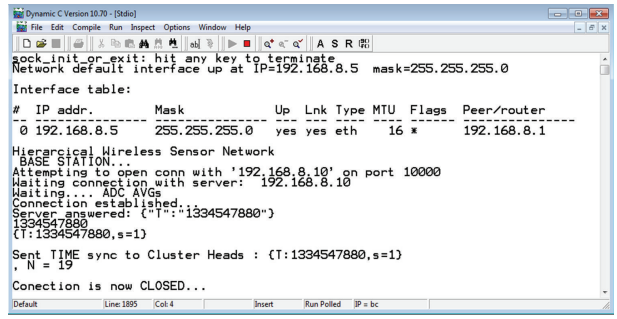

Figure 20: BS start-up processes: network connection and time synchronization.

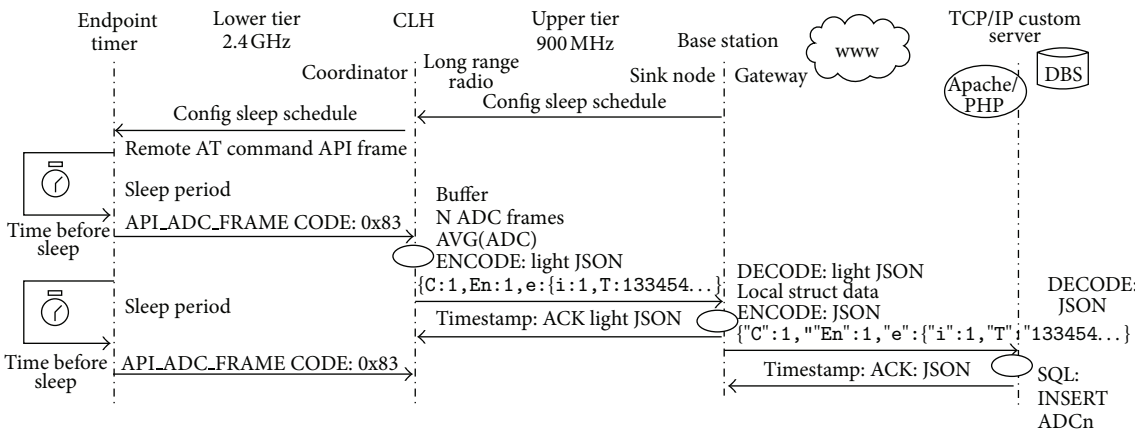

Figure 21: EP cyclic sleep scheduling sequence diagram.

the cluster-head message as "e" : [{"i" : 3, ...}, {"i" : 4, ...}]. Also, this message is timestamped with a ten numbered character string "T" : "1352244343" for EP 3 and for "T" : "1352244344" for EP 4. Furthermore, it also contains an "An" value that indicates the number of analog samples within the "Av" array. The message is 145 ANSI characters long, meaning that its total length in bytes is 290. It is important to note that standard 16-bit JSON does not admit hexadecimal numbers, which means that the latter example does not fully conform to JSON rules due to the presence of hexadecimal numbers within the data array. After applying L-JSON is reduced to

{C : 1, En : 2, e : [{i : 3, T : 1352244343, r : 68, An : 4, Av : [235, 9E8, 81C, 430]}, {i : 4, T : 1352244344, r : 72, An : 4, Av : [235, 9A6, 82C, 422]}]}.

Although this message conveys identical information, its syntax is lighter due to its quoteless nature and because it is made up of 8-bit ASCII characters. Its total length is 134 bytes, which is 46.2% the length in bytes of the previous standard JSON message, with the added advantage that in our L-JSON rules, and hexadecimal notation within ADC arrays is encouraged. If a LJSON message exceeds the intended transceiver maximum payload length, then it will have to undergo fragmentation. This means that a frame ID must be included in the message to be able to reconstruct the entire LJSON string at the BS.

5. Base-Station Operations, Message Decoding, and Internet Gateway Communications

For the base station systems core, we selected the Rabbit Controller Module RCM4300, which has an Ethernet interface onboard and four serial ports [47]. It comes with a complete TCP/IP stack software library, through which the network interface is configured and an IP address is assigned to it so it can communicate with the Internet. Rabbit modules are programmed using their native Dynamic C language, which is a modified C language with augmented capabilities for implementing state machine through their "costate" paradigm [48]. In this project, the RCM can work both as a server (with passive TCP connection) and as a client (through an active TCP connection). As a client, the BS opens a TCP connection only when it is required, such as during time synchronization or when LJSON messages arrive and have to be turned into standard JSON and sent to our custom WSN database web server. In Figure 20, the BS running software debugging window is shown when it starts up.

