

Goby:

A Framework for Scientific
Autonomous Marine Vehicle Collaboration



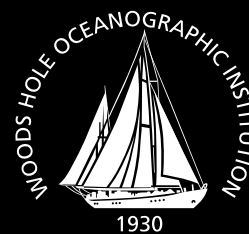
Toby Schneider

MIT/WHOI Joint Program

Henrik Schmidt

MIT Laboratory for Autonomous Marine Sensing Systems

Plus contributions from Goby Developers Team
launchpad.net/~goby-dev



Perfection is achieved, not when there is nothing more to add, but rather when there is nothing more to take away.

— *Antoine de Saint-Exupéry, Terre des Hommes (1939)*

Nasty complexities

Why is marine autonomy difficult?

Physical problems:

- Expensive vehicles
- Limited power
- Poor communications

Human problems:

- Lack of interoperability standards
- Lack of networking / computer systems expertise
- Lack of comprehension of scientific needs by computer folks

What is Goby?

- Goby-**Core** is an open framework for communication based on ZeroMQ:
 - Inside a vehicle (UNIX sockets, TCP / UDP)
 - Between vehicles over ethernet links (TCP / UDP)
 - Between different autonomy systems (MOOS, LCM...)
 - Amongst applications written in 20+ languages
 - Using different messaging protocols
(CMOOSMsg, Google Protocol Buffers, LCM types, boost::serialization...)
- Goby-**Core** also includes an
 - SQLite3 / PostgreSQL logger (`goby_database`)
 - Web browser based scope, command, etc. tool (`goby_liaison`)

What is Goby?

- Goby-**Protobuf**
 - implements Goby-**Core** for C++ and Google Protocol Buffers
 - is a high quality, easy to learn replacement for existing architectures like MOOS and LCM for new projects.
- Goby-**Acomms** (another talk) provides a field-tested networking system for acoustic and other slow links that seamlessly integrates into Goby-**Protobuf**
- Goby-**MOOS** provides
 - pAcommsHandler & iCommander acomms apps
 - a MOOS to Goby gateway
 - .alog file to/from SQLite3 conversion tools.
 - CMOOSMsg to Protobuf translators

Why Goby-Protobuf?

Much of Goby-Core (and the Protobuf implementation) was motivated to write the “next generation” MOOS.

What works in MOOS:

- Discrete processes
- Publish / subscribe
- Easy to learn
- Choice of a high quality, general purpose language (C++)

Goby-Protobuf: Typed Messages

MOOS: *Untyped variables lead to trouble in large projects*

Data sources (e.g. sensors) provide *objects* not single scalars or strings. Examples:

- vehicle position and Euler angles.
- acoustic environment

MOOS requires either

- splitting the object into scalars (problems “restitching”)
- creating custom strings: costly in programmer time, CPU time, and prone to subtle errors.

Goby-Protobuf: Typed Messages

Goby: Strictly typed variables allow compile time type checking and integrated support for arbitrarily complex objects.

Google Protobuf

```
message HelloWorldMsg
{
    required string telegram = 1;
    required uint32 count = 2;
}
```

```
static int i = 0;

HelloWorldMsg msg;
msg.set_telegram("hello world!");
msg.set_count(++i);

publish (msg) ;
```

C++

Static types allows Doxygen-style analysis of messaging

Goby-Protobuf: Configuration

MOOS: Configuration is prone to syntactical and type errors

`CProcessConfigReader` does not provide

- Checking for values that shouldn't exist
- Checking that required values do exist
- Type checking on values

Other oddities include

- Removal of all spaces from strings
- Confusion between `CMOOSFileReader::GetValue` (reads from entire file) and `CProcessConfigReader::GetConfigurationParam` (reads from block)
- Difficulty in reading repeated values

Goby-Protobuf: Configuration

Goby: Configuration schema allows strict behind-the-scenes validation

Application author schema:

```
message MyConfig {  
  optional double speed = 1 [default=3];  
  required string port = 2;  
}
```

