ROS Message Transport over Underwater Acoustic Links with ros_acomms

Eric Gallimore
Applied Ocean Physics and Engineering
Woods Hole Oceanographic Institution
Woods Hole, USA
egallimore@whoi.edu

Dennis Giaya
Applied Ocean Physics and Engineering
Woods Hole Oceanographic Institution
Woods Hole, USA
dgiaya@whoi.edu

Brennan Miller-Klugman
Applied Ocean Physics and Engineering
Woods Hole Oceanographic Institution
Woods Hole, USA
bmillerklugman@whoi.edu

Caileigh Fitzgerald
Applied Ocean Physics and Engineering
Woods Hole Oceanographic Institution
Woods Hole, USA
cfitzgerald@whoi.edu

Kayleah Griffen
Applied Ocean Physics and Engineering
Woods Hole Oceanographic Institution
Woods Hole, USA
kgriffen@whoi.edu

Laura Lindzey
Ocean Engineering Department
University of Washington
Applied Physics Laboratory
Seattle, USA
lindzey@uw.edu

Lee Freitag
Applied Ocean Physics and Engineering
Woods Hole Oceanographic Institution
Woods Hole, USA
lfreitag@whoi.edu

***Abstract*— A software framework, "ros_acomms," has been developed to enable transport of ROS messages and other data across low-throughput and high-latency underwater acoustic links. Messages are efficiently marshalled using user-provided configuration data, if available, or automatically via message introspection. A modular set of modem drivers, media-access-control engines, and message queues transport messages from one system to another via a modem. It supports message fragmentation, positive acknowledgment, and custody-transfer routing. It also includes an acoustic link simulator that uses a ray-tracing model to estimate link performance and latency. While it targets the WHOI Micromodem family of acoustic modems, the modular modem driver implementation has been leveraged to support low-throughput Iridium satellite links and other acoustic modems. It has been tested and used operationally at sea for remote redirection of autonomous underwater vehicles while providing operators with near real time vehicle telemetry and sensor data.**

Keywords—underwater acoustic communication, ROS, codec

I. Introduction

Ros_acomms provides a complete framework for efficiently transferring messages across underwater acoustic links. Robot Operating System (ROS) [1] messages can be sent with minimal configuration, and the included encoder/decoder (codec) engine will use message introspection to determine how field values should be encoded. All ROS message primitive types are supported, including arrays and nested ROS messages. Users may optionally provide additional information about the fields in a message (such as the range of expected values or desired resolution) via a configuration file, which is used to compress and bit-pack data more efficiently.

Reliable transport (via positive acknowledgement and retransmission) and large message fragmentation are provided. This allows sending files and messages larger than the maximum acoustic packet size.

MOOS messages [2] may also be transferred via a companion package, the moos_bridge, and the message coding scheme can also be used to encode arbitrary Python dictionaries.

The framework is designed to co-exist with other acoustic traffic, including interoperating with older standards such as the Compact Control Language (CCL).

It is designed against the WHOI Micromodem acoustic modems [3], [4], although the modular modem driver implementation supports integrating other modems. The system has also been extended to support low-throughput Iridium satellite communication links. There is a companion package, ros_iridium, that uses the ros_acomms message queuing and marshalling system to transfer messages via the Iridium satellite network.

In addition to supporting the WHOI Micromodem, it includes an acoustic communication simulator to facilitate software-only development and testing.

It is released under open source (LGPL/AGPL) licenses.

II. Background

A. Message Marshalling

Underwater acoustic communication links are inherently low-throughput and high-latency when compared with the radio links commonly used with terrestrial robots [5]. Thus, care is required when using underwater acoustic communication for robot telemetry, command, and control. Often, data must be compressed prior to transmission to communicate effectively.

Many commonly-used data marshalling protocols are not optimized for low-throughput links. Formats such as JSON [6],

This work was funded in part by the National Oceanographic and Atmospheric Administration, the Office of Naval Research, the National Science Foundation, and the Woods Hole Oceanographic Institution

MessagePack [7], and Concise Binary Object Representation (CBOR) [8] provide fully-descriptive encoded forms: message content can be extracted without knowing the schema used to encode it. While this feature offers obvious advantages, it requires embedding message type information within the message, which increases the amount of data that must be transmitted. This overhead is undesirable and generally unacceptable when using very-low-throughput links.

