Goby-Acomms version 2: extensible marshalling, queuing, and link layer interfacing for acoustic telemetry

Citation: Schneider, Toby E., and Henrik Schmidt. "Goby-Acomms Version 2: Extensible Marshalling, Queuing, and Link Layer Interfacing for Acoustic Telemetry." IFAC Proceedings Volumes 45, 27 (2012): 331–335 © IFAC

As Published: http://dx.doi.org/10.3182/20120919-3-IT-2046.00056

Publisher: Elsevier

Persistent URL: http://hdl.handle.net/1721.1/114881

Version: Author's final manuscript: final author's manuscript post peer review, without publisher's formatting or copy editing

Goby-Acomms version 2: extensible marshalling, queuing, and link layer interfacing for acoustic telemetry ⋆

Toby E. Schneider * **Henrik Schmidt** *

* *Center for Ocean Engineering, Dept. of Mechanical Engineering, Massachusetts Institute of Technology, Cambridge, MA 02139 USA (email: tes@mit.edu, henrik@mit.edu).*

Abstract: We present the Goby-Acomms project version 2 (Goby2) which provides software for communication amongst autonomous marine vehicles over extremely bandwidth-constrained links. Goby2's modular design provides four discrete yet interoperable components: 1) physics-oriented marshalling via the Dynamic Compact Control Language (DCCL); 2) dynamic priority queuing; 3) time division multiple access (TDMA) medium access control (MAC); 4) and an extensible link-layer interface (ModemDriver).

Keywords: Communication protocols, autonomous vehicles, marine systems, telemetry, source coding

1. INTRODUCTION

For successful collaboration of autonomous underwater vehicles (AUVs) in tasks ranging from the scientific (e.g. oceanographic sensing; see Petillo et al. (2012)) to commercial and military (e.g. harbor surveillance; see Shafer (2008)), transmission of datagrams is essential to propagate state and sensor data. However, the only practical communications link, one carried by acoustics, has extremely low throughput and high latencies due to the physics governing propagation of sound in the ocean (primarily little available bandwidth, low speed, and multipath due to boundary interactions) For an overview of these challenges see Chitre et al. (2008) and Preisig (2007).

This severe throughput constraint forces the design of novel communications protocols that attempt to maximize the useful data sent. Commonly used terrestrial systems impose overhead requirements that are unacceptable. For example, a common maximum transmission unit (MTU) for acoustic links is 32 bytes and the Internet Protocol (IP) uses a minimum 20 bytes of header alone which would comprise over half of the message. Early solutions to this problem, such as the Compact Control Language (CCL), used hand designed encoders and decoders for every data structure ("message") to be transmitted. While reasonably efficient in terms of message size, adding a new message was time-consuming and error-prone. The original version of the Dynamic Compact Control Language (DCCL1) provided an XML-based structure language that used a common set of source coders. While it made the process of creating new messages significantly quicker and more robust, DCCL1 had only a rudimentary mechanism for using physics-based (e.g. entropy coding) coders. The new version of DCCL (DCCL2) presented in section 2 removes this limitation by providing an interface for writing application specific coders.

Another significant reality caused by the acoustic link's low data rates is that total throughput is rarely or never a possibility. Ideally all collaborating vehicles and the topside human operator would have the entire data set of all vehicles in the network in order to make the best mission decisions. Instead, only a miniscule subset of the data generated by an AUV can be relayed acoustically. This leads to a significant prioritization problem; some of this queuing can be automated by the Goby dynamic priority queuing module (Queue). Goby-Queue is discussed in detail in Schneider and Schmidt (2012b), the paper on Goby version 1 (Goby1), and is not significantly changed in release 2. Goby-Queue is not significantly changed in release 2, and is therefore not further addressed in this paper.

The final significant challenge for underwater telemetry addressed by Goby is the lack of standardization or even significant commonality between different acoustic modems. In order to allow communication that appears seamless to the application in a variety of hardware environments, the Goby interface to the link layer (ModemDriver, see section 3) is extensible and polymorphic, allowing Goby to communicate over any device that can send bytes from point A to point B. At the same time, the application can still use hardware-specific features as desired. In collaboration with the ModemDriver, Goby provides basic time division multiple access (TDMA) medium access control (MAC). Like Goby-Queue, the MAC from Goby1 is conceptually the same in Goby2 and is therefore not treated in this paper.

All of these modules of Goby are designed to work together as diagrammed in Fig. 1, but each has a distinct interface allowing for mix-and-match functionality between Goby and other research or commercial systems.

