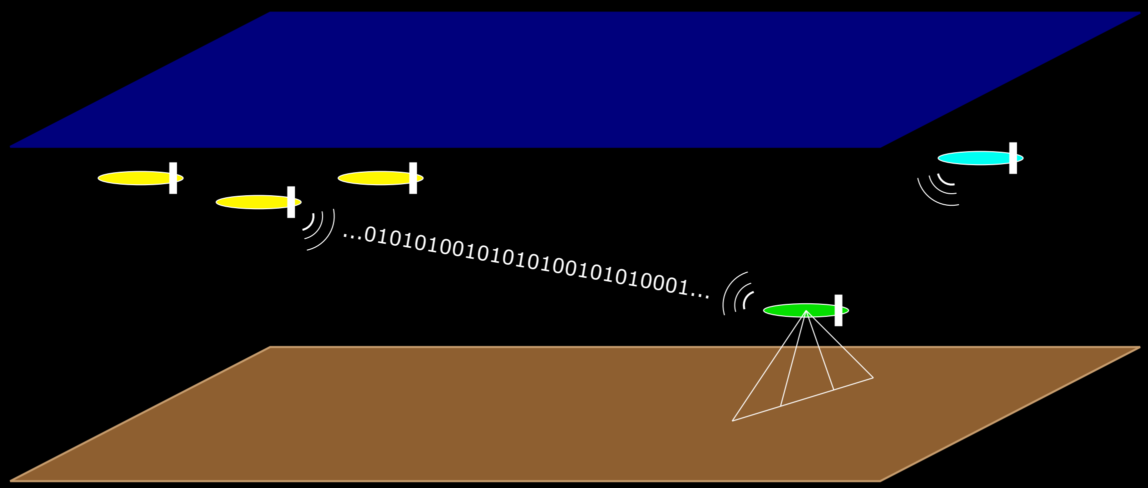
NRL Goby3 Course

Lecture 1: Introduction to Goby3 (MOOS focus)



Toby Schneider

GobySoft, LLC

Mashpee, MA, USA

gobysoft.org
aquatic software

What is Goby?

- A C++ library with a collection of marine robotics related code, including MOOS applications?
- Drivers for acoustic modems and related “slow links” (e.g. satellites)?
- A publish/subscribe middleware (framework)?
- An open source project that is freely available under the terms of the LGPL?
- Any of more than 2,200 species of small, carnivorous fishes?

Goby is all of these things.
The goal is that you can use what you need, and no more.

What is Goby?

- A C++ library with modular components for:
  - acoustic communications
  - utilities for marine software (AIS, seawater calculations, NMEA-0183, ...)
  - time functions, supporting faster-than-realtime sims
- A "nested comms" publish/subscribe middleware:
  - a common transport interface for thread-to-thread, process-to-process, and vehicle-to-vehicle comms
  - application base classes for quickly writing processes
  - extensible marshalling schemes, supporting Protocol Buffers, MAVLink, DCCL, and easily many more.
  - a growing list of useful applications (GUIs, GPSD, logger, ...) and threads (I/O: serial, UDP, TCP, CAN, ...)

Why create Goby?

Marine robotics is small field of engineering, but with some relatively unique technical problems:

- communications (low throughput / high latency)
- harsh environments (expensive deployments)
- wide array of sensors, many specific to seawater

My hope:

Goby is a beginning to some solutions.

More importantly, it is open source so that we (as a community of oceanographic engineers) can build on it, and improve our use of software best practices along the way.

History

MIT / LAMSS (2007-2012) [lamss.mit.edu]:

- pGeneralCodec MOOS Application -> DCCL v1 (XML)
- pAcommsHandler MOOS Application -> Queue / ModemDriver
- Refactor pAcommsHandler into the beginning of Goby (v1)
- DCCL rewrite (v2): XML to Google Protocol Buffers
- Goby 2 (primarily to support DCCL v2)

GobySoft (2013-present) [gobysoft.org]:

- DCCL standalone (v3) library, Goby 2.1
- Goby 3 (nested middleware)

Syllabus

- Day 1 (Thursday August 12): MOOS Focus
  - Lecture 1
  - Break
  - Practical Session 1
  - Lunch
  - Lecture 2
  - Break
  - Practical Session 2
- Day 2 (Friday, August 13): Goby3 Middleware Focus
  - Practical Session 3
  - Lecture 3
  - Lunch
  - Lecture 4 (break in middle)

Repository Structure

Core:

- libgoby.so: acomms, middleware, time, util
- middleware tools: goby_clang_tool, goby_log_tool

ZeroMQ:

- libgoby_zeromq.so
- zeromq apps: gobyd, goby_logger, etc.