Formats such as Google Protocol Buffers are not fully descriptive and provide a mixed optimization for size and encoding/decoding speed [9], but encoding/decoding speed is not typically a limiting factor when transferring data via acoustic links.

Several marshalling systems targeted at robotics applications improve upon these schemes, but still carry unacceptable overhead due to the use of header information and failing to bit-pack field data. The ROS1 ros_comm messaging scheme is simple, not fully descriptive, and relatively compact. However, each non-array field type has a fixed bit length of at least one byte, and each message includes header information. ROS2 uses DDSI-RTPS, which also includes header information and byte-aligned, fixed-length fields [10]. Lightweight Communications and Marshalling (LCM) includes a marshalling scheme that is relatively efficient, but it prepends a 64-bit fingerprint to each message to identify its type. It is also optimized for speed, and uses byte-aligned fields [11].

Several marshalling schemes have been developed specifically for transporting data over acoustic links. One early effort at standardization was the Compact Control Language (CCL) [12], which is still widely used with the REMUS family of AUVs. CCL was designed to match the contemporary capabilities of the WHOI Micromodem-1 and its predecessor, the Utility Acoustic Modem (UAM) [13]. As a result, all messages are exactly 32 bytes in length, which corresponds with the length of a legacy Micromodem FSK packet. All message types are defined in a specification document [14], and specific field encoder/decoders are provided as C functions. CCL is often coupled with a basic call-and-response query system to request messages from a remote node, or a periodic transmission cycle.

A more recent development in this area is the Dynamic Compact Control Language (DCCL), which provides a runtime-configurable mechanism for marshalling messages based on extensions to Google Protobuf message definitions [15], [16]. This system does not require fixed message sizes. It is often coupled with Goby, a middleware library that provides modem interfaces and networking functions [17]. DCCL formed much of the inspiration for the codec system described here.

The implementations of both CCL and DCCL assume that all message types are registered with a central authority. In practice, many developers define new message types and either use reserved private message identifiers that are set aside by the registration authority or simply use otherwise-unused identifiers with the assumption that their systems won't have to interoperate. Additionally, many acoustic modem users do not use any of these standardized message marshalling schemes and instead define their own.

Ros_acomms specifically addresses this issue by allowing different sets of message encodings to be used for communication with different remote modems. Additionally, there is a CCL-compatible codec included in the ros_acomms suite, so that CCL messages can be sent and received. The codec system was designed with DCCL compatibility in mind, and early versions included some rudimentary DCCL interoperability, but this has not been maintained due to a lack of current applications.

B. *Message queueing and quality of service*

On practical communication links, it is often not possible to instantly transfer all data required by an application. To address this limitation, which is particularly concerning when using underwater acoustic links that are low-throughput and high-latency, ros_acomms implements a message queuing system that supports configurable-length queues with runtime-adjustable message prioritization.

Queues allow messages to be sent in first-in-first-out (FIFO) order or last-in-first-out (LIFO) order. In many robotic applications, such as sending vehicle status or pose information, newer messages render older ones obsolete, and the LIFO queue ensures that unnecessary older data aren't sent.

Message prioritization can ensure that more important data are transferred sooner. Ros_acomms allows users to specify a default priority for each topic being sent via the configuration file, and these priorities can be changed at runtime via acoustic messages.

C. *Fragmentation, Reliable Transport, and File Transfer*

Communication links typically have a maximum transmissible unit (MTU), which is the largest amount of data that can be sent in a single unit such as a frame or packet. To send messages or data larger than the MTU on a given link, data must be fragmented at the transmitter and reconstructed at the receiver.

