

# Aqua-Net: An Underwater Sensor Network Architecture — Design and Implementation

Zheng Peng, Zhong Zhou, Jun-Hong Cui, Zhijie Shi

{*zhengpeng, zhongzhou, jcui, zshi*} @*enr.uconn.edu*

Computer Science Department, University of Connecticut, Storrs, CT 06269

## Abstract

Underwater sensor network (UWSN) has emerged as a powerful technique for aquatic applications. Different from ground based networks, UWSN features long propagation delay, low bandwidth and high error rate. Limited power capacity and node mobility also make it difficult to apply existing protocols or algorithm designed for terrestrial networks to UWSN. It requires new research at every layer of the protocol stack. There is also a need to provide a general architecture for underwater sensor network. This paper proposes AQUA-NET, an architecture for underwater sensor network. Aqua-Net follows a layered structure and have much potential to support various optimizations. It serves as a groundwork for future advances for architectural and protocol designs.

## I. INTRODUCTION

Underwater sensor network (UWSN) is a wireless communication system that consists of battery-powered vessels, sensors nodes and a variety of devices. The fundamental differences between UWSN and terrestrial networks are the channel and signal characteristics. Unlike ground networks, high frequency electromagnetic wave doesn't work well in the water due to the conducting nature of the medium. Acoustic communication becomes the most commonly used technique and brings new challenges [4] to this area because of high attenuation and long propagation delay.

Recent years witness an increasing research interests and progresses in this area. Both academic researchers and engineers made a lot of efforts and many applications, network protocols and devices have been introduced. Most of them are strongly application driven and are not compatible with each other. To make the communication system more successful, there is need to provide a general architecture for underwater sensor network. [3] is a good starting point that designed an underwater architecture loosely based Open System Interconnection (OSI) model [11].

An underwater architecture should consider the characteristics of the network and make it extendable and flexible enough to accommodate the rapid development of applications and underlying hardware. We propose Aqua-Net, a network framework that follows a layered structure while permitting significant optimization. Different from [3], it supports cross-layer design by introducing a translucent vertical layer that is accessible for all applications and protocols. It's layered structure makes easy integration among implementations of different researchers. And we define an abstract layer, or narrow waist [5], that allow device and protocol developments to proceed apace. Aqua-Net

is a valuable platform that will facilitate the process of application development. So far, a modified ALOHA protocol and two routing protocols have been implemented under Aqua-Net.

The rest of this paper is organized as follows. In Section III, Aqua-Net architecture and design philosophy are described. Related work are introduced in Section II. Followed by Section IV which goes into the implementation details of Aqua-Net. System power consumption is shortly discussed in Section V. In Section VI, we studied the performance of implemented protocols based on some preliminary results. Conclusions and future research directions are discussed in Section VII.

## II. RELATED WORK

Sensor network system has been studied for many years. A lot of work have been done on designing and developing sensors and communication systems [16, 18, 9]. TinyOS [10] is one of those successful software and hardware integrated platforms that is very popular in sensor network research community. And there are other systems, such as EmStar [6], VM\* [12], SOS [2] and T2 [13]. However, some of them mainly focused on software side of the system and none of them are designed for or even considered of underwater environment.

For underwater network, due to its different environment and constrains, limited number of systems are proposed compared with the terrestrial networks. T. C. Austin et al. introduced PARADIGM [1] for underwater network. But it is mainly designed for Autonomous Underwater Vehicles (AUVs) applications. Another pioneering work is done in [3], which proposed an underwater architecture loosely based Open System Interconnection (OSI) model [11]. Aqua-Lab [15] provides researchers underwater field test experiences in a lab controlled environment. Our work is based on the hardware provided by Aqua-Lab.

## III. NETWORK ARCHITECTURE

Recent years witness an increasing research interests and progresses in underwater sensor network. Both academic researchers and engineers made a lot of efforts and many applications, network protocols and devices have been introduced. Most of UWSN applications and hardware are strongly application driven and are not compatible with each other. To make a successful communication system, there is need to provide a general architecture for underwater sensor network.

An underwater architecture should consider the characteristics of the network, the underlying channel and environment. To make it more extendable and flexible, Aqua-Net follows a layered structure, which is effective in accommodating the rapid development of applications and hardwares. Aqua-Net also has potentials in permitting significant optimization. Different from [3], it supports cross-layer design by introducing a translucent vertical layer that is accessible for all applications and protocols. It's layered structure makes easy integration among implementations of different researchers. And we define an abstract layer, or narrow waist [5], that allow device and protocol develop-

ments to proceed apace. Aqua-Net is a valuable platform that will facilitate the process of application development.