MOOS:

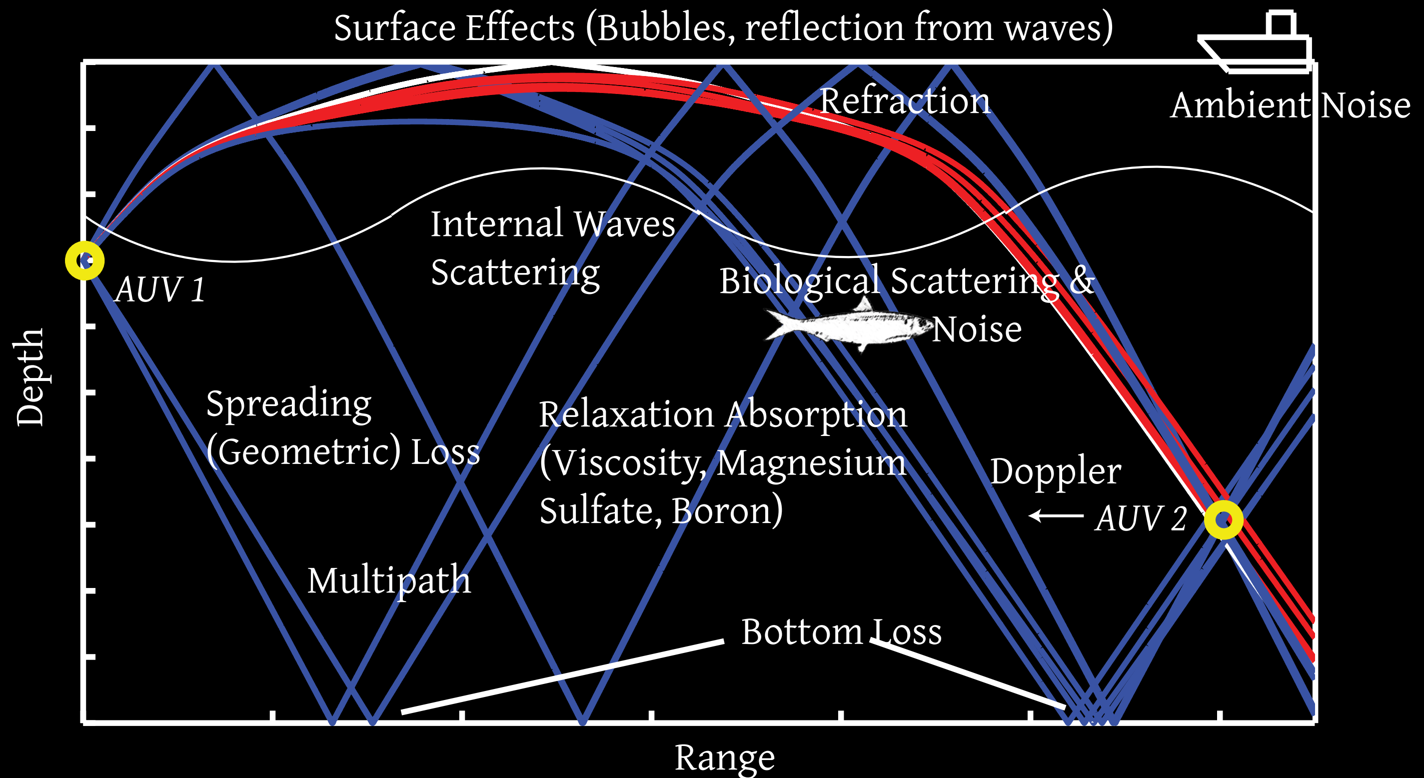
- libgoby_moos.so
- pAcommsHandler, iFrontSeat