⋆ This work was sponsored under the Office of Naval Research programs N00014-08-1-0011 and N00014-11-1-0097.

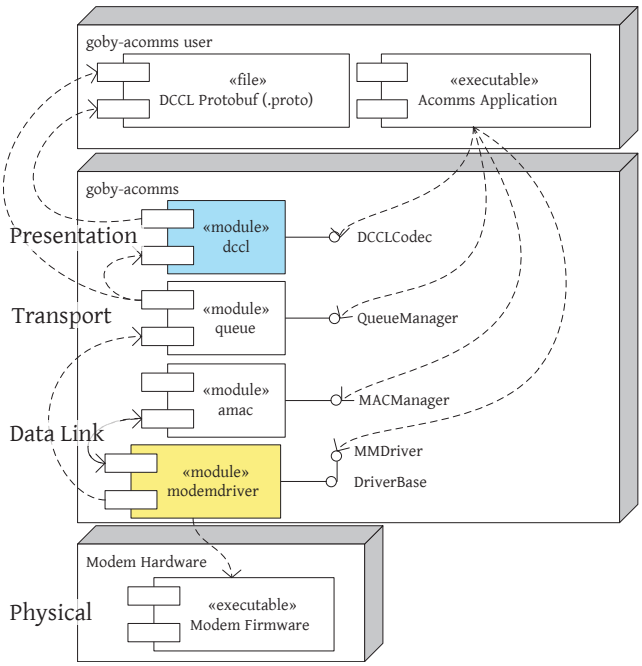

Fig. 1. UML Structure Diagram of Goby2 including dependencies between modules. The approximate Open Sources Initiative (OSI) network layer (see Zimmermann (2002)) is given to the left.

1.1 Philosophy of Goby2

The overall goals of Goby2 versus the prior release are to

(1) improve extensibility by third-party authors. Release 1 was primarily focused on a specific marine vehicle middleware (the MOOS project), and did not offer many expansion opportunities except the ability to define one's own DCCL messages.
(2) promote the development of a system that provides high correctness assurance as far before deployment as possible, since ship time is highly valuable and vehicles are costly. The communications system is, almost by definition, an essential part of collaborative vehicle missions, and thus it cannot fail.

2. MARSHALLING: THE DYNAMIC COMPACT CONTROL LANGUAGE

The Dynamic Compact Control Language (DCCL2) is comprised of two parts designed to make creating very small datagrams straightforward and reliable: a structure language and an encoding library. The language provides a way of representing "methodless classes" or "dumb data objects" that are similar to what can be represented in a C struct, but with the addition of meta-data that provides information allowing optimized encoding.

2.1 DCCL Structure Language

The DCCL2 structure language is based on an extension of the Google Protocol Buffers ("protobuf") language (see Google (2012)). Protobuf was chosen to replace XML, which was used in DCCL1, because it provides static (compile-time) type safety and syntax checking. Developing systems for AUVs involves integrating large codebases

```
8  import "goby/common/protobuf/option_extensions.proto";
5
17 message CTDMessage
   {
     option (goby.msg).dccl.id = 102;
     option (goby.msg).dccl.max_bytes = 64;

100  required int32 destination = 1 [(goby.field).dccl.max=31,
                                      (goby.field).dccl.min=0,
                                      (goby.field).queue.is_dest=true
                                      (goby.field).dccl.in_head=true];

     required uint64 time = 2 [(goby.field).dccl.codec="_time",
                               (goby.field).queue.is_time=true];

30   repeated int32 depth = 3 [(goby.field).dccl.max=1000,
                               (goby.field).dccl.min=0,
                               (goby.field).dccl.max_repeat=10];

     repeated int32 temperature = 4 [(goby.field).dccl.max=40,
                                     (goby.field).dccl.min=0,
                                     (goby.field).dccl.max_repeat=10];
110
     repeated double salinity = 5 [(goby.field).dccl.max=40,
                                   (goby.field).dccl.min=25,
                                   (goby.field).dccl.precision=2,
                                   (goby.field).dccl.max_repeat=10];
   }
```

Fig. 2. Definition of a DCCL message for sending ten samples from a Conductivity-Temperature-Depth (CTD) sensor. On the left is the size of each encoded field in bits; the whole message is 280 bits (35 bytes) including required bit padding on the header and body. For comparison, the default Protobuf encoding uses 81 bytes to encode this message with a representative set of values.

from multiple research centers, and without a high degree of compile-time correctness assurance, costly ship time will be wasted tracking down avoidable software bugs. Since DCCL1 was defined in XML, all correctness checking was done at runtime, deferring detection of syntactical and type errors.