### A. *Design philosophy*

Protocol stack should be extendable. This means new modules should be able to be integrated into the stack without too much trouble. It inexplicitly requires an open architecture.

Protocol stack should be debugging friendly. Debugging is always a headache for system software. The protocol stack is such a system and may get more complicated during its evolution. An error in the software may get notoriously difficult to detect. It's very critical to make sure the system is easy to debug.

Protocol stack should be upgradable. As time goes by and feedbacks collected from users, protocol stack may need to have modifications to better satisfy the needs of applications. So the stack should be easy to upgrade.

#### A.1 Cross-Layer Design

Any design reflects special properties of the target system. Most networks, such as the Internet, follows an architectural structure. However, a strict layered design is not flexible enough to deal with the dynamics of underwater environments and may prevent performance optimizations. Underwater network adopts acoustic signal to communicate in a highly impaired environment. Due to the unique features of large latency, low bandwidth, and high error rate, underwater acoustic channels bring many challenges to the protocol design [4]. Energy consumption is another critical issue, since most of current networking hardware are battery powered and have limited capacities. Network protocols have to be very efficient to survive such a harsh environment. This makes cross-layer design a popular methodology among researchers. Note that cross-layer design should not be a non-layered or single-layered design, which compromise the overall ability to provide continuous growth in scale and diversity of the whole system.

Cross-layer design is necessary because:

- System energy consumption is affected by all layers. On the other hand, all layers should co-operate to reduce overall system energy consumption. Cross-layer design is a way to make such co-operation possible.
- Network protocols need a bigger picture of the system. This requires some parameters be accessible to multiple layers. However, according to [19], in networks with high attenuation channels, all cross-layer design, can only improve throughput by at most a constant factor. And layered design could be order-optimal.

The AquaNet adopts a cross-layer architecture which offers an alternative to the pure layered approach that promotes stricter local interaction among protocols in underwater nodes. Useful cross-layer feedbacks will be utilized by other layers to improve system performance. Table III-A.1 shows examples of these parameters.

TABLE I  
USEFUL CROSS-LAYER INFORMATION

Physical	Channel Condition (e.g. BER) Transmission Power Channel Assignment
Link/MAC	Number of retransmissions Back off time
Network	Mobile-IP hand-off begin/end information
Transport	Packet Error Rate
Application	Acceptable delay Packet loses
User	User behavior End to end delay

## A.2 Lowering of the Waist

Architectural design provides abstract layers for different level users. This makes it easier and possible to implement a system that could be used for a variety of applications and networking devices. There should be a set of shared interfaces and components to bridge the gap between applications and hardwares. The most essential component is referred as "narrow waist", just like Internet Protocol (IP) for Internet. This idea is first introduced by [5]. Depending on the properties of the network, the location of "narrow waist" varies. [5] and [17] proposed a Sensor-net Protocol (SP) to be the "narrow waist" of sensor network. We further argue that this waist could be even lower to Media Access Control (MAC) layer in underwater sensor network.

Internet is a "network of networks" and designed to provide end-to-end communication between heterogenous autonomous systems. IP naturally becomes the bridge, or narrow waist, of Internet. Similarly, for underwater network, there also exists such narrow waist. However, underwater networks are more application driven and have more security concerns. It doesn't even need to interoperate between different network system. The Ad-hoc feature also implicitly focus more on single-hop best-effort communication. MAC layer thus emerges as a good candidates of the narrow waist of underwater sensor network.

## B. System Architecture

As described in IV-B, protocol stack could be implemented in either kernel space or user space. To make the system more flexible and easier to use, we choose to make the system running in user space. So it doesn't involve any kernel

support and will not be affected by kernel upgrading or customizing.

Figure 2 shows the structure of the protocol stack. Apparently we are not using system socket. Instead, we developed a pseudo BSD socket interface. So end user is still able to use similar interfaces for their implementation. So one of the key problems is, how to emulate BSD socket for ordinary user. Another problem is how to provide the service to the user, or in other words, how to establish communications between emulated BSD socket to our protocol stack. We will see how these be handled in the following section.