(Explore goby3-repository-layout.pdf)

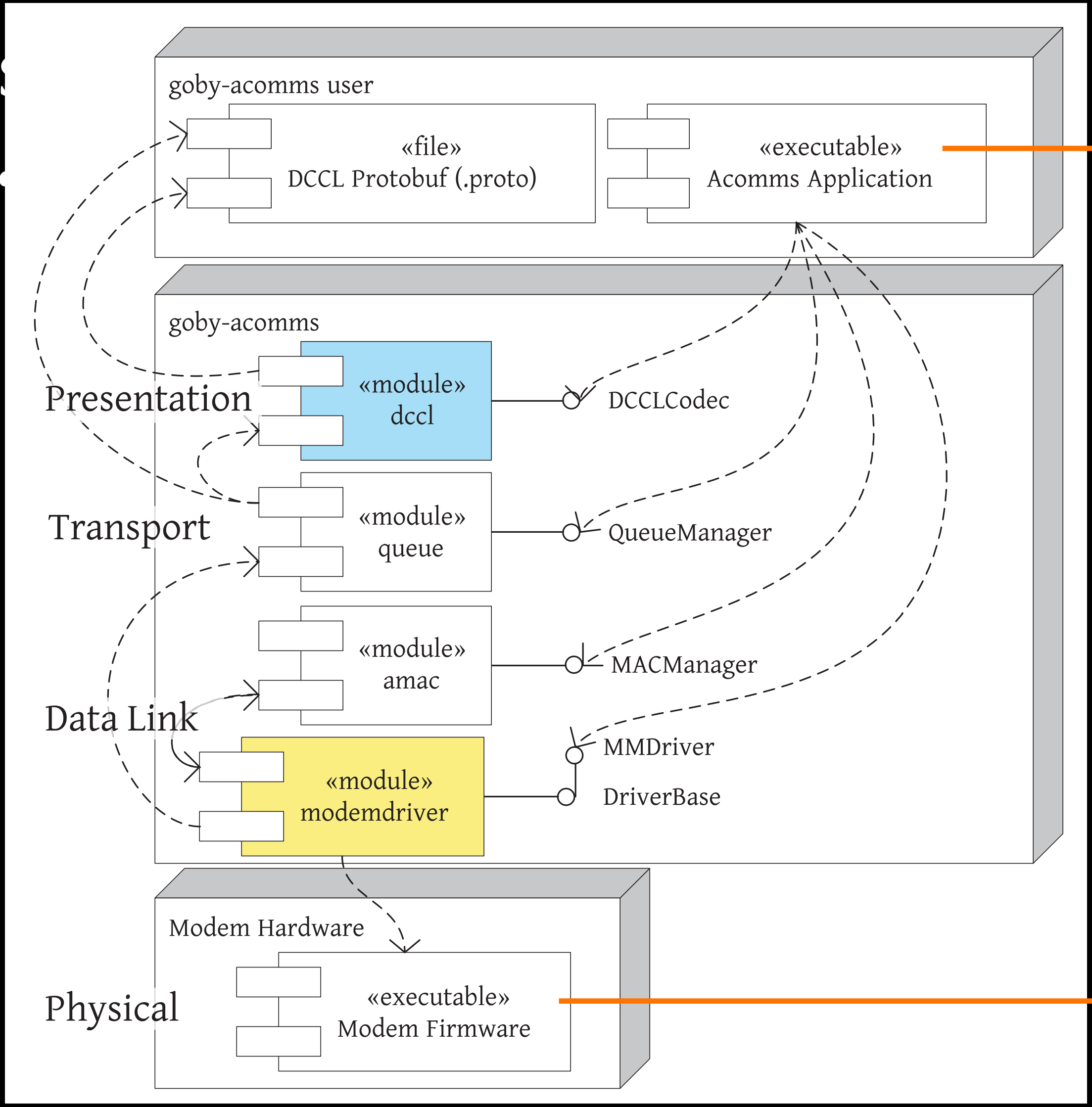
Physical link limitations

Acoustic communication channel is inherently difficult:

- Low bandwidth
- Time (multipath) and frequency (Doppler) spreading
- Signal/Noise: Absorption, scattering, spreading; noise



Goby Acomms



pAcommsHandler (for MOOS use)

WHOI Micro-Modem, Benthos ATM-900, Iridium 9523, etc.