Furthermore, Protobuf messages (and thus their derivatives, DCCL messages) use a compiler that produces native C++ classes, allowing for high efficiency which is critical for embedded systems. Due to the fundamental tradeoff between power and longevity (as detailed in Bradley et al. (2001)), AUVs can only support low performance (but low power) hardware, such as ARM based computers. C++ provides an excellent balance between programming ease and runtime efficiency.

Finally, Protobuf provides class introspection, which allows DCCL to operate on arbitrary user-provided messages, which can be compiled into the application or loaded dynamically either through shared libraries or runtime compilation of the DCCL message definition.

An example of the syntax of the structure language is given in Fig. 2. Also in that figure is the encoded size of the message. As in DCCL1, byte boundaries are dissolved and fields can be comprised of any whole number of bits.

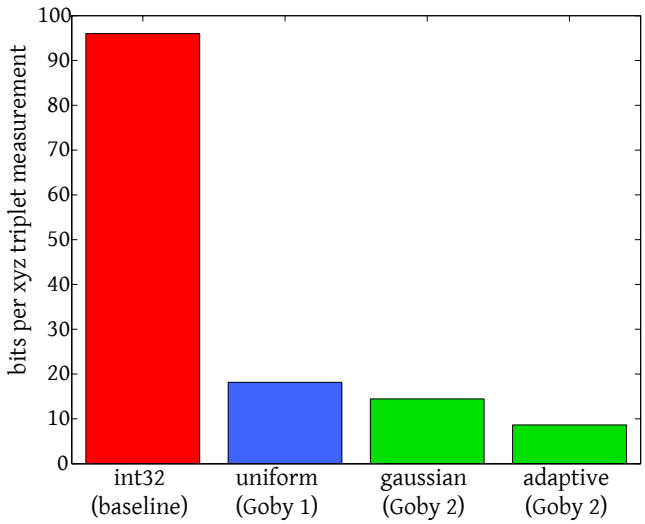

Fig. 3. Mean message size for 24420 hypothetical transmissions using actual AUV positions (meters in northings (Y), eastings (X), and negative depth (Z)) from the "Unicorn" vehicle in the GLINT10 Tyrrhenian sea shallow water experiment. The use (made possible in Goby2) of a fixed Gaussian model and an adaptive model in an arithmetic encoder reduces the mean message size by 20% and 52% respectively from the Goby1 "uniform" model and 85% and 91% from using standard 32-bit integers.

2.2 DCCL Encoding Library

The DCCL encoding library, which is written in C++, contains a set of basic coders for all the primitive and user-defined types supported by Google Protocol Buffers. These default coders use the same algorithms as DCCL1 (see Schneider and Schmidt (2012b)), which are basically arbitrary length numerics (integers and fixed-point numbers). For example, a 5-bit integer is used when the range $[20-52)$ is given for a field. Another way to think of these coders is that they are similar to an arithmetic encoder operating on a uniform probability distribution.

The new functionality for the Goby2 DCCL encoding library is a set of base classes for defining application specific coders (such as physics-based entropy coders) that can be loaded by the application at run time. The `_time` encoder/decoder ("codec") given in Fig. 2 is an example of such a special purpose coder. This coder assumes the message is received within one-half day from its transmission and thus only encodes seconds from start of day, saving 15 bits (46%) over using a standard unsigned 32-bit UNIX time since midnight UTC start of 1970. This is a primary example of the improved expansibility of Goby2 over Goby1 as mentioned in section 1.1. Users (message designers) understand the specifics of the data they are sending and can encapsulate this knowledge (if so desired) in a custom encoder.

An example of a user-defined codec is an implementation of an arithmetic codec for sending highly compressed position measurements of an AUV. Fig. 3 shows the bit sizes for messages using standard 32-bit integers (as a baseline), Goby 1 default encoders, and a non-adaptive and adaptive model using the arithmetic codec. For more details and results from this coder, see Schneider and Schmidt (2012a).