During initialization, the BS enables its Ethernet connection and establishes its IP address as well. Afterwards, it connects to the TCP server and requests the current Unix Time. After the server correctly responds, the BS broadcasts the time synchronization to all CLH in range, this process corresponds to the one depicted in Figure 18. All BS

```
$socket = socket_create(AF_INET, SOCK_STREAM, 0);
     // bind socket to port
$result = socket_bind($socket,$address, $port);
     // start listening for connections
$result = socket_listen($socket, 3);
echo "Server UP... waiting BS messages";
do{
     // accept incoming connections
     $spawn = socket_accept($socket);

     // read client input
     $input = socket_read($spawn, 2048);
     echo "\n\n".$input."\n\n";

     $msg_obj = json_decode(safeJSON_chars($input),0,2048);

     if  ($msg_obj == NULL)
          $output = "JSON ERROR... null message.\0";
     else
     { // Process $msg_obj
          ...
     }
     socket_write($spawn, $output, strlen ($output));
     //close sockets
     socket_close($spawn);
} while($input! = "quit");
socket_close($socket);
```

Algorithm 1: TCP socket server program with JSON decoding.

tasks are done by the RCM4300 module when it receives LJSON messages coming from the CLHs through one of its serial ports, where an AC4490 transceiver is interfaced. In Figure 21, an end-to-end example is shown, and it represents sleep scheduling and automatic sampling operations with LJSON/JSON decoding/coding and data transfer.

The BS issues a Cyclic Sleep Schedule LJSON message towards all CLH. Afterwards, every CLH generates a Remote AT command API frame and sends it to their corresponding cluster EP members. Part of the configuration includes establishing the appropriate sampling rate, enabling the data acquisition process and automatic wireless transfer.

When a CLH starts receiving sensor data through the EP API frames, it aggregates pertinent data and creates a "light" JSON string with structured information. When the BS receives the LJSON string, it extracts relevant information and stores it in local data structures. Afterwards, the BS increases the string length by adding symbols to comply with the JSON standard casting it as 16-bit character message and opens a TCP socket connecting to the off-site server waiting for such information. The server receives the message, extracts sensor data from the JSON string, and queries the DBS to finally store all relevant information in preformatted data tables for future data mining.

An important aspect is that in practice there is a limit on the length that a wireless message can have. This depends on two things: the transceivers interface buffer size and the limit of the API frames payload length. Usually the input/output buffer is larger than individual frame payload lengths to accommodate several incoming wireless data payloads. The buffer and payload size differences are illustrated in Figure 22.

In our application the particular AC4490 transceiver has an 80-byte API payload length limit and a 256-byte input/output buffer. This means that the incoming buffer accommodates a little over three full API payloads before it fills up, as shown in Figure 15. If more bytes are received and the incoming data is not read immediately, output buffer overflow would occur and data would be lost at the receiving host. In this situation, if LJSON message lengths exceed the allowed Enhanced API Receive payload length, fragmentation will take place and the latter discussion becomes relevant. This implies that at the CLH, proper message segmentation has to be done in a way that after the BS receives these segments the entire message can be reassembled without losing data. It is well known that in traditional and wireless networks many schemes have been devised for this purpose while conveying data through different layers of the network model. In a similar way, we deployed a segment transport control mechanism at the BS to guarantee reliable data transfer, which assures data integrity during the entire process. Nevertheless, it all starts at the CLH level where data aggregation takes place and LJSON message coding is done and where the total LJSON message length determines if segmentation is necessary. And consequently, the CLH has to do an orderly and predefined segmentation that the BS will expect in order to decode incoming messages.

```
if((@$msg_obj -> {“T”})!= NULL)
{
      $T = $msg_obj -> {“T”};
      if ($T == “?”)
      {
            $my_t = @getdate(@date(“U”));
            $output = “{\“T\”:\ “”. $my_t[0]. “\”}\0”;
            echo “TIME Sync >> $my_t[0]\n”;
      }
}else if (($C = $msg_obj -> {“C”})!= NULL)
{
       echo “\nCluster ID: ”. $C. “\n”;
       if (($En = $msg_obj -> {“En”})!= NULL)
            echo “\nNumber EndPoints: ”. $En. “\n”;

       $endpoint = array();
       $endpoint = $msg_obj -> {“e”};
       ...
}
```

Algorithm 2: Server time-update response and CLH message JSON decoding.