Goby/DCCL - Potential Solutions

Three broad (complementary) approaches to improving marine robotic performance in “poor” link scenarios:

- Improve the throughput and latency of the underlying physical link *(advances in signal processing).*
- Improve the autonomous capabilities of the individual robots *(advances in perception, autonomous control, machine learning).*
- Improve the information value of the data transmitted *(advances in source encoding).* <-- Our focus in DCCL

What do *we* want?

- Vehicle position / attitude
  - latitude, longitude, depth, altitude
  - Euler angles: pitch, roll, yaw
- Vehicle health
  - battery remaining
  - sensors' status
- Sensor output
  - scalar data (pressure, temperature, turbidity)
  - imagery (sea floor, sidescan sonar)
  - processed data (beam/time record from sonar)
- Commands
- Shared state (handshaking, mission updates)

All data are not created equal

Clue from physics:

- You will not run into a `double` on the street.
- Sensors measure quantities with dimensions (mass, temperature, salinity, pressure), not *numbers*.
- These quantities are governed by rules, which are often understood, at least probabilistically:
  - Prior measurements in the region (for sensor data)
  - Motion prediction / tracking (for navigation data)
  - Conops requirements (for command messages)

All data are not created equal

Clue from object-oriented programming:

- Group collection of primitive types into sensible unit (object)
  - e.g. CTDMessage, DeployCommand, NodeReport, SonarSettings

Clue from design:

- 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)

DCCL Overview

The Dynamic Compact Control Language v3 contains:

- an interface description language (IDL) with static (compile-time) units of measure support
- a flexible data marshalling (source encoding/decoding) C++ library (which allows user extension).

Open source (LGPL 2) implementation

(http://libdccl.org).

(Prior releases part of Goby project / DCCL3 is stand-alone)

Interface Description Language (IDL)

```
message CommandMessage
{
  required int32 destination = 1
  optional string description = 2
  enum SonarPower { NOMINAL = 10; LOW = 5; OFF = 0; }
  optional SonarPower sonar_power = 10;
  required double speed = 11
  repeated int32 waypoint_depth = 12
}
```

Base Protobuf message example

DCCL acts as an “invisible” extension to Google Protocol Buffers (Protobuf).

- Code unaware of DCCL can still use the same messages (e.g. internal vehicle data)
- Extensions provide additional information (e.g. numeric bounding).

Protobuf is widely used and provides syntax checking and multi-language support (C++, Python, Java, C, ...)

DCCL IDL: Field bounds

```
import "dccl/protobuf/option_extensions.proto";

message CommandMessage
{
  option (dccl.msg) = { id:  125  max_bytes:  32  codec_version: 3 }

  required int32 destination = 1
                 [(dccl.field)  = { max: 31 min: 0 in_head: true }];
  optional string description = 2
                 [(dccl.field).omit = true];
  enum SonarPower { NOMINAL = 10; LOW = 5; OFF = 0; }
  optional SonarPower sonar_power = 10;
  required double speed = 11
                 [(dccl.field) = { units { base_dimensions: “LT^-1” }
                                   max: 2.0  min: -0.5 precision: 1 }];
  repeated int32 waypoint_depth = 12
                 [(dccl.field) = { units { base_dimensions: “L” }
                                   max: 40  min: 0 max_repeat: 4 }];
}
```

DCCL message example

Numerics are bounded by

- *max*: largest value
- *min*: smallest value
- *precision*: *decimal* digits of precision preserved.

Message level information

- id: identifies message (CommandMessage == 125)
- max_bytes: enforced upper bound for MTU targetting

DCCL IDL: Static units of measure

For scientific data, unit safety is as important as type safety.

