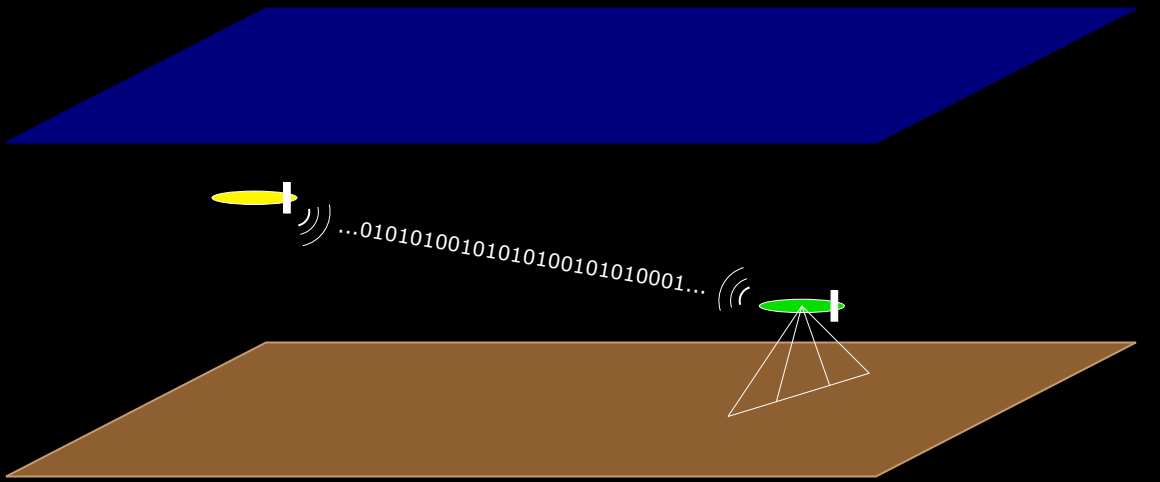
pAcommsHandler:
Acoustic networking in MOOS with the WHOI Micro-Modem



Toby Schneider

MIT/WHOI Joint Program

Henrik Schmidt

MIT Laboratory for Autonomous Marine Sensing Systems

Massachusetts Institute of Technology

WOODS HOLE OCEANOGRAPHIC INSTITUTION
1930

Problem

Given:

1. Acoustic communications are highly rate-limited

(as low as 32 bytes / minute).