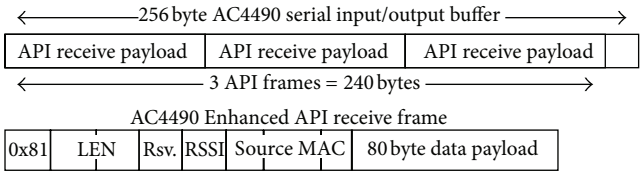

Figure 22: API data payload is placed on the AC4490 serial Tx buffer.

TCP Server Coding and Operation. On the web server side, PHP scripting was done to develop a TCP socket server [49]. Part of the code is shown in Algorithm 1. The server listens on a designated open TCP port, through which it receives the BS's messages that contain relevant sensor endpoint information. After validating and decoding incoming JSON messages, the server extracts the data of interest and stores it on a localhost DBS using structured query language (SQL). The first interaction that the BS has with the server is a Time Synchronization request. Some TCP socket functions that PHP implements are similar to the BSD (Berkley Socket Distribution) functions that originated this standard. At the server side, data coming in through the socket is read and stored in an `$input` variable and then is passed through the PHP JSON decoder function getting a corresponding object message that is stored in the `$msg_obj` object.

For our application, a typical JSON string example is shown as follows:

{"C" : 1, "En" : 1, "e" : [{"i" : 2, "T" : "1334545815", "An" : 3, "Av" : [235, 9E8, 81C]}]}

Another JSON string is the Time Sync string which starts with a quoted letter T. The code snippet of Algorithm 2 shows the way an object member can be extracted, be it a CLH sample message {"C" or a time sync message {"T".

After all of the JSON object members have been extracted, the corresponding timestamp and relevant sensor information are stored in a MySQL DBS [50] and backed up in a plain comma separated values (CSV) file. In Table 2, an example of resulting INSERT query issued on a DBS is shown, where a table of two EP timestamped sample values are stored.

6. Prototype Testing Results

In order to test the EP and CLH devices in real marine weather conditions, we selected an initial test scenario at the beach of Ensenada, Mexico. In Figure 23 we show the implementation scenario that details the relative distance and elevation of the deployed nodes and BS.

Our custom web server was installed within the Telematics building of the research center known as CICESE (Centro de Investigación Científica y de Educación Superior de Ensenada) on a hill, overlooking the Pacific Ocean. In Figure 24, a satellite image is shown of the experimental node long-range locations.

The experimental setup was left running for a week at the Ensenada test site. In this scenario, CLH to BS messaging was tested and the received data from both CLH and EP sensors was validated and stored at the DB server. The EP node sent acquired sensor data in its native API messages to the CLH every ten seconds. During that time, the CLH constructed timestamped LJSON messages, aggregated the most recent local and EP sensor data, and sent them towards the BS. LJSON to JSON translation was done by the BS, and afterwards the application message was relayed by the BS through the gateway, reaching the web server for final data storage. Figure 25 shows a 7-day log of the water temperature measured by the EP at the beach. And in Figures 26, 27, and 28, the cluster-heads air temperature, relative humidity, and

Table 2: Example of resulting MySQL INSERT command strings.

ID	TIME_STAMP ▼	CLH_ID	EP_ID	n	ADC0	ADC1	ADC2	ADC3	ADC4	ADC5
277	1334685220	1	2	4	188	230	274	495	0	0
276	1334685218	1	2	4	193	234	279	504	0	0
275	1334685217	1	1	4	1023	1023	1023	1023	0	0
274	1334685216	1	2	4	198	237	282	512	0	0
272	1334685214	1	1	4	1023	1023	1023	1023	0	0
273	1334685214	1	2	4	205	243	288	526	0	0
271	1334685212	1	2	4	208	244	290	527	0	0
270	1334685211	1	1	4	1023	1023	1023	1023	0	0

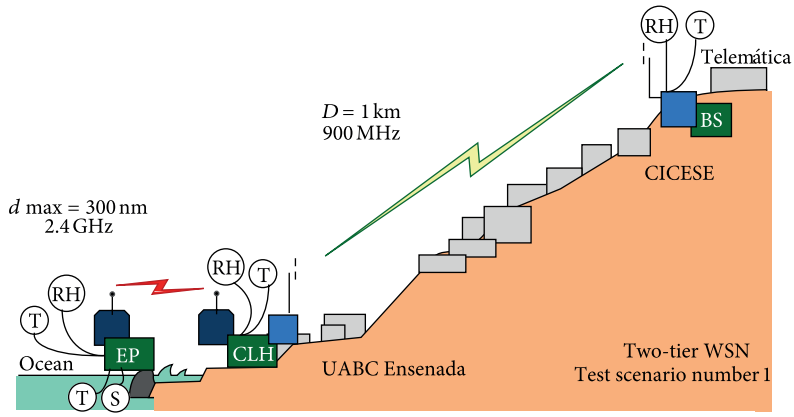

Figure 23: WSN test scenario where the BS is located on top of a hill.