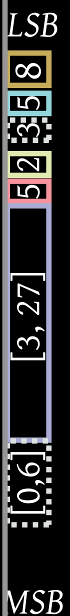
Each numeric field is given a unit system (e.g. SI) and specified dimension (e.g. length, power, speed) or a unit outside a consistent system (e.g. nautical mile).

BASE DIMENSIONS IN DCCL

Physical dimension	Symbol character
length	L
time	T
mass	M
plane angle	A
solid angle	S
current	I
temperature	K
amount	N
luminous intensity	J
information	B
dimensionless	-

Optionally, DCCL produces "unit safe" accessors and mutators for fields with units using the Boost Units library.

DCCL Default encoders (sizes)




```
import "dccl/protobuf/option_extensions.proto";

message CommandMessage
{
  option (dccl.msg) = { id:  125  max_bytes:  32  codec_version: 3 }

  required int32 destination = 1
                [(dccl.field)  = { max: 31 min: 0 in_head: true }];
  optional string description = 2
                [(dccl.field).omit = true];
  enum SonarPower { NOMINAL = 10; LOW = 5; OFF = 0; }
  optional SonarPower sonar_power = 10;
  required double speed = 11
                [(dccl.field) = { units { base_dimensions: “LT^-1” }
                                  max: 2.0  min: -0.5 precision: 1 }];
  repeated int32 waypoint_depth = 12
                [(dccl.field) = { units { base_dimensions: “L” }
                                  max: 40  min: 0 max_repeat: 4 }];
}
```

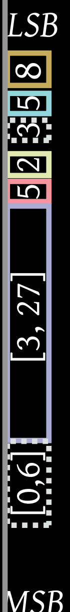
Header Fields:

- 8 bits header (dccl.id)
- 5 bits: destination $[0, (2^5-1)]$
- (3 bits padding to byte)

Body Fields (can be encrypted):

- 2 bits: sonar_power
- 5 bits: speed
- [3, 27] bits: waypoint_salinity vector (size varies on # of elements)
- ([0, 6] bits padding to byte)

DCCL Default encoders (example)




```
import "dccl/protobuf/option_extensions.proto";

message CommandMessage
{
  option (dccl.msg) = { id:  125  max_bytes:  32  codec_version: 3 }

  required int32 destination = 1
                [(dccl.field)  = { max: 31 min: 0 in_head: true }];
  optional string description = 2
                [(dccl.field).omit = true];
  enum SonarPower { NOMINAL = 10; LOW = 5; OFF = 0; }
  optional SonarPower sonar_power = 10;
  required double speed = 11
                [(dccl.field) = { units { base_dimensions: “LT^-1” }
                          max: 2.0  min: -0.5 precision: 1 }];
  repeated int32 waypoint_depth = 12
                [(dccl.field) = { units { base_dimensions: “L” }
                          max: 40  min: 0 max_repeat: 4 }];
}
```

Message definition

Example instantiation (x)

and encoded values (x_{enc})

x_{enc} (bin)	x_{enc} (dec)	x
11111010	250	id: 125 (CommandMessage)
00011	3	destination: 3
000	(padding)	
10	2	sonar_power: LOW (i = 1)
10001	17	speed: 1.2
100	4 [10 15 10 12]	waypoint_depth: [10, 15, 10, 12]
[001010		
001111		
001010		
001100]		
000000	(padding)	

hex:	fa	03	46	2a	8f	c2	00
bin:	11111010	00000011	01000110	00101010	10001111	11000010	00000000

gobysoft.org

Goby/Queue: Priority buffering

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

Solution: Prioritize data based on

1. overall value
2. time-sensitivity

Goby/Queue: Algorithm

Each DCCL message type has its own queue.