2. Marine robots are increasingly used in clusters.

3. Research at sea demands robust & reconfigurable software.

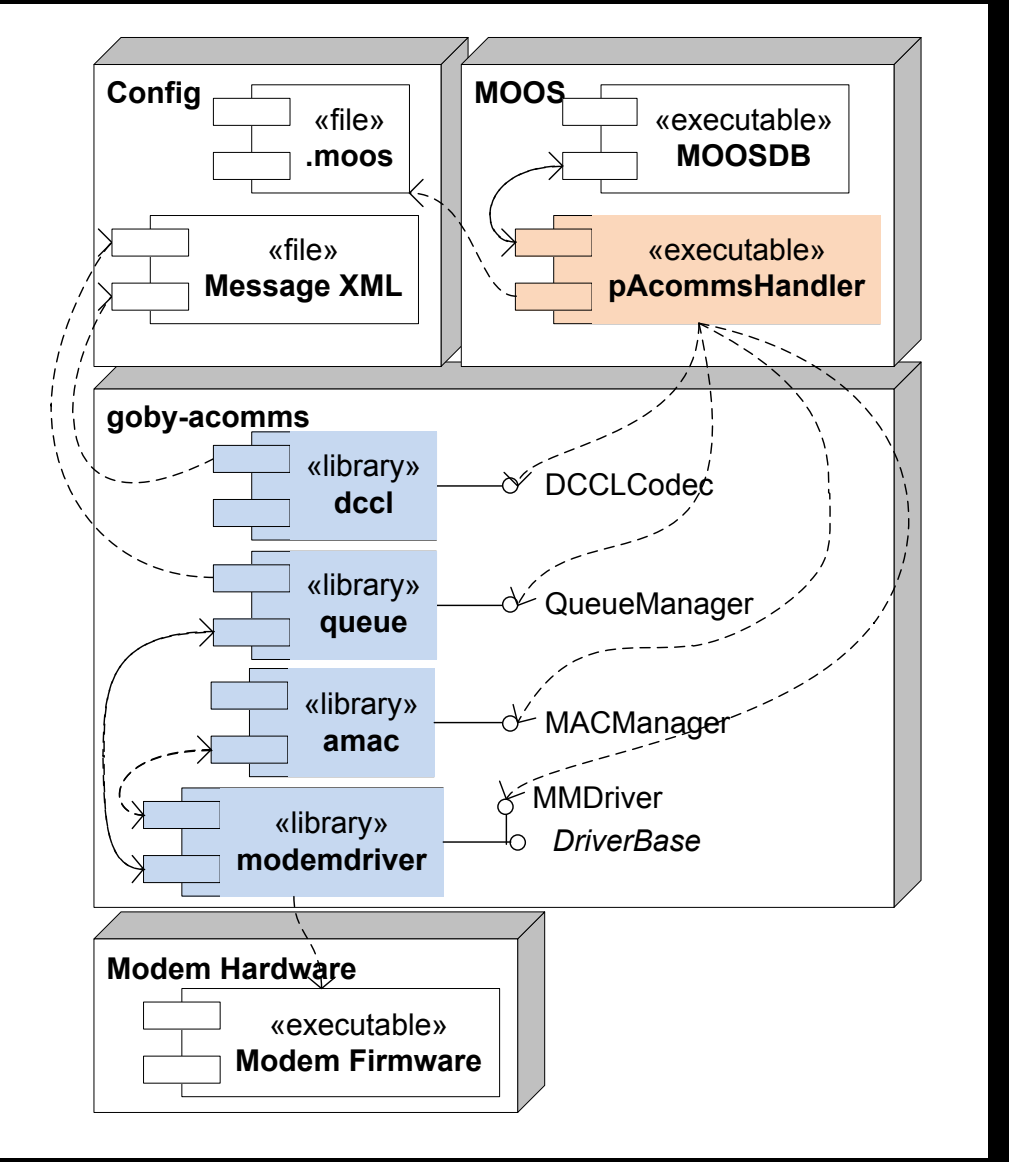
How can we design a networking framework that satisfies these constraints?

Modular design with emphasis on efficiency instead of abstraction.

Comparison with OSI Layers

OSI Layer	Module
Application	pAcommsHandler
Presentation	libdccl: Encoding and decoding
Session	N/A (unsolicited datagrams)
Transport	libqueue: Priority based message queuing
Network	currently N/A (active research area)
Data Link	libmodemdriver: Modem driver libamac: Medium Access Control (MAC)
Physical	WHOI Micro-Modem

Structure Diagram



libdccl: What is DCCL?

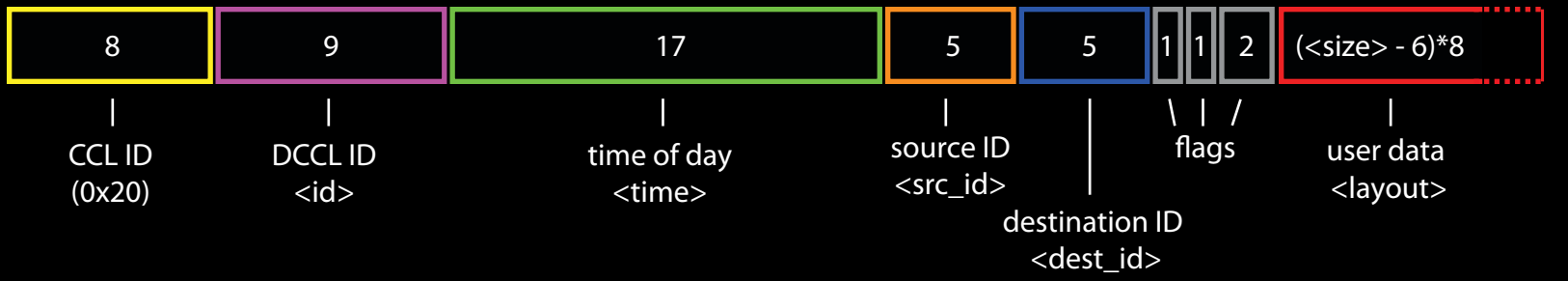
DCCL is the Dynamic Compact Control Language which is:

1. A structure language for fixed size small messages:
    - O(10-100) bytes
    - suitable for acoustic modems
      (e.g. WHOI Micro-Modem)

libdccl: What is DCCL?

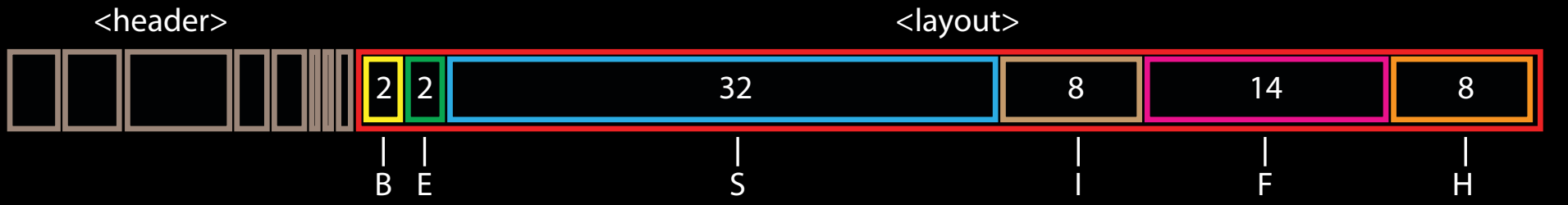
1. A structure language for fixed size small messages.
2. A set of predefined encoding & decoding algorithms (*libdccl*):
   - flexible: runtime translation of XML to C++ objects
   - efficient: bounded numbers & unaligned packing
   - simple: encoding based on unsigned integers
   - customizable: user-supplied pre-encode / post-decode callbacks
   - secure: provides (optional) AES encryption

DCCL Structure: Header




```
<?xml version="1.0" encoding="UTF-8"?>
<message_set>
  <message>
    <name>Example</name>
    <size>32</size>
    <id>1</id>
    <header>
      <src_id>
        <name>Src</name>
      </src_id>
      <dest_id>
        <name>Dest</name>
      </dest_id>
      <time>
        <name>Time</name>
      </time>
    </header>
    <layout>
      ...
    </layout>
  </message>
</message_set>
```

DCCL Structure: Layout




```
<?xml version="1.0" encoding="UTF-8"?>
<message_set>
  <message>
    <name>Example</name>
    <size>32</size>
    <id>1</id>
    <header>
      ...
    </header>
    <layout>
      <bool>
        <name>B</name>
      </bool>
      <enum>
        <name>E</name>
        <value>cat</value>
        <value>dog</value>
        <value>mouse</value>
      </enum>
      <string>
        <name>S</name>
        <max_length>4</max_length>
      </string>
      <int>
        <name>I</name>
        <max>100</max>
        <min>-50</min>
      </int>
      <float>
        <name>F</name>
        <max>100</max>
        <min>-50</min>
        <precision>2</precision>
      </float>
      <hex>
        <name>H</name>
        <num_bytes>1</num_bytes>
      </hex>
    </layout>
  </message>
</message_set>
```

libdccl Encoder: Algorithms

All types have “not given” or “invalid” == 0

- <bool>: true is 2; false is 1
- <float>:

    subtract <min>
    multiply by $10^{<precision>}$
    round to nearest integer
    size defined by <max>, <min>, & <precision>

  example:

    precision = 1, min = 2, max = 3
    encode 2.5 as (2.5-2)*10 = 5
    need field size: ceil($\log_2((3-2)*10)$) = 4

libdccl Encoder: Algorithms

- <int>: same as <float> with precision = 0
- <hex>: send as is.
- <string>: ASCII octets; no '\0' needed.
- <enum>: map values to integers 1,2,N.