Figure 24: WSN device real weather testing at Ensenada, Mexico.

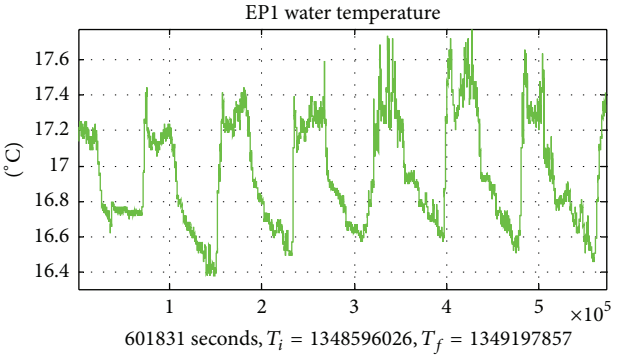

Figure 25: Water temperature oscillations measured by an EP.

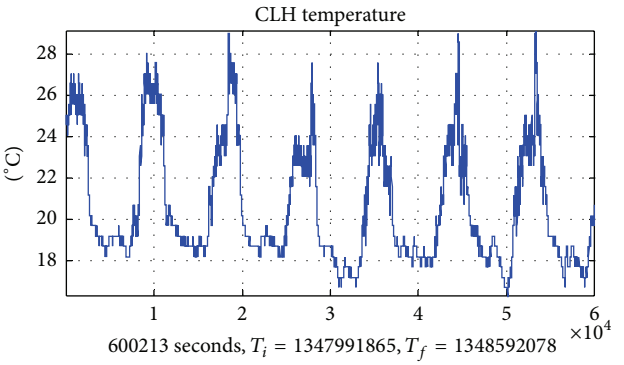

Figure 26: CLH air temperature at seven-day measurements.

battery voltage measurements are presented, where day/night oscillations can be identified clearly.

The actual intended deployment of this low depth marine habitat WSN is Bahia Falsa, Mexico. Marine infrastructure is needed to house the remote telecommunication and electronic instrumentation systems that make up both EP and CLH systems. Resources such as moored floating buoys or

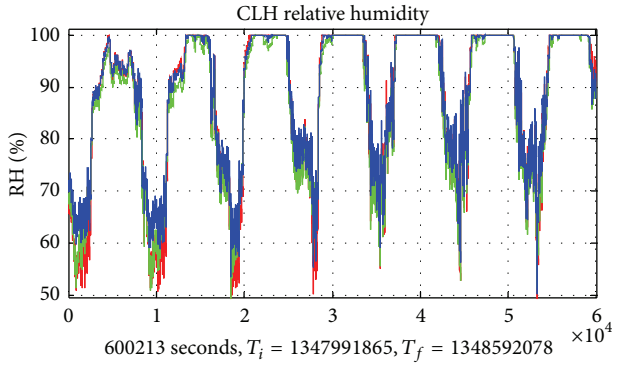

Figure 27: CLH week long air relative humidity readings.

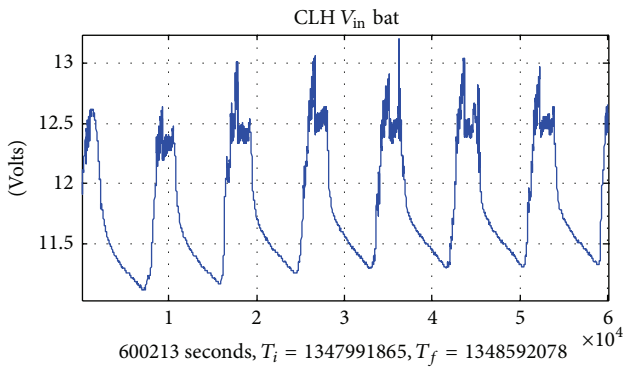

Figure 28: Week long solar panel CLH battery charging cycles.

mast installations properly anchored to the sea floor were considered for this purpose [51]. In this sense, we made progress developing an experimental buoy for shallow water, which we built and now have at our disposal. The initial test involved putting one of our CLH systems, including batteries and charger, within a water-tight enclosure onboard our buoy prototype. We also installed encapsulated photovoltaic (PV) cell arrays for battery recharging purposes, and this is shown in Figure 29.