Valid configuration file:

```
speed: 34.25  
port: "/dev/ttyUSB1"
```

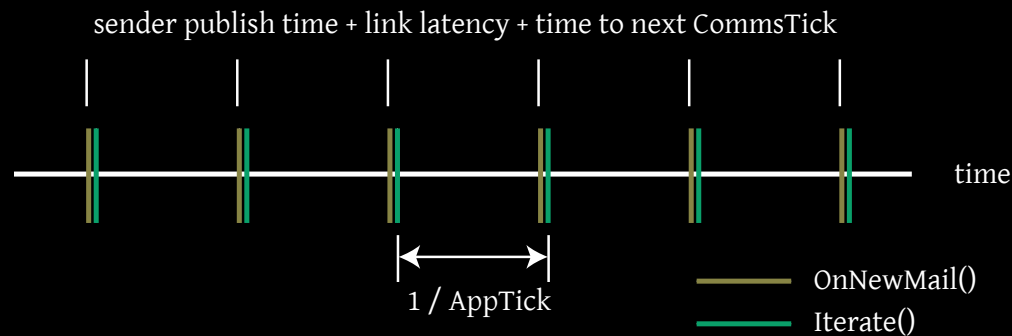
Invalid configuration file (caught at launch):

```
speed: "high" — wrong type  
prt: "/dev/ttyUSB1" — typo - no field "prt"
```

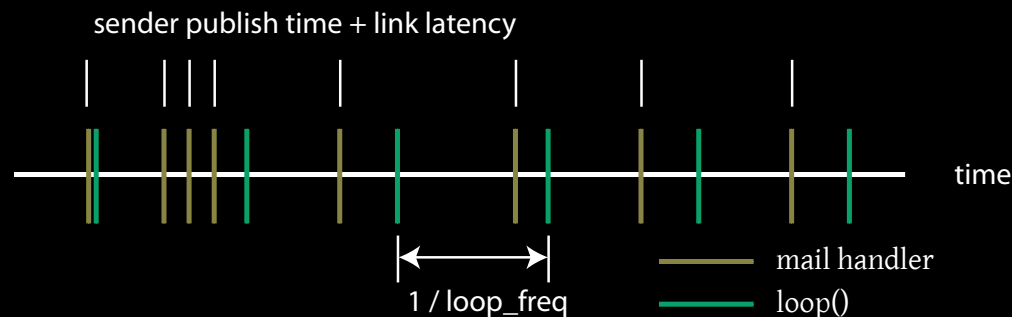
Goby-Protobuf: Asynchronous

MOOS: Messaging is done synchronously (`CommsTick` frequency)

MOOS messaging is done on a regular frequency, leading to artificial and unnecessary latencies.



Goby: Messages are delivered asynchronously, so latencies are governed by constraints of the physical link(s) only.



“Legacy code” often differs from its suggested alternative by actually working and scaling.

— Bjarne Stroustrup,

<http://www2.research.att.com/~bs/bs_faq.html#legacy>

Goby-Core: Interoperability

Much good work has already been done and tested, so we

- Want to use existing marshalling schemes (e.g. CMOOSMsg), existing transports (MOOS TCP) to interface “legacy” code to Goby-Protobuf
- Want to use favorite programming language (good code in several languages is better than bad code in one)

How?

Split the transport (e.g. TCP) and data marshalling (e.g. Protobuf) into separate entities.

Goby-Core: Interoperability

Goby-Core adds a thin header for existing marshalling schemes: [MarshallingScheme][Identifier][Data]

- *Marshalling scheme* (32 bit unsigned int). String = 1, Protobuf = 2, CCL = 3, MOOS = 4, DCCL = 5, LCM = 6 ...
- “C style” string *identifier* (variable length):
{Type Name} + ‘/’ + {Group Name} + ‘/’ + {Subgroup Name} ... + ‘\0’
 - MOOS: “CMOOSMsg/DB_TIME/”
 - Protobuf: “SomeMessage/”
 - LCM: “my_lcm_type/my_lcm_channel/”
- Data (variable length) encoded as defined by *marshalling scheme*.