Header fields: fixed size (always).

Time sent as seconds since start of day.

Integrating DCCL with MOOS

DCCL provides additional features for integrating with publish/subscribe architectures (such as MOOS)

1. Source variables: MOOS variable to extract data from for a given message field. Supports string parsing of

key1=value2,key2=value2,key3={value03,value13,value23}

2. Triggers: publish event or time based trigger to create message.

3. Publish variables: MOOS variables to publish data upon message receipt. Supports a printf-type syntax.

libqueue: Priority buffering

Problem: Desired data throughput exceeds capacity (even with intelligent encoding)

Solution: Prioritize data based on

1. overall value
2. time-sensitivity

libqueue: Algorithm

Each DCCL message type has its own queue.

Each queue has:

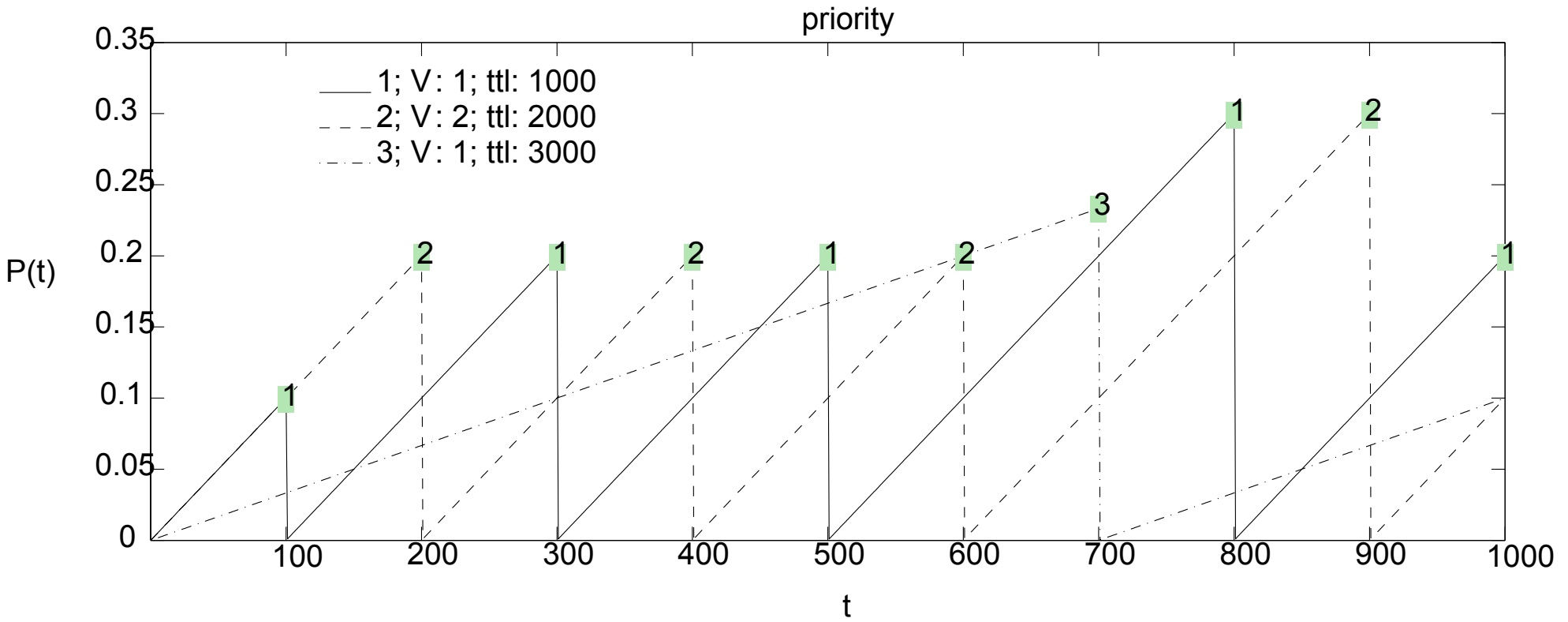
1. base value [V_{base}]
2. time-to-live [ttl]
3. time when queue was last used (t_{last})

The priority [P(t)] at any given time [t] is:

$$P(t) = V_{base} * (t - t_{last}) / ttl$$

libqueue: Example

$$P(t) = V_{base} * (t - t_{last}) / ttl$$



libqueue: Additional features

Configured using same XML files as DCCL messages in <queuing> block.