At Bahia Falsa we did CLH range tests at fixed 200 mW Tx power levels. We set up at the dock of Ostricola Nautilus oyster farm an Ethernet crossover connection between the BS system and a laptop, where we ran our server for data logging purposes. The oyster farm is licensed to operate within Bahia Falsa by the Mexican government. We assembled our buoy at the dock and then dragged it with a boat towards an anchored research raft, called Balsa Nautilus, at more than a kilometer away. A satellite composite image of the boats path is shown in Figure 30. Also, the received power recorded at the BS is shown in Figure 31. The fading RSSI effect shows the relative distance gained by the buoy as we dragged it along the bays channel.

These initial buoy test results not only served as the first step in verifying and debugging the design approach and operation of our systems, but also pointed the way towards real environment long-term testing of the overall electronics and algorithms that were presented here.

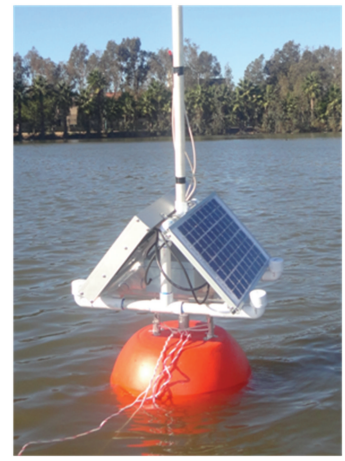

Figure 29: Prototype buoy for shallow bodies of water.

Figure 30: Path taken while testing the telemetry buoys range.

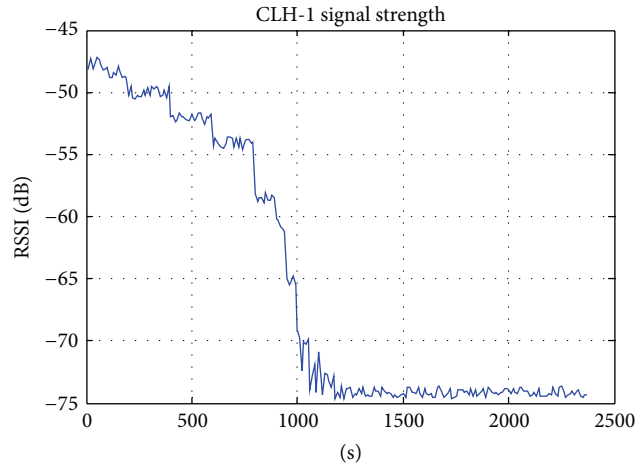

Figure 31: RSSI values recorded at the BS during the buoys displacement towards the experimental raft.

7. LJSON Compared to Other WSN Internet Integration Approaches

The advantages of connecting a WSN to the Internet cloud are obvious, but the way to go about it is unclear. Whether a sensor node should be completely integrated inside the Internet or if it should maintain its independence, as an isolated or adjacent element within its own WSN cloud, is still a valid discussion [52, 53].

In the first Internet integration approach, all interaction of the WSN with the outside world is done by intermediate software that can access front-end stored sensor data, responding to queries sent by a web host.

A second method is called the Gateway solution. In this approach, an Internet enabled BS works as the intermediary between both the world wide web cloud and its own WSN. The BS has an IP address, and sensor nodes are associated to its IP address and have their own node ID. This means that a web host and a WSN node communicate indirectly through the BS gateway at an application level, and the BS relays messages and routes them locally.

The third and last approach is the IP overlay solution, where the web host and the WSN node communicate directly, the BS is only a means for the information exchange to take place. The BS only works at the routing and transport layers.

Our approach works as an application gateway solution. The BS functions as an intermediary that translates WSN application layer LJSON messages, generated by distributed upper tier CLH nodes, to JSON messages that are relayed towards a web service. A feature comparison of this LJSON/JSON approach against other WSN web integration methods is presented in Table 3.

The main benefit in using LJSON is that application layer messages originate within the WSN at the CLH level, and when they reach the sink node, the BS converts them to JSON through a simple translation process and relays them to the custom web service for DB storage. Likewise, a web host can communicate with a CLH node in an inverse fashion, making this scheme a partial gateway solution.