2.3 Further improvements

Besides the change from XML to Protobuf as the underlying language for the DCCL structure, a number of other advances were made for DCCL2 in light of feedback from collaborators and users, as well as over half a dozen field trials involving multiple AUVs. The first change was switching from a fixed six-byte header to a one- to two-byte minimum user-configurable header. The minimal header is used to identify the message type and can support up to $2^{15}-1$ (32767) types [1]. Even this part of the header can be reconfigured by the user by providing a custom coder that performs the same functionality. This allows for a smaller type database to be used (e.g. three bits allows eight types) for closed projects requiring a small number of messages, or for use on extremely small datagrams such as the WHOI Micro-Modem user mini-packet (13 bits, see Freitag et al. (2005)).

Another significant change was supporting variable-length coders. DCCL1 used fixed-length coders for all message fields. DCCL2 supports coders such as ones using a "presence bit" to omit uncommonly used fields. For example, a command message may have mutually exclusive fields, so it makes no sense to take up space in the message storing both fields. Another clear use for variable length fields includes entropy coders or lossy coders of various sorts. However, DCCL2 still mandates the generation of a field's maximum size. This allows the system designer to know definitively if a given message will always fit within a given MTU. Since it can be minutes or longer between the receipt of consecutive datagrams, it is often undesireable to fragment application level data. For this reason, Goby does not support any automatic fragmentation (e.g. such as that provided by the Transmission Control Protocol (TCP)).

2.4 Comparison to prior work

In the marine community, two contributions provide a similar framework to DCCL:

- REMUS Compact Control Language (CCL, see Stokey et al. (2005)): while CCL and derivatives are relatively widely used in the AUV community, it is inflexible and difficult to extend since the encoders are relatively ad-hoc and no mechanism is provided to automatically encode a newly designed message. Since DCCL2 adds the ability to add custom encoders, CCL can now be written as a proper subset of DCCL. This allows vehicles communicating using DCCL to optionally interoperate with vehicles using the older CCL language.
- Inter-Module Communication (IMC, see Martins et al. (2009)): uses XML with XSLT transformations into the native language code (e.g. C++) that gives a similar language-neutral data object (but with compile-time type safety) to DCCL2. However, IMC uses the standard system primitive types (e.g. 32-

[1] A variable integer coder is employed such that one byte is used for types 0-127 and two bytes otherwise

and 64-bit integers) and does not allow arbitrary bounding or user-defined codecs.

In the general computing regime, Abstract Syntax Notation One (ASN.1) and Advanced Message Queuing Protocol (AMQP) both provide complicated standards for defining data structures to be encoded via a standard set of rules. However, similarly to IMC, neither provides any mechanism for adding user-defined encoders.

Finally, while DCCL only uses Google Protocol Buffers (Protobuf) as a starting point for its language definition, Protobuf also provides a binary encoding. The encoding is reasonably compact, but does not take into account the origin of each field's data, which allows DCCL to provide a more compact encoding. The DCCL encoding of the message given in Fig. 2 is 56% more compact than the Protobuf built-in encoding because the DCCL message designer can incorporate information about the fields' physical origins (e.g. salinity is bounded between 25 and 40 in the world's oceans).

3. LINK LAYER INTERFACE: THE MODEMDRIVER MODULE

No standard exists for the interfaces to acoustic modems, and it is unlikely one will arise in the near future. Even with a common interface, modems will likely continue to have useful special features (such as navigation and ranging functionality) that exist outside realm of the core functionality of a modem, which is to send data. Thus, in order to make use of the rest of Goby (especially DCCL) on a variety of AUVs using different hardware, the link layer interface (ModemDriver) was written with a standard object-oriented design: a base class that provides an interface to the higher layers of Goby, and an increasing suite of derived classes to implement this interface for a specific piece of hardware. A beneficial side effect of this design is the ability to write drivers for a variety of non-acoustic links that have similar characteristics to acoustic links (low throughput), such as satellite communications or a faster-than-realtime underwater autonomy simulator, such as the MOOS-IvP "uField Toolbox" from Benjamin (2012).

The key to the success of this design is simplicity. A single function call initiates a transmission using the ModemTransmission class, which is typically a data telegram (`type == DATA`). Symmetrically, a single signal (again containing an instantiation of the ModemTransmission) is emitted upon asynchronous receipt of data. However, the ModemTransmission can be extended (even outside the Goby project) to support any number of additional data structures relating to an alternative type of transmission (e.g. long baseline (LBL) pings such as the WHOI Micro-Modem's `type == NARROWBAND_LBL`). If the application is aware that is using a specific piece of hardware, all these extensions become visible. If not, the application can still use the core functionality (data transmission). This design allows the system designer to choose abstraction when desirable or have full control of a modem's functionality as needed. See Fig. 4 for a diagram illustrating the core and extended functionality for several acoustic and other "slow" links.