Many existing fragmentation protocols are not optimized for the limited channel reliability, high latency, and low throughput associated with underwater acoustic links. The Transmission Control Protocol (TCP), for example, is commonly used in conjunction with the Internet Protocol (IP) and provides end-to-end reliable and ordered transport of a stream of bytes. TCP specifies data headers that are minimally 20 bytes long per datagram if no options fields are utilized[18]; this is an acceptable amount of overhead for an IP datagram that may contain up to 65,535 bytes [19]. In the context of an acoustic link where the maximum frame size is typically between 32 and 256 bytes and the maximum packet size is typically less than a few kilobytes, 20 bytes of header per frame

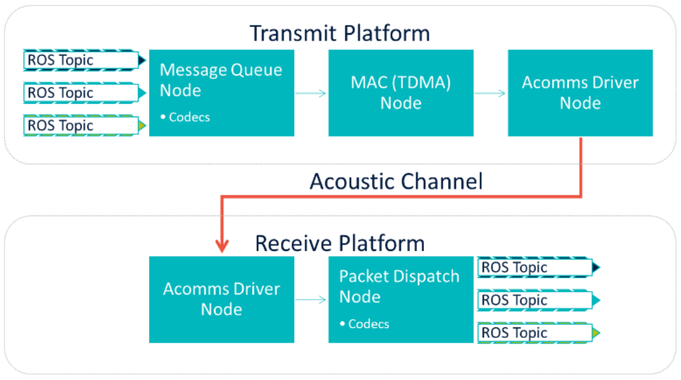

Figure 1: Message flow within ros_acomms

is a significant cost to the total data throughput. The three-way handshake to establish a TCP connection is another feature that penalizes high-latency and unreliable links. In the event that communication opportunities are sporadic and only allow time for a single transmission, it becomes critical to include data in the initial transmission rather than dedicating it to the specific sync sequence.

The specification for a Delay-Tolerant Networking Architecture (DTN) addresses many of the shortcomings that TCP and other Internet protocols exhibit when operating on unreliable or high latency links [20]. The ros_acomms fragmentation protocol draws inspiration from features of DTN such as performing custody transfers of bundles between nodes in the network, delivering messages in order of priority, and selectable delivery options in order to cater to the circumstances of a particular acoustic environment. However, DTN is not optimized for low-throughput links such as underwater acoustic links.

D. *Acoustic modem features*

Ros_acomms supports modern acoustic packet protocols available on the WHOI Micromodem-2, in addition to legacy acoustic packets also supported by the Micromodem-1. Ros_acomms effectively leverages the Flexible Data Packet system on the Micromodem-2, which allows sending a single acoustic packet that uses multiple error correction coding levels to include both high-reliability/low-throughput data and lower-reliability/higher throughput data. This is commonly used to allow a single packet to include critical information, such as AUV position and status, in the high-reliability section, and science or image data in the high-data-rate section.

Acoustic timing and ranging features of the Micromodem are also supported, allowing ros_acomms to integrate with acoustic navigation systems.

III. Architecture

A. *Message transport flow*

A primary goal of the project is to support “transparent”, bi-directional transport of ROS messages published on one platform, such as an autonomous underwater vehicle, and re-publishing them on a different platform, such as different vehicle or operator computer (and vice-vera). Configuration of the system is done via rosparams, typically as a YAML file containing a list of ROS topics to transport along with instructions for how to encode those messages, as shown in Listing 1.

```yaml
- codec_name: default
  match_dest: [100]
  except_src: [1, 5]
  packet_codec: ros
  miniframe_header: [0x12]
  message_codecs:
    - id: 1
      message_codec: default
      subscribe_topic: "/status"
      publish_topic: "/from_acomms/status"
      ros_type: "my_auv_pkg/Status"
      default_dest: 99
      queue_order: lifo
      queue_maxsize: 1
      priority: 100
      fields:
        header:
          codec: msg
          ros_type: "std_msgs/Header"
          fields:
            stamp:
              codec: rostime
        latitude:
          codec: float
          min_value: -90
          max_value: 90
          precision: 5
        longitude:
          codec: float
          min_value: -180
          max_value: 180
          precision: 5
       amperage:
          codec: linspace
          min_value: 0
          max_value: 10
          resolution: 0.2
```

Listing 1: Example ros_acomms codec and queue configuration.

Messages are transferred as described below and in Figure 1. On the transmitting system:

1) A ROS message is published on a topic

2) The message queue node is subscribed to this topic.

3) When a new ROS message is received, the message queue node uses a message codec to convert the data in the message to a compact representation by using field codecs to encode each message field. This is done recursively for nested ROS messages.

4) The media access control (MAC) node, typically a time-division-multiple-access (TDMA) MAC, queries the message queue node to retreive the highest priority message(s).