Each queue has:

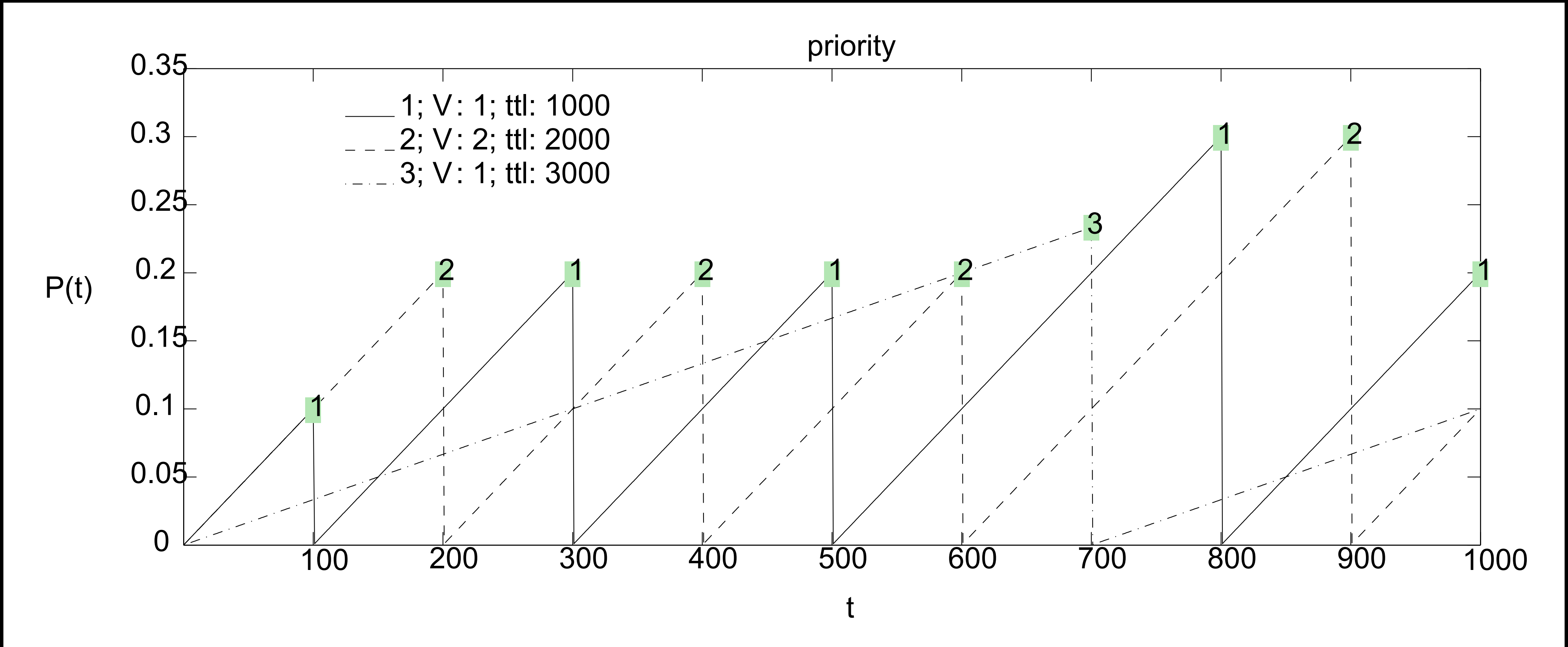
1. base value [V_{base}]
2. time-to-live [ttl]
3. time when queue was last used (t_{last})

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

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

Goby/Queue: Example

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



Goby/Queue: Additional features

Transparent stitching of DCCL messages to form larger data frames.

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

Link-layer acknowledgments handled (e.g. Micro-Modem $CAACK).

Goby/AMAC

Acoustic Medium Access Control (using Decentralized Time Division Multiple Access (TDMA))

- Required by some drivers, but not all.
- Two components:
    - std::list<ModemTransmission>
    - timer
- Each ModemTransmission has a source address and a duration (slot_seconds)
- When the end of the list is reached, it is restarted.
- All vehicles must have globally synchronized clocks.
- Start of the list is (by default) synchronized to the start of the current day (in UTC)

Goby/ModemDriver

Problem:

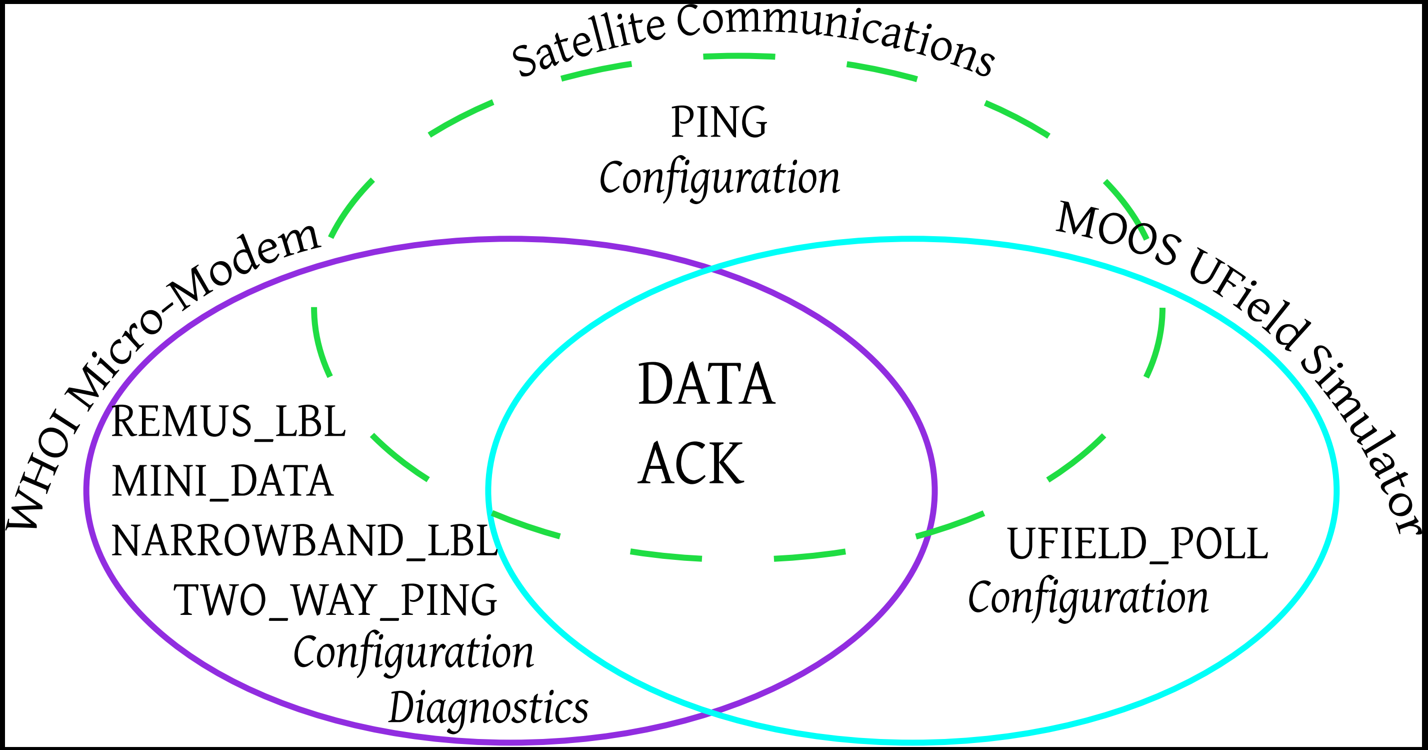
- No standard API to acoustic modems
- Modems provide useful features beyond strict functionality of a modem (send bytes from point A -> B)

Goby approach (complexity layered on simplicity):

- Define core requirement of modem driver as sending data (simple to implement)
- All other features (ranging pings, LBL, USBL) are extensions to modem driver
- User can ignore extensions if only data transmission is required (simple), and later add in extensions (complex) as needed.

ModemDriver

- can work with any system that can transmit datagrams (including existing systems such as UDP/IP)
- preserves access to unique & advanced hardware features.