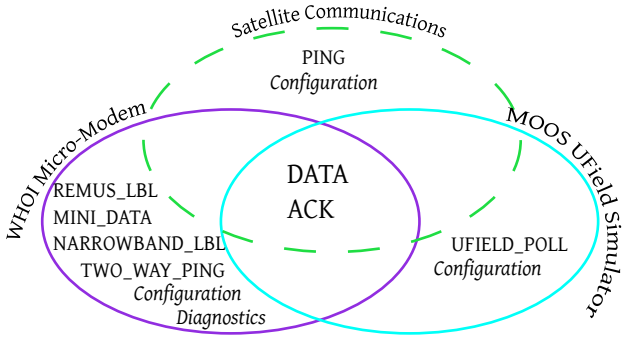

Fig. 4. Diagram of core ModemDriver functionality (center) versus extended modem-specific features for an acoustic modem (the WHOI Micro-Modem), a satellite link, and a faster-than-realtime virtual "modem" (the uField simulator).

This simplicity makes the task of writing a new driver easier. Rather than deal with a plethora of functionality (the superset of all supported modems would have been an alternative design), the new driver author must only deal with sending and receiving data. Once that is finalized, new functionality can be added as needed.

4. FIELD TRIALS

Since Goby2 is new, it has only been involved in three field trials (plus many hours of simulation and hardware-in-the-loop testing), compared to over a dozen or more that the authors are aware of that used Goby1:

- CAPTURE11 (August 2011): Chief scientist: C. Murphy (WHOI). Nodes: 2 OceanServer Iver2 AUVs (R. Eustice, University of Michigan), 1 WHOI SeaBED AUV (H. Singh, WHOI), 1 unmanned surface vehicle (F. Hover, MIT), and two research vessels, all equipped with an acoustic WHOI Micro-Modem. This experiment (using hardware and software assets from four different laboratories) successfully demonstrated multi-hop transmission of rich (e.g. imagery) datasets using C. Murphy's CAPTURE protocol. CAPTURE used Goby2 DCCL and ModemDriver, showing its extensibility in the hands of several other research groups that do not collaborate on a daily basis. The design and results of CAPTURE are detailed in Murphy (2012).
- Cyborg12 (May 2012): Chief scientist: A. Balasuriya (MIT). Nodes: 1 Bluefin 9" AUV, 1 WHOI Micro-Modem shallow water buoy, 1 research vessel, all equipped with an acoustic Micro-Modem. This trial was an engineering test in the process of developing a collaborative network of human experts and AUVs for mine countermeasures.
- Tiger12 (June 2012): Chief scientist: L. Freitag (WHOI). Nodes: 1 Tiger sonar mooring (prototype for an AUV), 1 Liquid Robotics WaveGlider, 1 research vessel. This cruise was a test of using Goby2 over a heterogenous mix of physical links for the purpose of sending control messages from the research vessel to the Tiger mooring and status messages in the opposite direction. The research vessel was out of acoustic range of the mooring, so the waveglider was used as a forwarding router between the acoustic

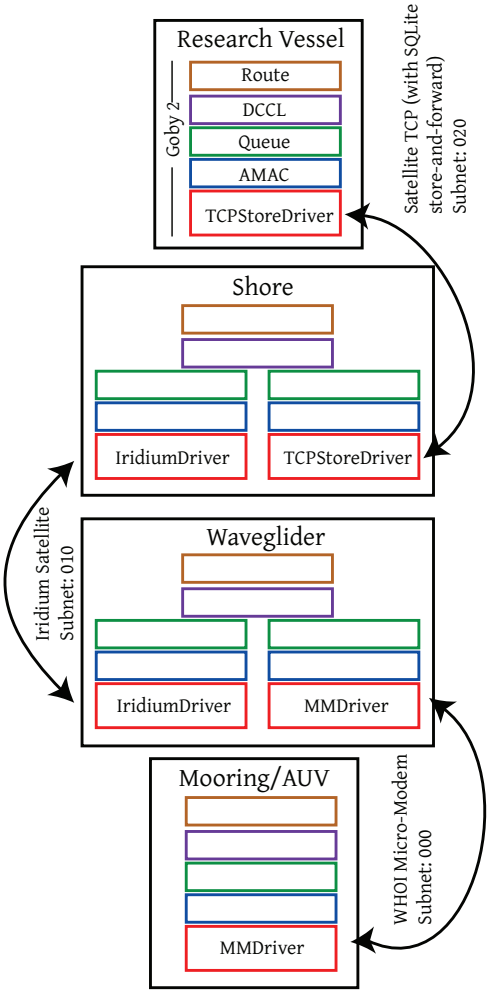

Fig. 5. Structure diagram of the Goby components for the Tiger12 cruise to form a seamless link from the Research vessel to the Tiger mooring (and intermediate nodes) over a collection of different link types (satellite-TCP, satellite-Iridium "call", acoustic Micro-Modem).