Table 3: WSN web integration method features.

Approach	Integration strategy	Web messaging	Transport layer
CoAP	IP overlay	XML	UDP
Sensor network common interface	Gateway solution	SOAP-XML	TCP
ConnectAPI	Gateway solution	XML	TCP
LJSON	Gateway solution	JSON-PHP service	TCP

8. Conclusions

In general, a WSN deployment of this sort is challenging because of several different physical factors involved, such as the great distance between sensor fields and the telemetry base station. Also, harsh weather conditions present on the ocean surface and extreme humidity that the wireless channel posses may cause signal fading. This means that not only sensor information needs to get to the base station for operational evaluation, but other data have to be available also, such as the cluster ID, originating EP addresses, remaining battery charge, and received signal strengths at different hops within the hierarchy. We included this augmented WSN cross-layer information in our application layer messages for a complete functional assessment of the quality of the wireless links and of the data being transferred. This implies that the information flowing from EP all the way to the BS needs to be within highly structured messages, which at the end have to be transferred to a remote DBS for final storage.

End-to-end approaches aimed to achieve data delivery were deployed and tested. Initial tests were done on EP systems using their native API frame structures, which provide data integrity at the CLH level. Meanwhile, the LJSON encoder, running on the CLH systems, creates aggregated sensor data messages with a well-known syntax which are validated when they reach the BS system. Noting that LJSON application messages are 8-bit casted "quoteless" strings, with shorter lengths compared to 16-bit JSON, which are exchanged at the energy constrained WSN. In this case, LJSON message buffering takes place at the BS, where all incoming LJSON messages are converted to standard JSON and vice versa. Afterwards, the BS opens a socket connection and the resulting messages are sent to a custom TCP/JSON data server, which finally decodes incoming messages and stores extracted timestamped sensor data in a nonvolatile repository. The main advantage in using LJSON/JSON is that M2 M communication from a web host towards the BS or CLH and vice versa is done with a messaging scheme that can be translated with little effort and efficiently parsed to extract sensor data and operational states, and also it can be used to reconfigure CLH and EP functionality. This enables the distributed wireless sensors to participate—in their own way—in the ever-growing web of things (WoT).

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] K. Römer and F. Mattern, "The design space of wireless sensor networks," *IEEE Wireless Communications*, vol. 11, no. 6, pp. 54–61, 2004.

[2] C.-Y. Chong and S. P. Kumar, "Sensor networks: evolution, opportunities, and challenges," *Institute of Electrical and Electronics Engineers*, vol. 91, no. 8, pp. 1247–1256, 2003.

[3] K. Martinez, J. K. Hart, and R. Ong, "Environmental sensor networks," *IEEE Computer*, vol. 37, no. 8, pp. 50–56, 2004.

[4] G. V. Merret and Y. K. Tan, *Wireless Sensor Networks: Application-Centric Design*, InTech, Rijeka, Croatia, 2010.

[5] N. Matthys, R. Afzal, C. Huygens et al., "Towards fine-grained and application-centric access for wireless sensor networks," in *Proceedings of the ACM Symposium on Applied Computing (SAC '10)*, pp. 793–794, Sierre, Switzerland, March 2010.

[6] P. Barbosa, N. M. White, and N. R. Harris, "Design challenges in application-aware wireless sensor networks," in *Proceedings of the 5th Iberian Conference on Information Systems and Technologies, (CISTI '10)*, Santiago de Compostela, Spain, June 2010.

[7] B. S. Kaler and M. K. Kaler, "Challenges in Wireless Sensor Networks," *ACM SIGBED Review*, vol. 1, no. 2, pp. 142–142, 2004.

[8] A. M. Popescu, G. I. Tudorache, B. Peng, and A. H. Kemp, "Surveying position based routing protocols for wireless sensor and ad-hoc networks," *International Journal of Communication Networks and Information Security*, vol. 4, no. 1, 2012.

[9] S. Giordano, I. Stojmenovic, and L. Blazevic, "Position-based routing in ad hoc networks," *IEEE Communications Magazine*, vol. 40, no. 7, pp. 128–134, 2002.

[10] W3C, "HTML 4.01 specification," W3C Recommendation, 1999, http://www.w3.org/TR/REC-html40/.

[11] W3C, "Extensible markup language, XML," 1998, http://www.w3.org/TR/REC-xml/.