5) The message queue node identifies the highest priority message and uses an appropriate packet codec to pack as many messages as will fit into one modem packet for transport.

6) The MAC node queues the modem packet for transmit using the modem driver (the acomms driver node), which interfaces to the modem hardware to send the packet acoustically. (Alternately, when using the acoustic modem simulation built into ros_acomms, the acoustic communcations simulator node replaces the modem driver node.)

On the receiving system:

7) A packet is received by the acoustic modem.

8) The acomms driver node handles the incoming packet and publishes a ReceivedPacket message.

9) The packet dispatch node subscribes to these messages.

10) The packet dispatch node evaluates metadata on the packet message (such as the acoustic modem source address and optional header bytes) and determines which packet codec should be used to decode the packet.

11) The packet codec uses one or more message codecs to decode the received data into ROS message(s).

12) The packet dispatch node publishes the message on a ROS topic.

An arbitrary number of systems are supported, each of which can both send and receive messages.

Custody-transfer routing can be performed by sending messages to an intermediate platform, which then sends those messages on to a subsequent platform. This can be used to send messages over different links, such as an acoustic channel operating at a different frequency or a satellite link.

B. Message Marshalling and Codecs

Ros_acomms uses a layered set of encoders/decoders (codecs) that operate at the field, message, and packet level. Each message consists of multiple named fields, each of which contains a datum. Each field may contain a primitive type, such as an integer or string, or nested message type. Packets correspond to the atomic unit of data that can be sent by a modem at one time. A packet may contain a single message, multiple messages, or part of a fragmented message.

The codecs are implemented using base classes such that novel field codecs, message codecs, and packet codecs may be developed. For example, packet codecs can be written to interoperate with other standards, such as CCL.

The selection of which field codecs to use on a message is determined via a configuration file, as shown in Listing 1. If no field codecs are specified for a message in the configuration, the message codec engine will introspect the ROS message being sent and use appropriate field codecs for each field in the message. Even when using introspection, the resulting message size is smaller than the default ROS message serialization, largely because ROS message serialization adds headers and byte aligns all fields, including booleans and 7-bit ASCII string characters.

Each field in the message can be assigned a codec with parameters, which improves the efficiency of message compression. Both lossy and lossless codecs are available for most ROS message field types. For example, floating point numbers may be encoded as fixed-decimal values or linearly-spaced ranges. Strings are encoded by default with bit-packed 7-bit ASCII, but users may select a 6-bit ASCII encoding that includes only uppercase letters and fewer special and punctuation characters. Similar options exist for most field codecs.

C. Fragmentation, Reliable Transport, and File Transfer

The ros_acomms fragmentation protocol ensures reliable transport of ROS messages that, when encoded, are larger than the MTU of the link, which can vary. The concept for transfers models DTN; a ROS message is analogous to a bundle in DTN. The message can be fragmented to a variable block size to accommodate varying quality links and the queued fragments are transmitted in order of priority to reachable link destinations. It also borrows an acknowledgement scheme similar to TCP Selective Acknowledgement to minimize the amount of data that must be re-transmitted when the receiver ends up with discontinuous fragments of the original message.

Recognizing the bandwidth constraints of underwater acoustic links, the protocol is designed to minimize the required size of fragment headers, as described in Tables 1-3. Transfers are organized into sessions and utilize a 14-byte session start header, a 5-byte session continuation header, and an acknowledgement header that is minimally 6 bytes, but varies based on the number of discontinuous block segments being acknowledged. In practice, each header requires an additional byte prepended: the ros_acomms message identifier.

TABLE I. SESSION START HEADER

Field	*Size*
Session ID	8-bits
Session Source ID	8-bits
Session Dest. ID	8-bits
Timestamp	32-bits
Block size (bits)	8-bits
Message size (blocks)	16-bits
First block index	16-bits
Last block index	16-bits

TABLE II. SESSION CONTINUATION HEADER

Field	*Size*
Session ID	8-bits
First block index	16-bits
Last block index	16-bits

TABLE III. SESSION ACKNOWLEDGEMENT HEADER

Field	*Size*
Ack. Flag	1-bit
N_Blocks	7-bits
Session ID	8-bits
Block start index[0]	16-bits
Block end index[0]	16-bits
block start index[n]	16-bits
Block end index [n]	16-bits

Each session is uniquely identified through the 8-bit session ID combined with the source ID. In this manner, each individual source is able to keep track of open sessions and avoid re-using a session ID that has not been acknowledged. To avoid the penalty of long two-way-travel-times, a sender may include data up to the full size of a packet with the session start header. Additionally, the sender may proceed to send packets continuing the session before waiting to receive acknowledgement if so configured by the medium access controller node.