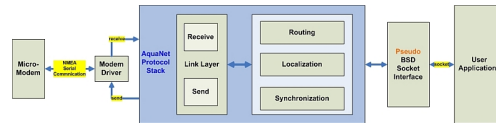


Fig. 1

System Architecture

#### IV. IMPLEMENTATION

##### A. Hardware platform: Gumstix

Underwater sensor network is a challenging area, considering all the limitations in the channel and physical devices. Experiences from terrestrial may not apply in such a harsh environment. Latest technologies, including software and hardware, should be adopted to improve overall system performance. Take energy consumption as an example. Simple schemes are not enough for underwater network devices which are assumably powered by batteries and consume more power than their terrestrial counterparts. Advanced algorithms are proposed to reduce energy consumption but require more computation power from processor and support from advanced operating system.

Gumstix[8] is our choice as a result of continues search for energy efficient, compact, high performance and inexpensive device. Gumstix is based on Marvell PXA270 with XScale microprocessor core. It's the fifth generation of the ARM architecture. Our Gumstix system board has 64MB RAM and 16MB flash and is running at 400 MHz main frequency.

##### B. Operating System: Embedded Linux

Embedded Linux is the operating system designed or optimized for devices that have limited resources, such as battery life, computational power (compared with main frame desktops) and storage capacity. It's an open source platform that could be customized by users to suit the need of a specific application. People do not need to learn a new set of developing tools to work on embedded Linux. This makes the development much easier. As a variant of Linux, embedded Linux provides similar, if not the same, features, and has more supports from open source community. So

researchers may not need to invent every wheel they want, instead, there are a lot of libraries and tools available to use right out of the box. In all, embedded Linux is an advanced, customizable, developer friendly and ever-developing operating system that is ideal for developing underwater network protocol stack.

### *C. Protocol Stack*

The protocol stack is a framework that provides various network services. In other words, both network application and protocol developers rely on the services provided by a protocol stack. There are three interfaces for different purposes.

When implementing protocols in Linux, there are typically two options: kernel space or user space. Protocol implemented in kernel space is very efficient. However, it also has some limitations. The first issue is robustness. Since it's in system kernel, if the protocol has any flaw, it may lead to disastrous consequences and get the whole system crashed. Secondly, it's more difficult to debug when the protocol is running in kernel space. There are a few tools available [14][7], but still not very easy for people that are not very familiar with Linux kernel. What's more, the code may not be compatible with new Linux kernels. So the protocol and the system running the protocol may not be able to get benefit from latest kernel improvements, such as better security features.

For user space protocol, we also have two design choices. Just like how OPSF and RIP are implemented in Linux, we could find a way to fully utilize operating system to develop our own protocols. Since Linux is a network operating system, networking is an essential part of the system. So many fundamental protocols, TCP/IP for example, are implemented in Linux kernel. And all network data is handled internally. However, the operating system also provides several ways for user space application to communicate with the kernel. This makes it possible to implement some protocols in user space and get all the flexibilities of a user space program.

But this method has some limitations. First, Data need to go through the whole TCP/IP protocol stack if using STREAM or DGRAM socket type. Second, if asking user to use PF-PACKET, it requires a lot of system techniques and will complicate application development. Moreover, it still has kernel modules which may be affected by changes/evolutions of Linux kernel. Last, it involves communication between kernel space program and user space program, which is not very efficient in Linux. So that's why we choose to implement the protocol stack solely in user space as an application and provide a pseudo-BSD socket.

The pseudo-BSD socket is designed as an interface for network application developers. The application developers will concentrate on their application logic and don't need to know too much about the underlying network. By providing function calls with similar names as the ones in BSD Socket, developers may be able to compile and run their existing codes with no or minor modifications. Behind the sockets, communication among sockets and other system components are done with Unix Interprocess Communication (IPC) techniques. For network protocol designers, they should be able to pick up their interested packets, do all the necessary processings and usually (but not always) send

this packet back to an incoming/outgoing packet queue.

Inside the protocol stack, there's another interface we need to implement. It is the lowest layer and is connected to a real network. In Aqua-Net architecture, this refers to a serial port and related programs that control it. Besides these interfaces, the framework has some routines to coordinate all the components. An important routine is to keep all system information. This is actually a vertical layer in our software architecture which makes system information available to all layers. So cross-layer design is possible in our system. The implementation of protocol stack includes the following techniques: shared memory, semaphore and multi-threading. Figure2 demonstrates our design of protocol stack.