(sub-sea) and Iridium subnets. Since Iridium does not well support calls directly to mobile nodes, a shore station was used as a store-and-forward intermediate. All messages were logged here and retrieved at the next opportunity by the research vessel. This system provided reliable end-to-end transmission of DCCL messages for several days of continuous operation. While DCCL was designed initially for acoustic networks, it is equally useful for satellite links because of the similar characteristics (low throughput and very high cost per bit).

These trials have made us confident that we were successful in the primary design themes (see section 1.1) of third-party extensibility and field reliability. Furthermore, only one of these (Cyborg12) would have even been possible with Goby1.

The Goby project is open source (combination of GNU Public License (GPL) and Lesser GPL version 3) and is hosted at `https://launchpad.net/goby`. We welcome and greatly appreciate contributions of code, suggestions and bug reports.

ACKNOWLEDGEMENTS

We thank our collaborators at the Woods Hole Oceanographic Institution, especially the WHOI Micro-Modem group for their modem and Chris Murphy in the Deep Submergence Lab for essential critical feedback. We also thank the open source software community that provides many of the tools without which our work would not be possible. Finally, we extend our gratitude to the past and future contributors to the Goby project.

REFERENCES

Benjamin, M.R. (2012). The MOOS-IvP uField Toolbox for Multi-Vehicle Operations and Simulation. Technical Report 12.2, Massachusetts Institute of Technology.

Bradley, A., Feezor, M., Singh, H., and Yates Sorrell, F. (2001). Power systems for autonomous underwater vehicles. *Oceanic Engineering, IEEE Journal of*, 26(4), 526–538.

Chitre, M., Shahabudeen, S., and Stojanovic, M. (2008). Underwater acoustic communications and networking: Recent advances and future challenges. *The State of Technology in 2008*, 42(1), 103–114.

Freitag, L., Grund, M., Singh, S., Partan, J., Koski, P., and Ball, K. (2005). The WHOI Micro-Modem: an acoustic communications and navigation system for multiple platforms. In *IEEE Oceans Conference*.

Google (2012). Protocol Buffers - Google Code. URL `http://code.google.com/apis/protocolbuffers/`.

Martins, R., Dias, P., Marques, E., Pinto, J., Sousa, J., and Pereira, F. (2009). Imc: A communication protocol for networked vehicles and sensors. In *OCEANS 2009-EUROPE*, 1–6. IEEE.

Murphy, C. (2012). *Progressively communicating rich telemetry from autonomous underwater vehicles via relays*. Ph.D. thesis, Massachusetts Institute of Technology and Woods Hole Oceanographic Institution.

Petillo, S., Schmidt, H., and Balasuriya, A. (2012). Constructing a distributed auv network for underwater plume-tracking operations. *International Journal of Distributed Sensor Networks*, 2012.

Preisig, J. (2007). Acoustic propagation considerations for underwater acoustic communications network development. *ACM SIGMOBILE Mobile Computing and Communications Review*, 11(4), 2–10.

Schneider, T. and Schmidt, H. (2012a). Approaches to improving acoustic communications on autonomous mobile marine platforms. In *UComms 2012 Sestri Levante, Italy*.

Schneider, T. and Schmidt, H. (2012b). Goby-Acomms: A modular acoustic networking framework for short-range marine vehicle communications. URL `http://gobysoft.com/dl/goby-acomms1.pdf`.

Shafer, A. (2008). *Autonomous cooperation of heterogeneous platforms for sea-based search tasks*. Master's thesis, Massachusetts Institute of Technology.

Stokey, R.P., Freitag, L.E., and Grund, M.D. (2005). A compact control language for AUV acoustic communication. *Oceans 2005-Europe*, 2, 11331137.

Zimmermann, H. (2002). OSI reference model–The ISO model of architecture for open systems interconnection. *Communications, IEEE Transactions on*, 28(4), 425–432.