Acknowledgement headers comprise a list of received blocks as described by the start and end index for each contiguous section in multiples of the block size. The ack flag is set to describe blocks as positively acknowledged, or "ACKED", and it may be cleared to indicate that the list will describe only the blocks that are still outstanding, or "NACKED". The decision is made to send whichever results in the shorter of the two lists. Acknowledgement containing the zeroth block of a message indicates acknowledgement of the session start header back to the sender. Specifically sending a null acknowledgement for the zeroth block indicates to the sender that one or more session continuation packets were received for which the receiver does not have the full session start header information and will trigger a retransmit from the sender.

D. Software components and libraries

Ros_acomms is designed using modular ROS nodes that allow individual functions to be developed in a research setting without requiring changes to the rest of the system. For example, different media access control methods can be implemented by replacing only the MAC node.

Additional modems can be supported by any node implementing the modem driver interface, which has been used to develop drivers for Sonardyne acoustic modems and Iridium satellite modems. This interface also allows the acoustic communication simulator to be used in place of actual acoustic modems.

Ros_acomms incorporates several separately packaged libraries. Pyacomms, an interface library for the WHOI Micromodem, is used by the modem driver node to communicate with the modem. The message and field codecs are packaged as "ltcodecs" (for "Low Throughput Codecs"). These libraries are maintained by the WHOI Acoustic Communications Group and published on PyPi.

To operate over Iridium satellite links, a separate package, ros_iridium, provides a MAC node and modem drivers for Iridium modems and the RUDICS shore-side server. The message queues and codecs from ros_acomms are used without modification.

IV. Simulation

The base ros_acomms package includes a simple acoustic communication simulator. It uses the Bellhop raytracing model [21] to determine path transmission loss between two simulated modems. The effect of acoustic collisions (interference from other modems transmitting at the same time) is modeled. Configurable, time-varying noise is added, and the signal margin at the receiver is calculated. This is then used with an empirically-derived communication performance model to simulate channel performance, including both latency and packet success or failure.

V. Deployments

To date, ros_acomms has been deployed on a variety of oceanographic platforms, including REMUS AUVs, WaveGlider ASVs, DSV Sentry, acoustic gateway buoys, shore-side systems, and shipboard operator consoles. It has been used operationally to support research and scientific objectives.

A more detailed discussion of its use in the NOAA CoExploration program may be found in [22].

VI. Future Development

Ros_acomms is being used to perform underwater acoustic networking research, and as this matures, additional networking capabilities will be incorporated into the package.

A ROS Actions proxy is under development to facilitate remote command execution. It builds upon the fragmentation and acknowledgment system to provide asynchronous progress notifications as messages are passed between systems.

Acknowledgment

The authors wish to thank Dr. Ian Vaughn and Dr. Toby Schneider for fruitful conversations that informed the development of this system. They thank Isaac Vandor for his contributions to ros_acomms and his helpful suggestions for architectural improvements.