[12] "JSON: java script object notation std. spec," Internet Engineering Task Force RFC4627, 2006, http://www.ietf.org/rfc/rfc4627.txt.

[13] JSON-RPC Working Group, "JSON-RPC 2.0 Specification," 2013, http://www.jsonrpc.org/specification.

[14] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, "IPv6 over low-power wireless sensor networks (6LowPAN): overview, assumptions, problem statement, and goals," Internet Engineering Task Force RFC4919, 2007, http://tools.ietf.org/html/rf4919.

[15] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, "Transmission of IPv6 packets over IEEE 802.15.4 networks," Internet Engineering Task Force RFC4944, 2007.

[16] T. Winter, "RPL: IPv6 routing protocol for low-power and lossy networks," Internet Engineering Task Force RFC6550, 2012.

[17] O. Liskin, L. Singer, and K. Schneider, "Welcome to the real world. a notation for modeling REST services," *Internet Computing, IEEE Computer Society*, vol. 16, no. 4, pp. 36–44, 2012.

[18] W. Colitti, K. Steenhaut, N. De Caro, B. Buta, and V. Dobrota, "REST enabled wireless sensor networks for seamless integration with web applications," in *Proceedings of the 8th IEEE International Conference on Mobile Ad-hoc and Sensor Systems, (MASS '11)*, pp. 867–872, Valencia, Spain, October 2011.

[19] C. Ortega-Corral, L. E. Palafox, J. A. García-Macías et al., "A "Lighter" JSON for message exchange in highly resource constrained wireless sensor network applications," in *Proceedings of the Congreso Internacional de Electrónica (ELECTRO '12)*, Chihuahua, Mexico, October 2012.

[20] Digi Inc, "XBee/XBee-PRO 2.4 GHz OEM RF Modules," Product manual v1.xCx–802.15.4 protocol for OEM RF module part no. XB24-...-001, XBP24-...-001 IEEE 802.15.4 OEM RF modules by Digi International. MaxStream, 2008.

[21] Digi Inc, "XBee/XBee-PRO 2.4 GHz ZB RF modules technical manual," 2011, http://www.digi.com/.

[22] M. Lehning, N. Dawes, M. Bavay, M. Parlange, S. Nath, and F. Zhao, "Instrumenting the earth: next-generation sensor networks and environmental science," *The Fourth Paradigm, Data-Intensive Scientific Discovery*, Microsoft Research, pp. 45–51, 2009.

[23] G. Tarapata, J. Weremczuk, R. Jachowicz, X. C. Shan, and C. W. P. Shi, "Construction of wireless sensor for harsh environment operation," *Procedia Chemistry*, vol. 1, no. 1, pp. 465–468, September 2009.

[24] M. Loy, R. Karingattil, and L. Williams, "ISM-band and short range device regulatory compliance overview," Texas Instruments, 2005.

[25] M. Dolujanov, *Propagation of Radio Waves*, YPCC, Moscou, Russia, 1995.

[26] M. Britton and L. Sacks, "The SECOAS project: development of a self-organising, wireless sensor network for environmental monitoring," in *Proceedings of the 2nd International Workshop on Sensor and Actor Network Protocols and Applications (SANPA '04)*, pp. 1–7, Boston, Mass, USA, August 2004.

[27] K. Liu, Z. Yang, F. Hong et al., "Oceansense: monitoring the sea with wireless sensor networks," Tech. Rep. vol. 108, no. 318, SANE2008-81, Institute of Electronics, Information and Communication Engineers, 2008.

[28] Z. Jin, F. Hong, H. Chu, T. Shan, and Z. Guo, "Osnweb: sensor web of oceansense," in *Proceedng of the 6th International Conference on Wireless Communications Networking and Mobile Computing (WiCOM '10)*, Chengdu, China, September 2010.

[29] T. Myers, I. Atkinson, and R. Johnstone, "Semantically enabling the SEMAT project: extending marine sensor networks for decision support and hypothesis testing," in *Proceedings of the 4th International Conference on Complex, Intelligent and Software Intensive Systems (CISIS '10)*, pp. 974–979, Kraków, Poland, February 2010.

[30] S. Bainbridge, "GBROOS—an ocean observing system for the great barrier reef," in *Proceedings of the 11th International Coral reef Symposium*, Session Number 16, Fort Lauderdale, Fla, USA, July 2008.