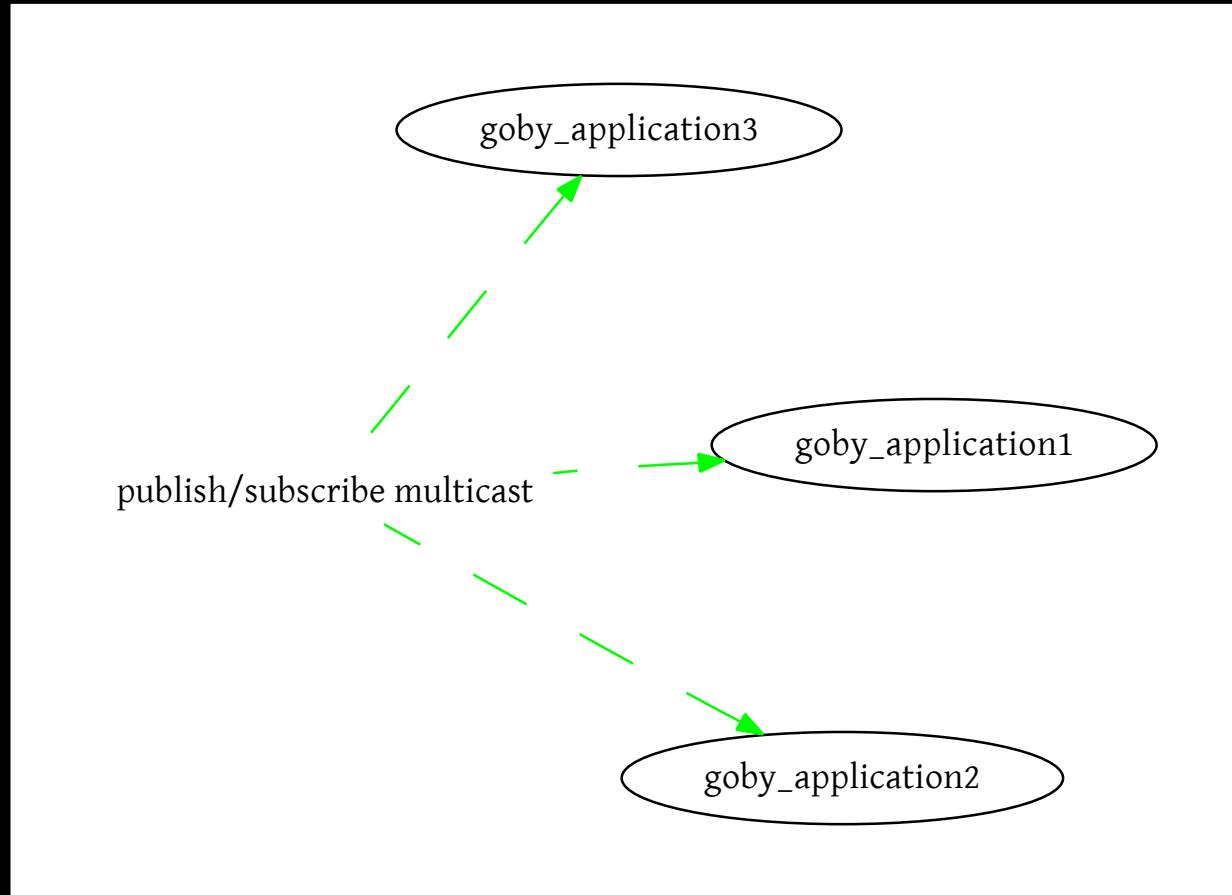
Goby-Core: Transport

Intravehicle and over ethernet, Goby-Core uses ZeroMQ:

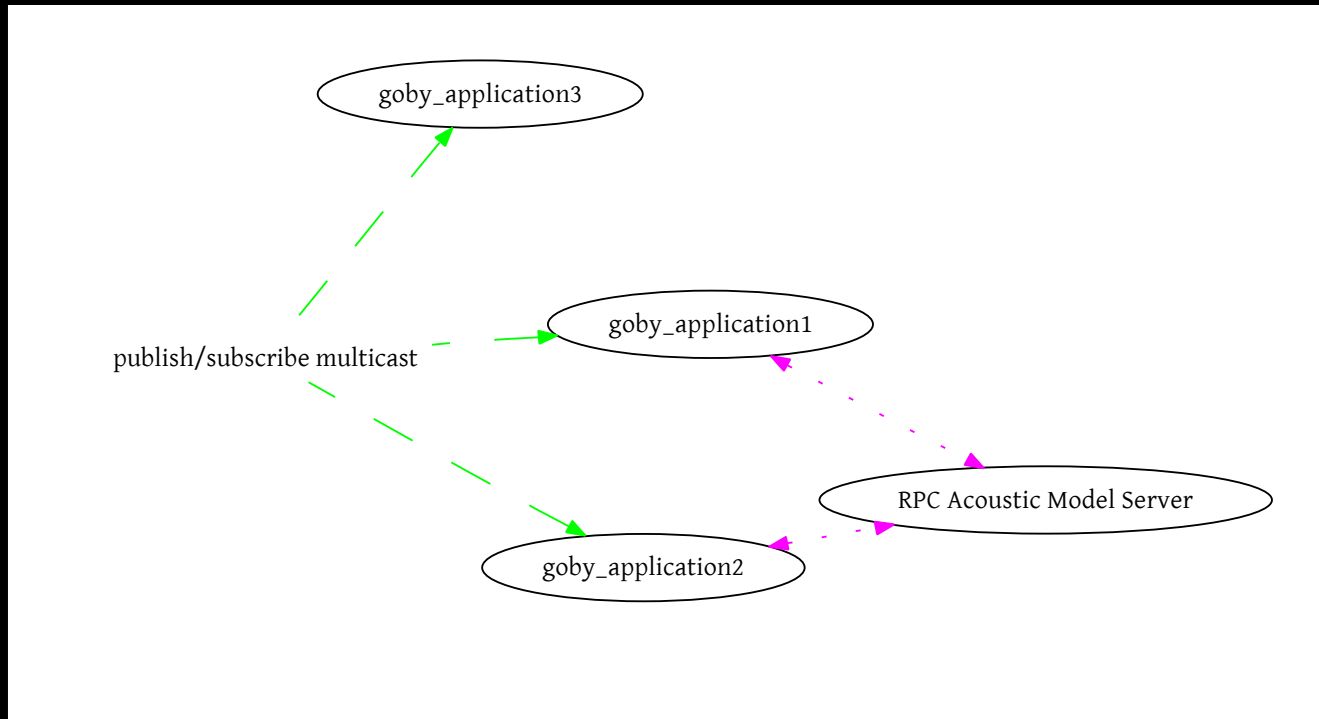
- Brokerless (e.g. no MOOSDB) abstraction of
 - TCP
 - PGM (reliable multicast over UDP)
 - UNIX sockets
- Mimics well known UNIX sockets API in 20+ languages
- Abstracts datagrams and streams into “messages”
(message = a variable length collection of bytes)

Over slow links, Goby-Core uses Goby-Acomms. Selection and routing is handled by `goby_router`.