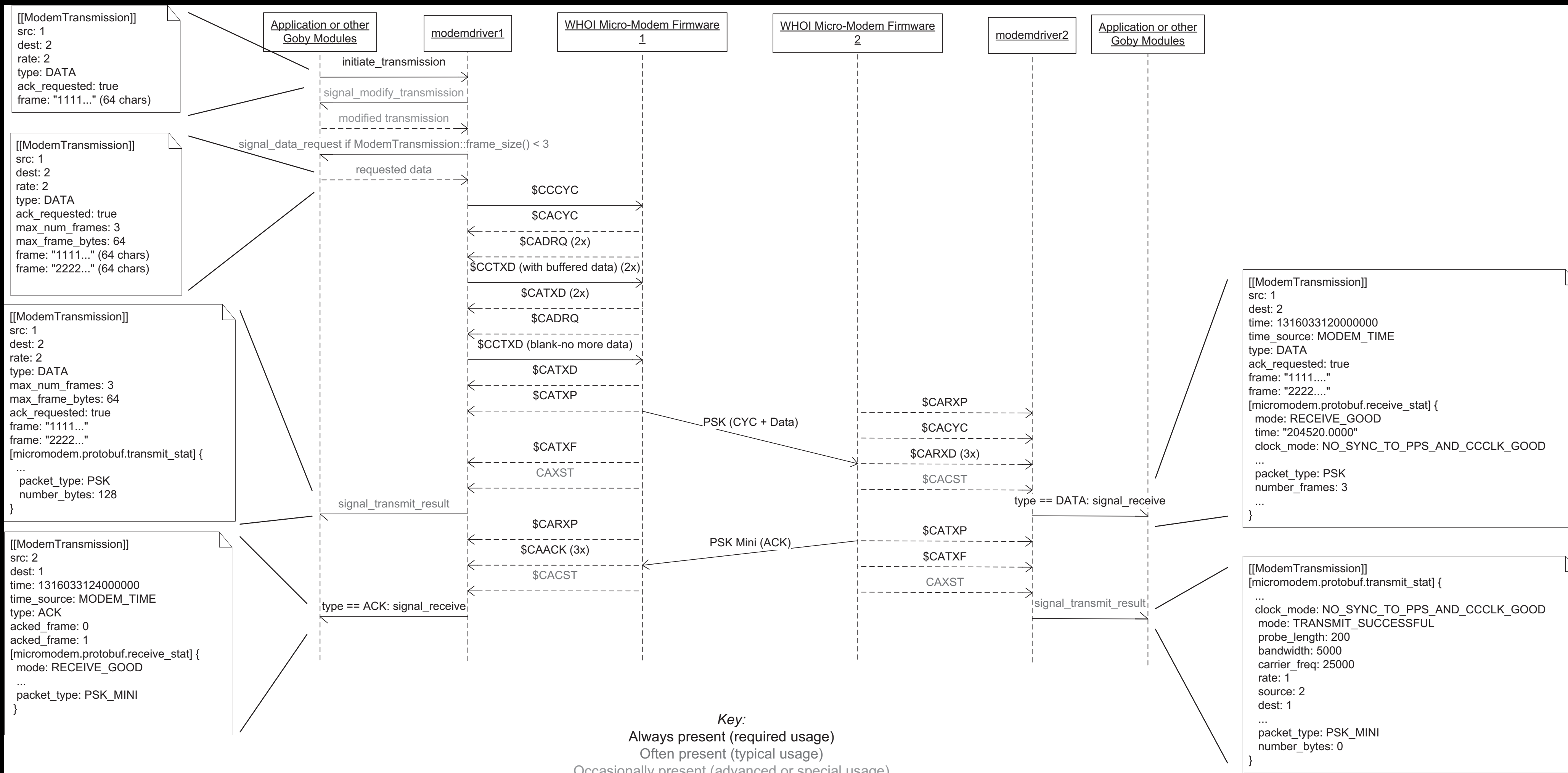


ModemDriver design

All drivers subclass goby::acomms::ModemDriverBase which provides

- Serial or TCP connection to the modem
- Set of callbacks (signals):
  signal_receive
  signal_data_request
  signal_modify_transmission
- Transmit virtual method:
  handle_initiate_transmission
- Most data between DriverBase and the implementation is in the form of the goby::acomms::protobuf::ModemTransmission message

Example: Micro-Modem Data



NETSIM

Hardware-in-the-loop virtual testbed

- Audioserver
- WHOI Micro-Modems (4x)
- We will use a subset of the functionality ("perfect channel") for this course.
- This allows us to explore the hardware without the complications of accurate channel simulation.