[31] P. Jiang, H. Xia, Z. He, and Z. Wang, "Design of a water environment monitoring system based on wireless sensor networks," *Sensors Journal*, vol. 9, no. 8, pp. 6411–6434, 2009.

[32] S. K. Singh, M. P. Singh, and D. K. Singh, "A survey of energy efficient hierarchical cluster-based routing in wireless sensor networks," *International Journal of Advanced Networking and Applications*, vol. 2, no. 2, pp. 570–580, 2010.

[33] E. Ravasz, A.-L. Barabasi, and D. K. Singh, "Hierarchical organization in complex networks," *Physical Review E*, vol. 67, no. 2, Article ID 026112, 2003.

[34] M. Ribas-Ribas, J. M. Hernández-Ayón, V. F. Camacho-Ibar et al., "Effects of upwelling, tides and biological processes on the inorganic carbon system of a coastal lagoon in Baja California," *Estuarine, Coastal and Shelf Science*, vol. 95, Elsevier Press, no. 4, pp. 367–376, 2011.

[35] V. F. Camacho-Ibar, J. D. Carriquiry, and S. V. Smith, "Non-conservative P and N fluxes and net ecosystem production in San Quintin Bay, México," *Estuaries Journal*, vol. 26, no. 5, pp. 1220–1237, 2003.

[36] L. Navia, "White paper for wireless IP deployment: Topic I.900 MHz .vs. 2.4 GHz and 5.8 GHz," 2010, http://www.ofinet.net.

[37] Zigbee Alliance, "Zigbee specification," ZigBee Document 053474r13. San Ramon, Calif, USA, 2006.

[38] G. Tarapata, J. Weremczuk, R. Jachowicz, X. C. Shan, and C. W. P. Shi, "Construction of wireless sensor for harsh environment operation," *Procedia Chemistry*, vol. 1, no. 1, pp. 465–468, September 2009.

[39] M. AL-Bzoor, L. Almazaydeh, and S. S. Rizvi, "Hierarchical coordination for data gathering (HCDG) in wireless sensor networks," *International Journal of Computer Science and Security*, vol. 5, no. 5, pp. 443–455, 2011.

[40] R. Faludi, *Building Wireless Sensor Networks*, O'Reilly Media, 2010.

[41] S. P. Lim and G. H. Yeap, "Centralised smart home control system via XBee transceivers," in *Proceedings of the IEEE Colloquium on Humanities, Science and Engineering (CHUSER '11)*, pp. 327–330, Penang, Malaysia, December 2011.

[42] A. H. Kioumars and L. Tang, "ATmega and XBee-based wireless sensing," in *Proceedings of the 5th International Conference on Automation, Robotics and Applications (ICARA '11)*, pp. 351–356, Wellington, New Zealand, December 2011.

[43] A. Rapp, "Arduino library for communicating with XBees in API mode," 2012, http://code.google.com/p/xbee-arduino.

[44] Laird, 2012, http://www.lairdtech.com.

[45] Laird, "AC4490 900 MHz Transceiver," User's Manual. Ver. 3.2.1., 2007.

[46] "JavaScript Object Notation," 2013, http://www.json.org/.

[47] Digi International, "Rabbit core RCM4300 c-programmable analog core module with microSD card storage and ethernet," User's Manual, 2010.

[48] Digicorp, "Dynamic C. integrated C development system for rabbit 4000, 5000 and 6000 microprocessor," User's Manual. Digi International Inc, 2011.

[49] PHP Manual, "TCP Socket scripting," 2013, http://php.net/manual/en/book.sockets.php.

[50] L. Welling and L. Thomson, *PHP and MySQL Web Development*, 5th edition, 2008.

[51] C. Albaladejo, P. Sánchez, A. Iborra, F. Soto, J. A. López, and R. Torres, "Wireless sensor networks for oceanographic monitoring: a systematic review," *Sensors Journal*, vol. 10, no. 7, pp. 6948–6968, 2010.

[52] S. Lan, M. Qilong, and J. Du, "Architecture of wireless sensor networks for environmental monitoring," in *Proceedings of the International Workshop on Education Technology (ETT '08) and Proceedings of the Training and International Workshop on Geoscience and Remote Sensing (GRS '08)*, pp. 579–582, Shanghai, China, December 2008.

[53] C. Alcaraz, P. Najera, J. Lopez, and R. Roman, "Wireless sensor networks and the internet of things: do we need a complete integration?" in *Proceedings of the 1st International Workshop on the Security of the Internet of Things (SecIoT '10)*, Tokyo, Japan, 2010.