MIT 2.680

UNMANNED MARINE VEHICLE AUTONOMY, SENSING, AND COMMUNICATIONS

Lecture 11- Inter-Vehicle Messaging

April 2th, 2024

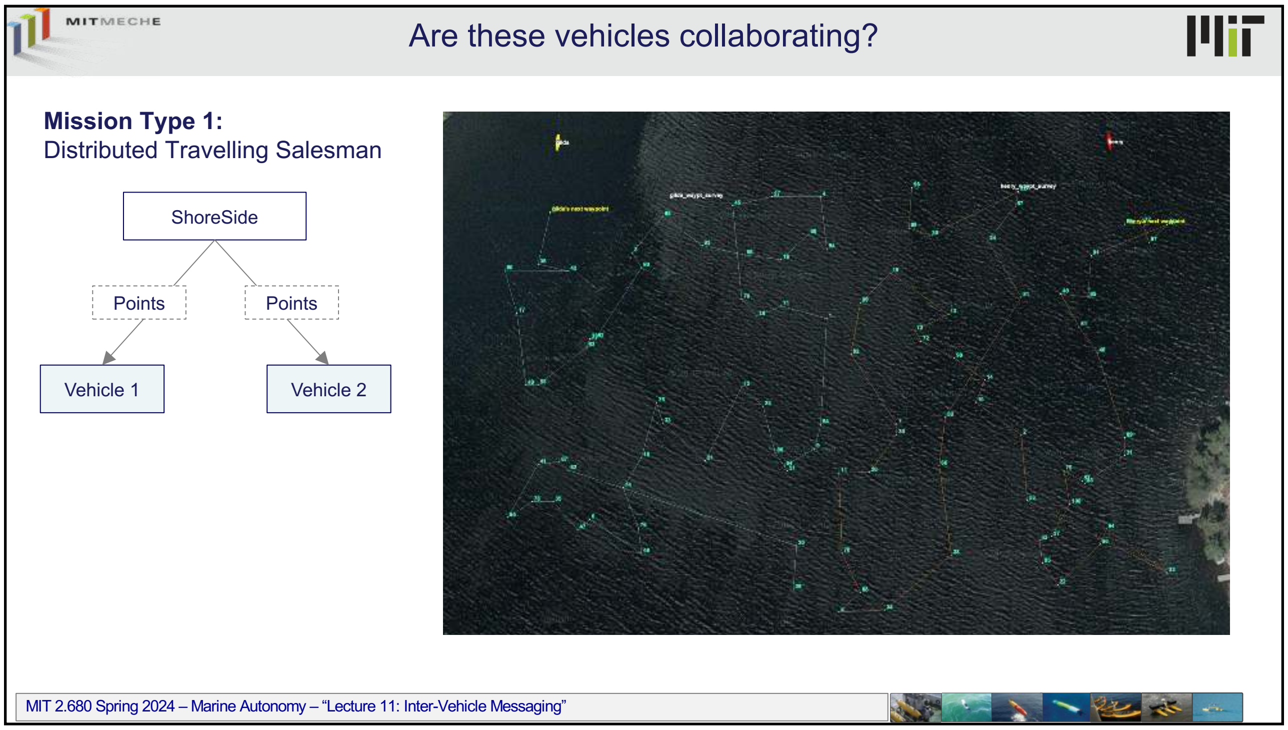
Web: http://oceanai.mit.edu/2.680

Email:
Mike Benjamin, mikerb@mit.edu

MIT 2.680 Spring 2024 – Marine Autonomy – "Lecture 11: Inter-Vehicle Messaging"

Photo by Arjan Vermeij
GLINT '09

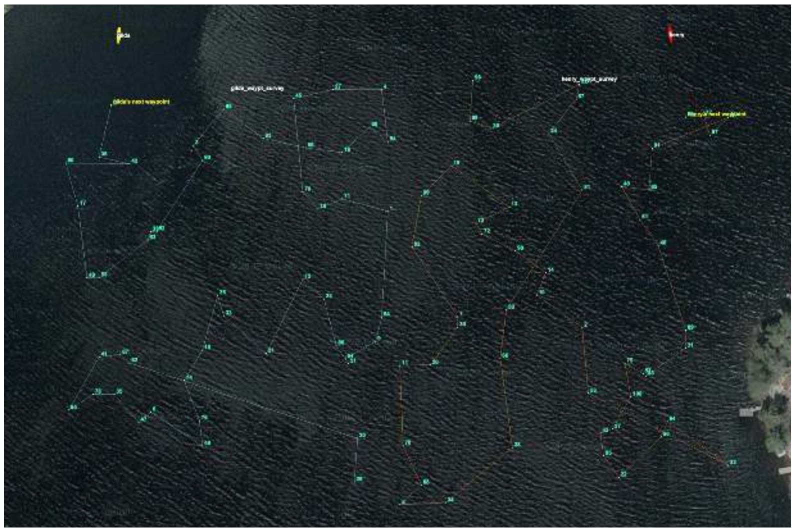
1



Are these vehicles collaborating?