Transparent stitching of DCCL messages to form larger data frames.

- e.g. 2x 16 bytes DCCL messages -> 1x 32 byte message.
- broadcast messages can be “piggybacked” on directed messages

Traditional CCL messages can also be queued and sent.

Acknowledgments handled.

libamac: TDMA MAC

Three modes of operation

- Centralized (“polling”)
  - Network master initiates all transactions
  - Synchronous clocks unnessary
  - Reconfigurable during operations
- Decentralized
  - Time slots preassigned at launch
  - Not reconfigurable during operations
- Decentralized (auto-discovery of peers)
  - Quiet slot inserted to allow discovery of new peers
  - Quiet slot pseudorandomly reassigned to improve discovery

libmodemdriver: Driver

Superclass *DriverBase* provides generic acoustic modem interface.

Subclass *MMDriver* provided for the WHOI Micro-Modem (AUV 0.92.0.85 and newer).

Other modems could be supported by new subclasses.

Features:

- Separate thread handles serial communications
- Abstracted serial interface handles serial over TCP (as client or server)
- Robust design requires NMEA acknowledgment of every command ($CAxxx for every $CCxxx)

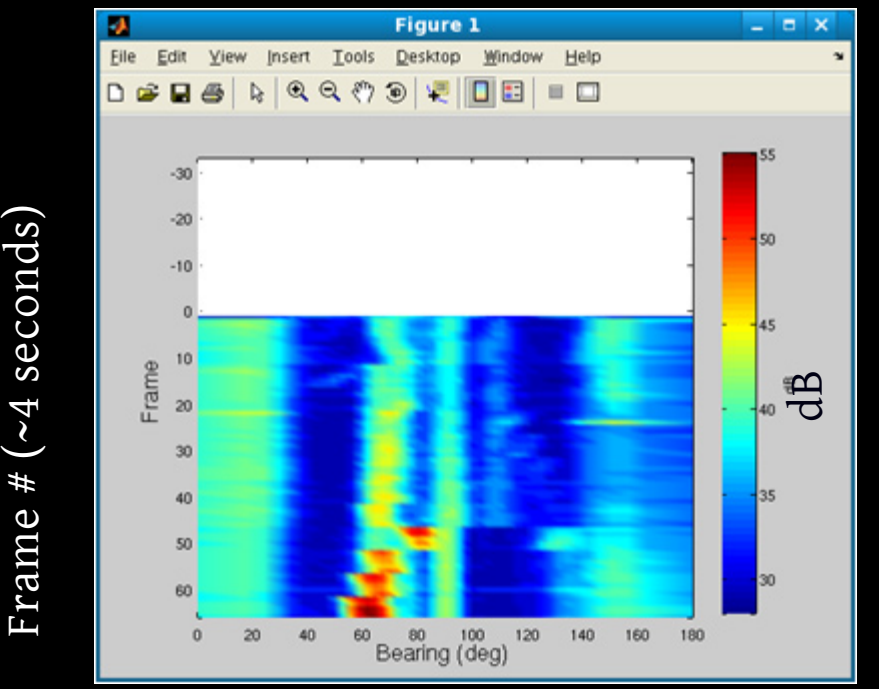
Experiments: Case Studies

- SWAMSI09: Bistatic detection of mine-like targets with two AUVs. Messages: LAMSS_DEPLOY, LAMSS_STATUS, ACTIVE_CONTACTS, ACTIVE_TRACKS