References

[1] M. Quigley *et al.*, "ROS: an open-source Robot Operating System," in *ICRA workshop on open source software*, 2009, vol. 3, p. 5.
[2] Oxford Mobile Robotics Group, "MOOS : Main - Introduction." https://www.robots.ox.ac.uk/~mobile/MOOS/wiki/pmwiki.php/Main/Introduction (accessed May 27, 2022).
[3] E. Gallimore, J. Partan, I. Vaughn, S. Singh, J. Shusta, and L. Freitag, "The WHOI micromodem-2: A scalable system for acoustic communications and networking," in *OCEANS 2010*, Sep. 2010, pp. 1–7. doi: 10.1109/OCEANS.2010.5664354.
[4] L. Freitag, M. Grund, S. Singh, J. Partan, P. Koski, and K. Ball, "The WHOI micro-modem: an acoustic communications and navigation system for multiple platforms," Washington, DC, USA, Sep. 2005.
[5] M. Stojanovic, "On the Relationship Between Capacity and Distance in an Underwater Acoustic Communication Channel," in *Proceedings of the 1st ACM International Workshop on Underwater Networks*, New York, NY, USA, 2006, pp. 41–47. doi: 10.1145/1161039.1161049.
[6] P. Charollais, "ECMA-404, 2nd edition, December 2017," p. 16, 2017.
[7] "MessagePack." MessagePack, Apr. 28, 2022. Accessed: Apr. 28, 2022. [Online]. Available: https://github.com/msgpack/msgpack/blob/8aa09e2a6a9180a49fc62ecfefe149f063cc5e4b/spec.md
[8] C. Bormann and P. E. Hoffman, "Concise Binary Object Representation (CBOR)," Internet Engineering Task Force, Request for Comments RFC 8949, Dec. 2020. doi: 10.17487/RFC8949.
[9] "Overview | Protocol Buffers | Google Developers." https://developers.google.com/protocol-buffers/docs/overview (accessed Apr. 28, 2022).
[10] Object Management Group, "The Real-time Publish-Subscribe Protocol DDS Interoperability Wire Protocol (DDSI-RTPSTM) Specification Version 2.5," OMG Standard ptc/2021-03-03, Mar. 2021. Accessed: Apr. 28, 2022. [Online]. Available: https://www.omg.org/spec/DDSI-RTPS/2.5/PDF

[11] A. S. Huang, E. Olson, and D. C. Moore, “LCM: Lightweight Communications and Marshalling,” in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Taipei, Oct. 2010, pp. 4057–4062. doi: 10.1109/IROS.2010.5649358.

[12] R. P. Stokey, L. E. Freitag, and M. D. Grund, “A Compact Control Language for AUV acoustic communication,” in *Oceans 2005 - Europe*, Jun. 2005, vol. 2, pp. 1133-1137 Vol. 2. doi: 10.1109/OCEANSE.2005.1513217.

[13] L. Freitag, M. Johnson, M. Grund, S. Singh, and J. Preisig, “Integrated acoustic communication and navigation for multiple UUVs,” in *MTS/IEEE Conference and Exhibition OCEANS, 2001*, 2001, vol. 4, pp. 2065–2070 vol.4. doi: 10.1109/OCEANS.2001.968315.

[14] Roger Stokey, “A Compact Control Language for Autonomous Underwater Vehicles,” Woods Hole Oceanographic Institution, Technical Report, Apr. 2005.

[15] T. Schneider and H. Schmidt, “The Dynamic Compact Control Language: A compact marshalling scheme for acoustic communications,” in *OCEANS 2010 IEEE - Sydney*, May 2010, pp. 1–10. doi: 10.1109/OCEANSSYD.2010.5603520.

[16] T. Schneider, S. Petillo, H. Schmidt, and C. Murphy, “The Dynamic Compact Control Language version 3,” in *OCEANS 2015 - Genova*, May 2015, pp. 1–7. doi: 10.1109/OCEANS-Genova.2015.7271608.

[17] “WhatIsGoby · GobySoft/goby Wiki,” *GitHub*. https://github.com/GobySoft/goby (accessed Apr. 28, 2022).

[18] “Transmission Control Protocol,” Internet Engineering Task Force, Request for Comments RFC 793, Sep. 1981. doi: 10.17487/RFC0793.

[19] “Internet Protocol,” Internet Engineering Task Force, Request for Comments RFC 791, Sep. 1981. doi: 10.17487/RFC0791.

[20] L. Torgerson *et al.*, “Delay-Tolerant Networking Architecture,” Internet Engineering Task Force, Request for Comments RFC 4838, Apr. 2007. doi: 10.17487/RFC4838.

[21] Michael B. Porter and Yong-Chun Liu, “Finite-Element Ray Tracing,” Mystic Hilton, USA, Oct. 1994.

[22] Laura Lindzey, Isaac Vandor, Toby Schneider, Carl Kaiser, and Michael Jakuba, “CoExploration for Adaptive AUV Survey,” Singapore, Sep. 2022.

.