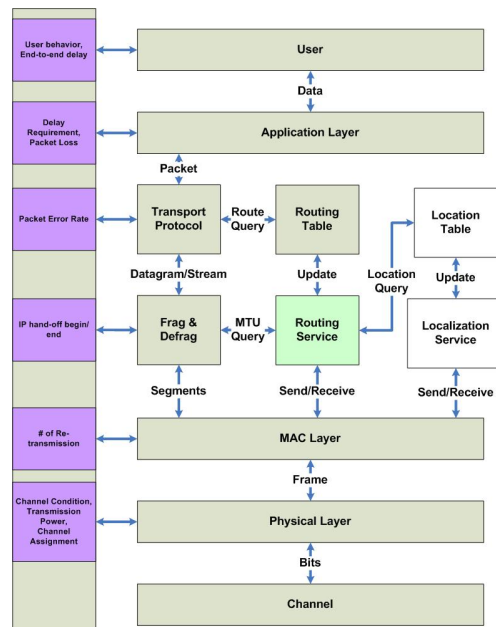


Fig. 2

Protocol Stack

Additional protocols could be implemented in Aqua-Net system. Between user developed protocol and Aqua-Net system, data contents are stored as plain HEX strings. In this way, the change of any protocol will not affect the functionality of the whole system. Neither layer needs to understand the design details of the other layer. It is the requirement of a layered system structure. The parameters of the protocol should be kept in a config file. Therefore, when a user needs to change any setting of its protocol, it doesn't need to re-compile the whole system. This could save a lot of time and efforts.

## D. BSD Socket Emulating

Typical socket system calls include the following functions:

- **socket()**: Get the File Descriptor.
- **sendto()**: The function to sends a package.
- **recvfrom()**: The function that receives an incoming package.
- **close()**: Close the transaction and related file descriptor.

These are the essential component of network programming. In our implementation, we re-invent these functions with similar parameters and functionalities. Connection oriented calls, such as *send()* and *recv()*, are not implemented. Because we don't have a reliable data transfer protocol implemented yet. But empty functions with similar names are reserved for future use.

## E. Implemented Protocols

### E.1 Modified ALOHA

MAC protocol is a sub-layer of Data Link Layer which is a part of OSI seven layer network architecture. It provides a way for multiple network nodes to communicate with each other in a shared media. When multiple nodes transmit data simultaneously, these signals may collide at the receiver side and thus make it difficult or even impossible to receive the data. In this case, network bandwidth is wasted on data retransmissions. MAC protocols is then designed to avoid, if not eliminate, the possible collisions and improve network throughput. In a wireless network, due to limited transmission range, a single node have no access to global information. This makes hidden terminal and exposed terminal problems occur. Unlike what CSMA/CD does in 802.3 Ethernet, the half duplex property of wireless devices also makes carrier sensing difficult. In underwater environment, much slower signal propagation speed ( $3 \times 10^8 m/s$  vs.  $1.5 \times 10^2 m/s$ ) means individual sensor node not only has limited knowledge of the network, but also has much longer delay in perceiving the events in the neighborhood. It is even more difficult to design a MAC protocol in this scenario.

In this paper, to demonstrate the use of AquaNet, an ALOHA based MAC protocol has been implemented. Although ALOHA protocol is quite simple, it is the basis of many advanced protocols such as CSMA/CA in 802.11 wireless network. The proposed MAC protocol incorporates acknowledgement into ALOHA to meet the characteristics of underwater wireless network. As discussed before, hidden and exposed terminal problem exists in wireless network. Pure ALOHA cannot help because collisions happen on receiver side which senders don't have any feedback from. In 802.11, it provides RTS/CTS as an additional collision avoidance approach to original CSMA/CA. It utilize RTS/CTS mechanism to coordinate sender and receiver. Because RTS/CTS packets are smaller than data packets, it introduces

lower cost (both on control packet size and collision probability) but has better system performance by decreasing data collisions. This method is widely used in wireless network. However, in underwater scenario, most acoustic modems have fix frame length and don't support small control packet. In this case, if we use normal data frame as RTS/CTS for control purpose, it only increases system overhead and introduces more collisions. So this paper takes another approach: MAC acknowledgement. The introduction of acknowledgement serves two purposes. Firstly, it's a feedback from the receivers to the sender and explicitly inform the sender if a packet is received or not. On the other hand, it also acts like CTS message to suppress other node from sending data. Figure.3 shows the work flow of the modified ALOHA protocol.