Operator Display: AUVs are green / purple track

Experiments: Case Studies

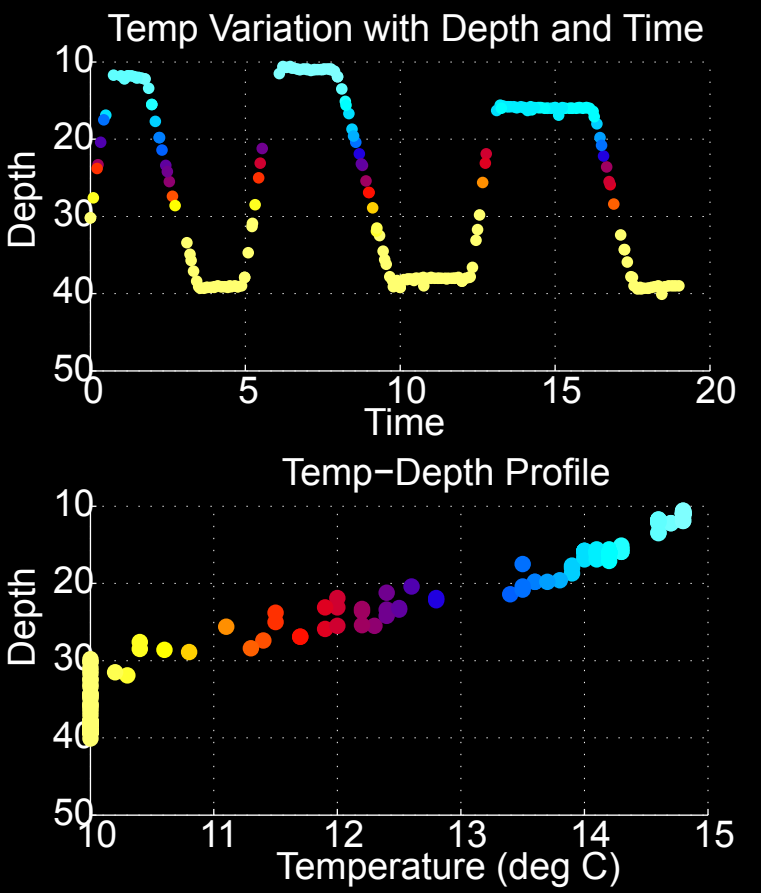
- GLINT09: Active detection of submarine-like target with surface craft (gateway), AUV, and source buoy.
  Messages: LAMSS_DEPLOY, LAMSS_STATUS, WINCH_CONTROL, SOURCE_ACTIVATION



Beam-Time Record

Experiments: Case Studies

- CHAMPLAIN09: Adaptive sampling of physical data (T, S, c) using an AUV
  - Messages: LAMSS_DEPLOY, LAMSS_STATUS, CTDCODEC



CTD Display

Experiments: Case Studies

- GLINT10: Historical “back-fill” of position to provide accurate track of vehicle position.
  Messages: LAMSS_STATUS, LAMSS_CTD, LAMSS_STATUS_FILL_IN

Summary

pAcommsHandler provides:

- A robust and field-tested software interface to the goby-acomms acoustic networking libraries:
  - libdccl: highly efficient but flexible data marshalling
  - libqueue: priority queuing to maximize receipt of desired messages (overall and timeliness)
  - libamac: basic TDMA medium access control
  - libmodemdriver: abstracted interface to the WHOI Micro-Modem.

pAcommsHandler code: moos-ivp-local on oceanai

goby-acomms hosting: https://launchpad.net/goby

Acknowledgments

- Office of Naval Research
- Inspiration: R. Stokey (CCL), M. Grund (iMicroModem)
- L. Freitag and the WHOI Micro-Modem group
- P. Newman / M. Benjamin (MOOS-IvP)
- Field trial support: NATO Undersea Research Centre, NUWC (Newport), NAVSEA (Panama City), Bluefin Robotics, Robotic Marine Systems
- Open source projects used by pAcommsHandler & Goby: Xerces, Boost, Crypto++, NCurses, ASIO
- LAMSS: A. Balasuriya, K. Cockrell, S. Petillo