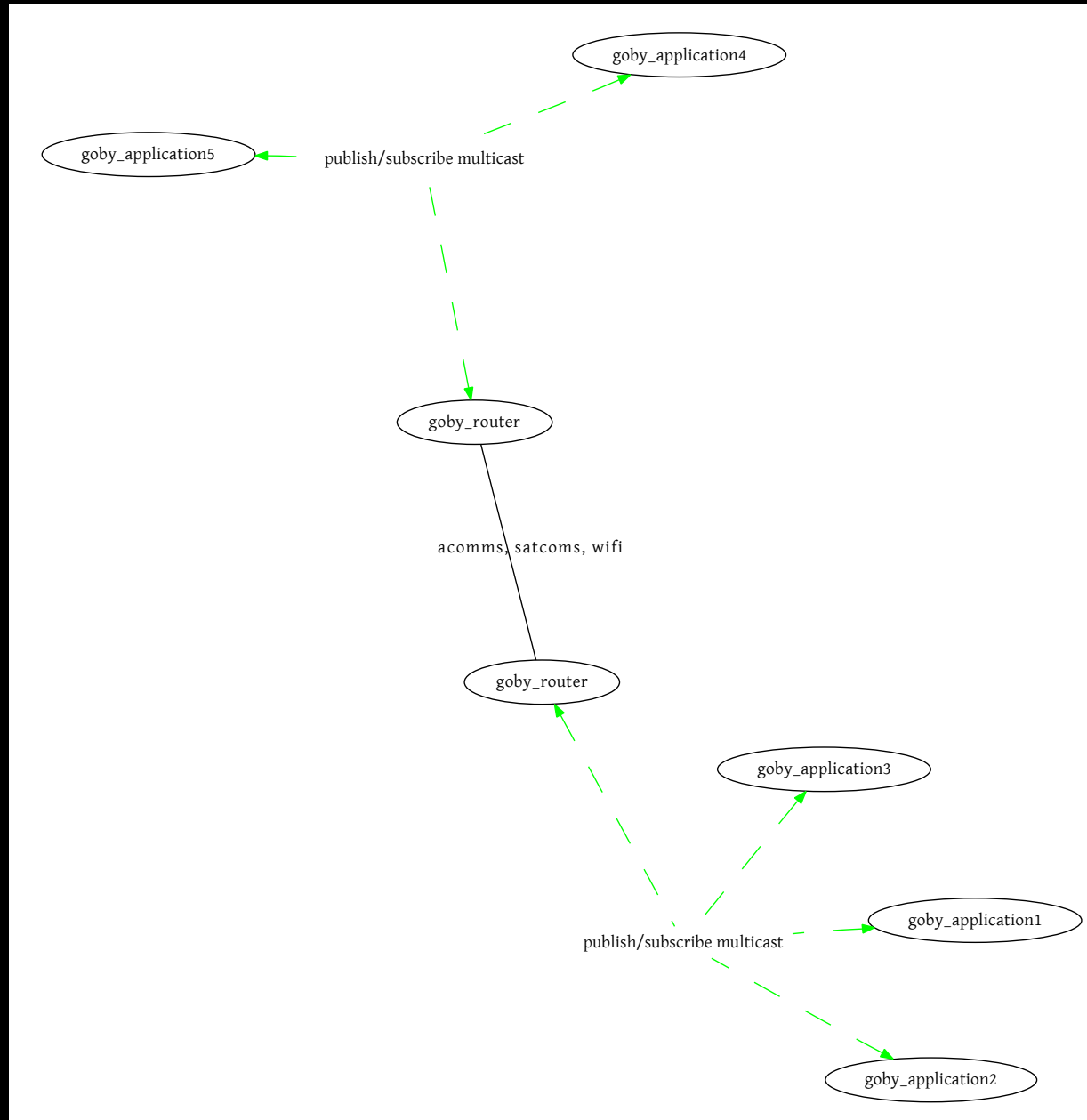
Topologies: Simple Goby Platform



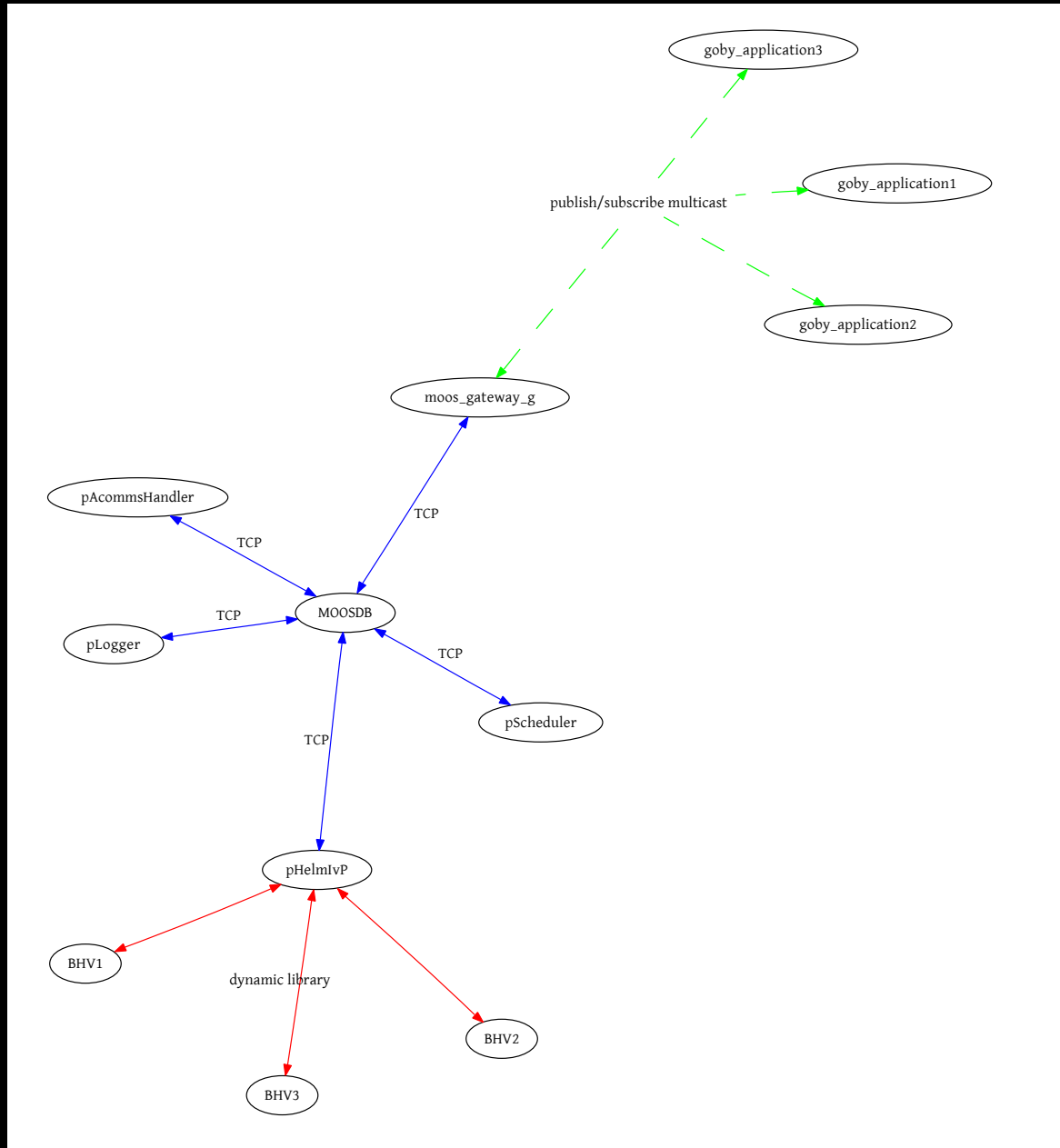
Topologies: Goby pub-sub and RPC



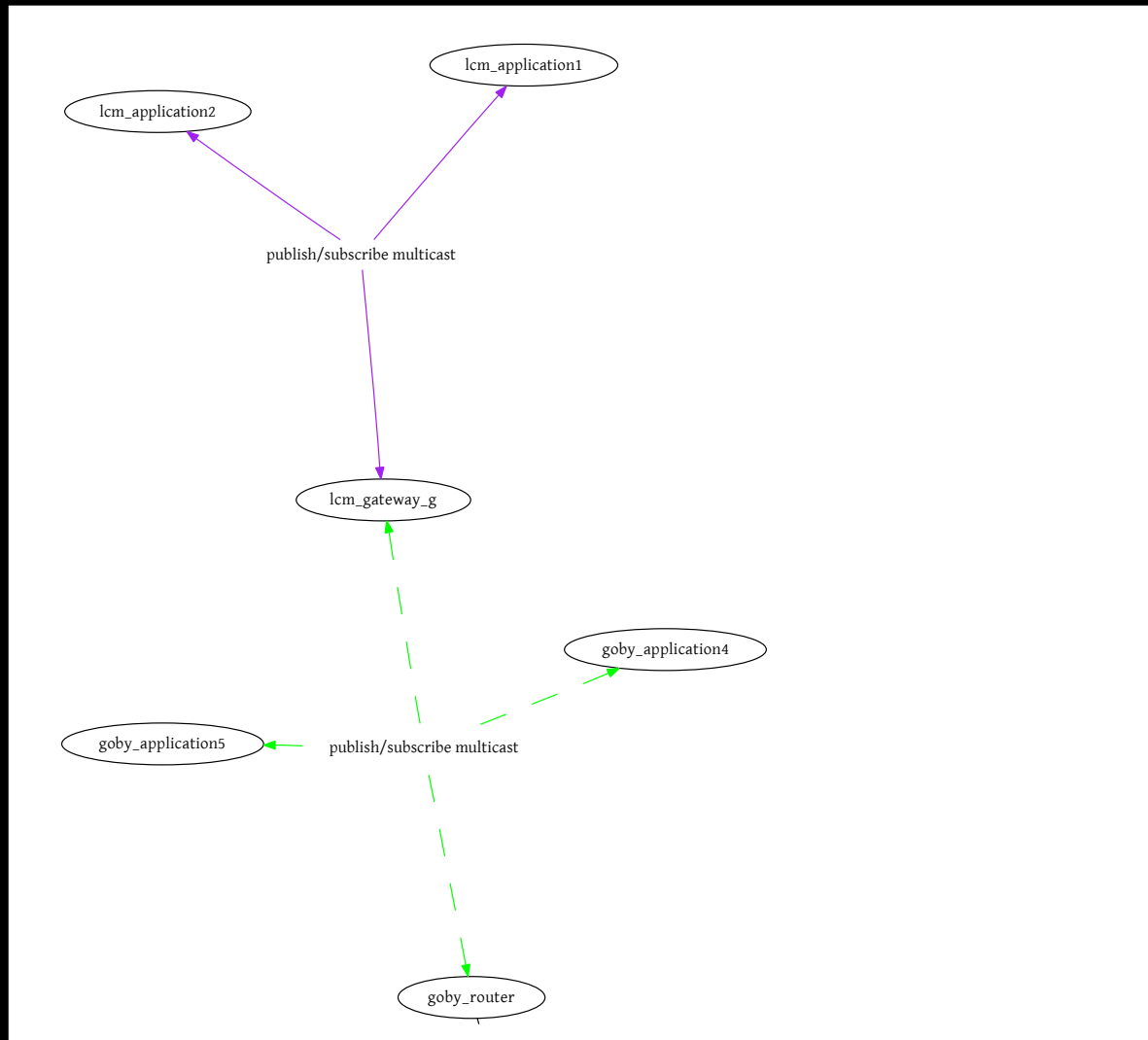
Topologies: Two Goby Platforms



Topologies: Goby & MOOS



Topologies: Goby & LCM



Goby Liaison

An extensible web tool, shown "scoping" CMOOSMsgs:

The screenshot displays the Goby Liaison web interface. At the top, there is a navigation bar with the 'launchpad' logo on the left, the text 'goby liaison: Unicorn' in the center, and the 'gobysoft.org' logo on the right. Below the navigation bar, there are several control elements: a 'Home Scope' link, a 'Play/Pause' button (currently showing 'Playing...'), an input field for adding a subscription (e.g., 'NAV*' or 'NAV_X') with an 'Apply' button, and a 'Subscriptions (click to remove):' button labeled 'NAV*'. Below this, there is an 'Add history for key:' section with a dropdown menu showing 'NAV_DEPTH' and an 'Add' button. The main part of the interface is a 'Set regex filter:' section with a 'Column:' dropdown set to 'Key', an 'Expression:' input field containing '*[^(_YAW)]', and buttons for 'Set', 'Clear', and 'Examples'. Below the filter section is a data table with the following columns: Key, Type, Value, Time, Communit, Source, and Source Aux. The table contains 14 rows of data for various navigation keys. Below the table, there is a 'History for' section with a dropdown menu showing 'NAV_DEPTH' and a '(click to remove)' button, followed by a 'Plot' button. At the bottom, there is another data table with the same columns as the first table, but it only contains 14 rows of data for the 'NAV_DEPTH' key.

launchpad goby liaison: Unicorn gobysoft.org

Home Scope

Play/Pause Playing...