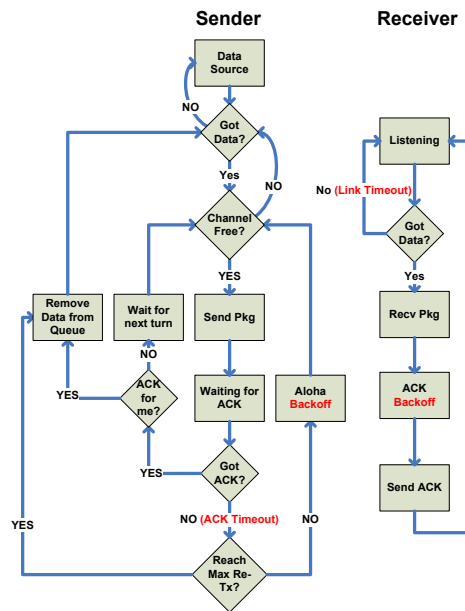


Fig. 3

Work Flow of the Modified ALOHA Protocol

## E.2 Routing Protocol

- VBF Routing:** As a case study, we implement a routing protocol, VBF (Vector-Based Forwarding) [20], in our protocol stack. VBF provides robust, scalable and energy efficient routing for underwater sensor networks. It is a position-based routing protocol: nodes close to the vector from the source to the destination will forward the packets; other nodes out of the range will simply discard the packets. In this way, only a small fraction of the nodes are involved in routing. VBF also adopts a localized and distributed self-adaptation algorithm which allows nodes to weigh the benefit of forwarding packets and thus reduce energy consumption by discarding the low benefit packets. For simplicity, in this paper, we implement a simple version of VBF (without self-adaption) using APIs provided by Aqua-stack.

- **Static Optimal Routing:** To compare the performance of routing protocols, an optimal static routing protocol is implemented. It reads routing information from a user created file. Routing decisions will be made according to the user-defined routing table. Because global information is provided in the routing table, this protocol should give us optimal performance in the network layer.

## V. POWER CONSUMPTION

The current system can run for about 10 hours using batteries with 2000 mAh capacity. It will draw  $70mA \sim 250mA$  current when ethernet is disabled, while increase to  $250mA \sim 400mA$  when enabled. The system lifetime could be further extended to at least 1 day, which means the system will only consume 200mA of current on average. There are several ways of doing that:

- **Sleep Mode:** Gumstix does provide a sleep mode and could be waken up by GPIO or RTC. However, these methods are not widely used because of bugs. People report that these two ways may either crash the kernel or put the file system into read only mode. We will explore more on this and see if there're ways to walk around.

- **Frequency Scaling:** A method that has been implemented for laptops. With embedded Linux, Gumstix has the potential to do frequency scaling and reduce energy consumption.

- **Two layered hardware:** Another approach is to design a two layered hardware architecture. An external circuit controlled by some low-end micro-controller can cut off Gumstix' power when necessary. And it can turn the power back on according to some user defined scheduling algorithm.

With the frequency scaling technique, the current system power consumption is reduced to about  $135mW$  (full speed without duty cycle). Some may still consider this number as too big for a embedded system. However, in underwater sensor network, it's affordable compared with other power consuming devices. The whole system is not just the board, it includes acoustic modem, transducer and other devices (e.g. GPS and buoy control). Their power consumption will be about 2 orders larger than our system board. So the current system power consumption would not be an issue when it comes to the all-in-one sensor node stage.

## VI. CASE STUDY

To test the system and protocol stack, a simple MAC (Media Access Control) protocol and routing protocols are implemented.

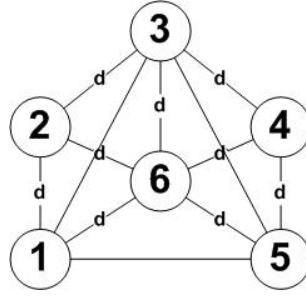


Fig. 4

MAC Test Topology

### A. MAC Protocol

Our test topology is illustrated in Figure.4. The numbers on the lines are the distances between nodes. We intent to set the transmission range to be  $2d$ . However, in reality, it's very difficult to have control on it, due to limited space and long transmission range of our transducer. Node 6 is configured as the receiver and all other nodes are sending data to node 6. This gives us a one hop, multi-source and single-sink topology. Note that everyone can hear from each other in this setting. Micro-modem provides a sending rate at 80bps.

We implemented an application level Poisson traffic generator. We first created a series of exponential distributed numbers. And use these numbers as the packet arrival intervals. Exponential numbers are generated using the following formula:

$$T = \frac{-\ln U}{\lambda} \quad (1)$$

Here  $U$  is uniform on  $(0,1)$  and  $\lambda$  is the packet arrival rate which could be configured by users.

The packet arrival rate of a single node is  $\lambda$ . We increased the number of sending node from 1 to 5 and measured corresponding system throughputs. The MAC protocol has been tested in two scenarios. In the first case, the total system data generation rate is fixed and evenly distributed to individual nodes. For example, if there is only one sending node, data arrives at speed  $\lambda$ ; if there are  $N$  senders, each of them has an arrival rate of  $\frac{\lambda}{N}$ . In the second case, individual node has at a fixed data arrival rate  $\lambda$ . So if  $N$  nodes are sending, the total system data arrival rate is  $N\lambda$ . Lab test result is shown in Figure.5.

In the first scenario, the total system data arrival rate is the same. Note that with 80bps modem sending rate, transmitting one packet will take at least  $3.2 + \epsilon$  seconds.  $\epsilon$  includes overheads like the modem mode switching time and etc. So the maximal arrival rate should be less than  $\frac{1}{3.2} = 0.3125$ . When MAC involves, it introduces extra costs. The sender needs to wait until an ACK is received or time out. In the best case it will take  $2 \times (3.2 + \epsilon)$ . So the maximal

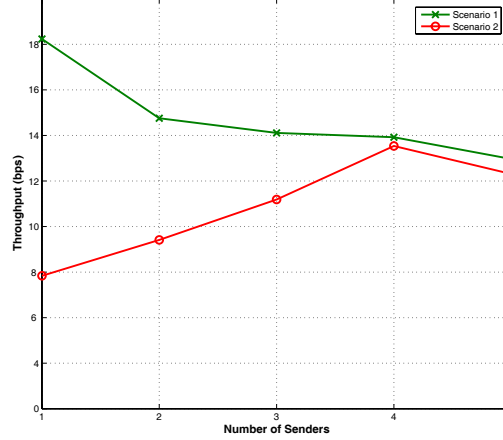


Fig. 5

### MAC Test Results

possible system throughput is further downed to  $\frac{(32 \times 8)}{(2 \times 3.2)} = 40bps$ . So the upper bond of  $\lambda$  becomes  $\frac{1}{2 \times 3.2} = 0.1562$ . In this test, we set total  $\lambda$  to be 0.1, which means that average speaking, the system sends 1 packet per 10 seconds. This equals a maximal system throughput of  $25.6bps$ . As shown in Figure.5, system throughput decreases as more senders get involved. However, when there are more than 3 senders, the difference is small. This is because the system is approaching a point which is the upper limit of ALOHA channel utilization.

In the second test,  $\lambda$  is fixed to be 0.02, which equals  $5.12bps$ . When there is only one sender, it almost archive the target throughput. When there are two senders, since lambda is fixed, the total data rate doubles ( $10.24bps$ ). And the throughput increases to  $7.7bps$ . Similarly, it goes to  $10bps$  with 4 senders but drops to  $8.9bps$  because of the increase of collisions.

It seems that with the current setting, channel utilization decreases with the increase of senders (and thus collisions between senders). But the system throughput increases as the total data rate increases until the data loss caused by network collisions canceled out the increased data rate. In this test, the maximal throughput occurs when there are 4 senders.

## VII. CONCLUSIONS

In this paper, we propose and develop Aqua-Net, a network architecture scalably and efficiently in underwater sensor network. It provides a general software and hardware platform for network researchers. BSD socket-like interface make it easy for user to implement application. User developed protocol can also be plugged in to the system. Also, by lowering the "narrow waist", our system can efficiently support a variety of applications and hardwares. The main

innovation is that while keep the system flexible and make researchers easy to use or incorporate their own protocol, the system also make cross-layer design possible by providing system parameters across all network layers. This facilities the process of doing research and field test.