Add subscription (e.g. NAV* or NAV_X): Apply

Subscriptions (click to remove): NAV*

Add history for key: NAV_DEPTH Add

Set regex filter: Column: Key Expression: *[^(_YAW)] Set Clear Examples

Key	Type	Value	Time	Communit	Source	Source Aux
NAV_ALTITUDE	double	10.74303562070043	07/07/11 19:48:07	unicorn	iHuxley	moos_gateway_g
NAV_DEPTH	double	0.25696437929956945	07/07/11 19:48:07	unicorn	iHuxley	moos_gateway_g
NAV_HEADING	double	14.485063573529334	07/07/11 19:48:07	unicorn	iHuxley	moos_gateway_g
NAV_LAT	double	44.092428311265138	07/07/11 19:48:07	unicorn	iHuxley	moos_gateway_g
NAV_LONG	double	9.8586603015923355	07/07/11 19:48:07	unicorn	iHuxley	moos_gateway_g
NAV_PITCH	double	0.0037524340610175836	07/07/11 19:48:07	unicorn	iHuxley	moos_gateway_g
NAV_SPEED	double	0.0082757386044119696	07/07/11 19:48:07	unicorn	iHuxley	moos_gateway_g
NAV_X	double	800.38489865673932	07/07/11 19:48:07	unicorn	iHuxley	moos_gateway_g
NAV_Z	double	-0.25696437929956945	07/07/11 19:48:07	unicorn	iHuxley	moos_gateway_g

History for NAV_DEPTH (click to remove)

Plot

Key	Type	Value	Time	Communit	Source	Source Aux
NAV_DEPTH	double	0.25696437929956945	07/07/11 19:48:07	unicorn	iHuxley	moos_gateway_g
NAV_DEPTH	double	0.25793750366558355	07/07/11 19:48:07	unicorn	iHuxley	moos_gateway_g
NAV_DEPTH	double	0.25915636807309522	07/07/11 19:48:07	unicorn	iHuxley	moos_gateway_g
NAV_DEPTH	double	0.25923596089220718	07/07/11 19:48:06	unicorn	iHuxley	moos_gateway_g
NAV_DEPTH	double	0.25892947838580432	07/07/11 19:48:06	unicorn	iHuxley	moos_gateway_g
NAV_DEPTH	double	0.25831570270870274	07/07/11 19:48:06	unicorn	iHuxley	moos_gateway_g
NAV_DEPTH	double	0.25724159532414509	07/07/11 19:48:06	unicorn	iHuxley	moos_gateway_g
NAV_DEPTH	double	0.2561259482112262	07/07/11 19:48:05	unicorn	iHuxley	moos_gateway_g
NAV_DEPTH	double	0.25499724791298667	07/07/11 19:48:05	unicorn	iHuxley	moos_gateway_g
NAV_DEPTH	double	0.25347654027965927	07/07/11 19:48:05	unicorn	iHuxley	moos_gateway_g

Goby Database

An SQL logger currently with CMOOSMsg and Protobuf plugins:

```
query: select double_value
       from cmoosmsg
       where key='NAV_X'
          and double_value > 0;
```

versus with .alog file:

```
grep " NAV_X" file.alog | sed "s/ */ /g"
| cut -d " " -f 4 | egrep "[^(-)]"
```

Goby Database: Alog converter

`alog2goby_db` converts a MOOS `.alog` file (from `pLogger`) to an SQLite database similar to that generated by `goby_logger`.

Features:

- Support for `CMOOSMsg`
- Support for `scanf`-like string parsing of MOOS strings into Google Protobuf objects. E.g. NMEA-0183 parsing.
- Support for key=value string parsing of MOOS to Protobuf. E.g. `NODE_REPORT` parsing (`Name=AUV1, X=2300, Y=5023, DEPTH=3, ...`).

Goby Project

- Free open source (GNU Public License 3)
- Easily accessible: <http://launchpad.net/goby>
- Documented: <http://gobysoft.org/doc/1.0/>
<http://gobysoft.org/wiki>
- Organized releases: version 1 stable (acoustic comms only); version 2 (contents of this talk) coming this fall
- Well tracked bugs

Please contribute (use the software, report bugs, write code for Goby). Any feedback is helpful and welcome.

This can be your project as much as ours.

Acknowledgments

- Goby-Developers group (MIT / WHOI / UMichigan)
- Office of Naval Research
- Open source projects used by Goby: Boost, Crypto++, NCurses, ASIO, Google Protobuf, ZeroMQ
- All authors and contributors to the C++ language and GNU tools
- LAMSS: S. Petillo, I. Katz, A. Balasuriya, S. Danesh, E. Fischell, M. Benjamin