## REFERENCES

- [1] T. C. Austin, R. P. Stokey, and K. M. Sharp. PARADIGM: a buoy-based system for auv navigation and tracking. In *IEEE Proceedings of Oceans*, 2000.
- [2] C. chieh Han, R. Kumar, R. Shea, E. Kohler, and M. Srivastava. Sos: A dynamic operating system for sensor networks. In *Proceedings of the Third International Conference on Mobile Systems, Applications, And Services (Mobisys)*. ACM Press, 2005.
- [3] M. Chitre, L. Freitag, E. Sozer, S. Shahabudeen, M. Stojanovic, and J. Potter. An architecture for underwater networks. In *Proc. IEEE Oceans06 Asia Pacific Conference*, June 2006.
- [4] J.-H. Cui, J. Kong, M. Gerla, and S. Zhou. Challenges: Building scalable mobile underwater wireless sensor networks for aquatic applications. *IEEE Network, Special Issue on Wireless Sensor Networking*, 20(3):12–18, 2006.
- [5] D. Culler, P. Dutta, C. T. Ee, R. Fonseca, J. Hui, P. Levis, J. Polastre, S. Shenker, I. Stoica, G. Tolle, and J. Zhao. Towards a sensor network architecture: lowering the waistline. In *Proceedings of the 10th conference on Hot Topics in Operating Systems*, volume 10, pages 24–24, 2005.
- [6] L. Girod, N. Ramanathan, J. Elson, T. Stathopoulos, M. Lukac, and D. Estrin. Emstar: A software environment for developing and deploying heterogeneous sensor-actuator networks. *ACM Trans. Sen. Netw.*, 3(3):13, 2007.
- [7] D. Grothe. KGDB: Linux Kernel Source Level Debugger, <http://kgdb.linsyssoft.com>.
- [8] Gumstix Inc. Gumstix - Dream, Design, Deliver, <http://www.gumstix.com>.
- [9] J. Hill, M. Horton, R. Kling, and L. Krishnamurthy. The platforms enabling wireless sensor networks. *Commun. ACM*, 47(6):41–46, 2004.
- [10] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. *SIGPLAN Not.*, 35(11):93–104, 2000.
- [11] ISO/IEC. ISO 7498: Open systems interconnection - basic reference model, 1984.
- [12] J. Koshy and R. Pandey. Vmstar: synthesizing scalable runtime environments for sensor networks. In *SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 243–254, New York, NY, USA, 2005. ACM.
- [13] P. Levis, D. Gay, V. H. J. hinrich Hauer, B. Greenstein, M. Turon, J. Hui, K. Klues, C. Sharp, R. Szewczyk, J. Polastre, P. Buonadonna, L. Nachman, G. Tolle, D. Culler, A. Wolisz, T. U. Berlin, C. Inc, and A. R. Corpration. T2: A second generation os for embedded sensor networks. Technical report, 2005.
- [14] Linux kernel debugger. KDB (Built-in Kernel Debugger), <http://oss.sgi.com/projects/kdb/>.
- [15] Z. Peng, J.-H. Cui, B. Wang, K. Ball, and L. Freitag. An underwater sensor network testbed: Design, implementation and measurement. In *In Proceedings of Second ACM International Workshop on UnderWater Networks (WUWNet'07)*, 2007.
- [16] K. S. J. Pister, J. M. Kahn, and B. E. Boser. Smart dust: Wireless networks of millimeter-scale sensor nodes. 1999.
- [17] J. Polastre, J. Hui, P. Levis, J. Zhao, D. Culler, S. Shenker, and I. Stoica. A unifying link abstraction for wireless sensor networks. In *Proceedings of the Third ACM Conference on Embedded Networked Sensor Systems (SenSys05)*, 2005.
- [18] G. J. Pottie and W. J. Kaiser. Wireless integrated network sensors. *Commun. ACM*, 43(5):51–58, 2000.
- [19] S. Toumpis and A. J. Goldsmith. Performance, optimization, and cross-layer design of media access protocols for wireless ad hoc networks. In *IEEE International Conference on Communications*, 2003.
- [20] P. Xie, J. Cui, and L. Li. Vbf: Vector-based forwarding protocol for underwater sensor networks. In *Proceedings of IFIP Networking*,

*Coimbra , Portugal , May 2006, A longer version is currently available as UCONN CSE Technical Report: UbiNet-TR05-02 (BECAT/CSE-TR-05-5), 2006.*