Mission Type 1:
Distributed Travelling Salesman

$$\begin{tikzcd} & \text{ShoreSide} \arrow[dl, dash] \arrow[dr, dash] & \\ \text{Points} \arrow[d] & & \text{Points} \arrow[d] \\ \text{Vehicle 1} & & \text{Vehicle 2} \end{tikzcd}$$

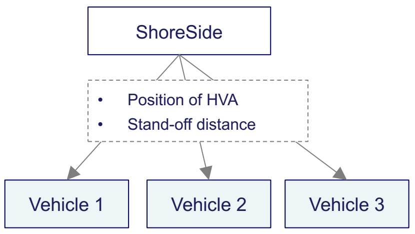


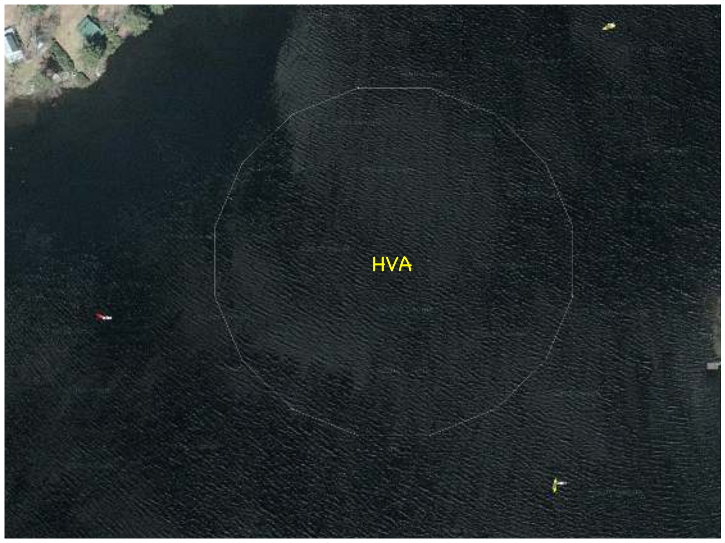
MIT 2.680 Spring 2024 – Marine Autonomy – "Lecture 11: Inter-Vehicle Messaging"

2

Are these vehicles collaborating?

Mission Type 2:
Protection of High Value Asset (HVA)





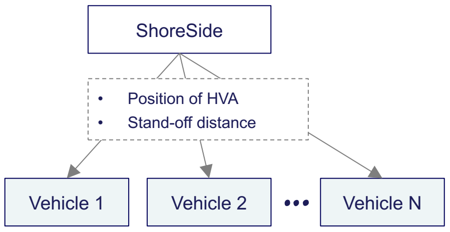
Two modes shown:

- Sharing local position information for collision avoidance only.
- Inter-vehicle messaging for achieving spacing parity

3

Are these vehicles collaborating?

Mission Type 3:
Protection of High Value Asset (HVA)
Plus Decentralized Interdiction



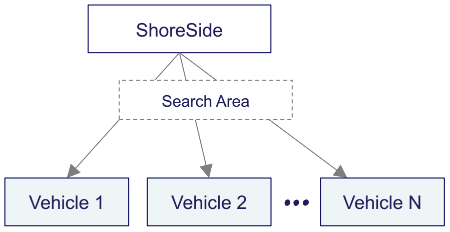
Two types of inter-vehicle communication:

- Position information for collision avoidance and for achieving position parity (protocol based)
- Inter-vehicle auctions for decentralized decision-making for handling “intruders”

4

Are these vehicles collaborating?

Mission Type 4:
Distributed Area Coverage/Survey



- Position information for collision avoidance and for achieving region parity (protocol based)
- Each vehicle drives toward the centroid of its own locally understood Voronoi region. The region evolves as information arrives.

5

Inter-Vehicle Communications

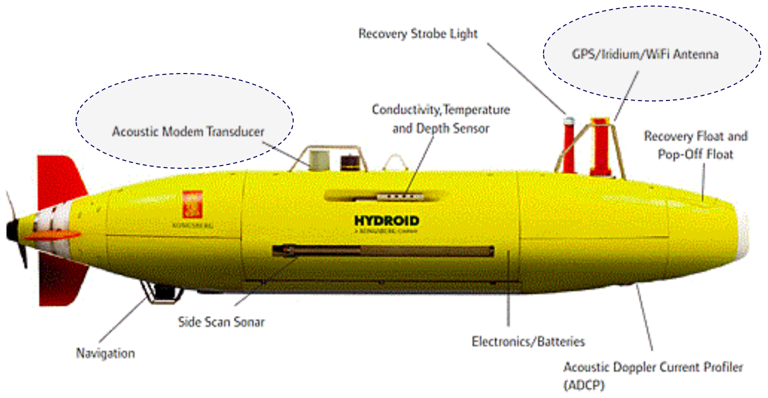
- Methods of Inter-vehicle communications
- Limits on inter-vehicle communications
- Simulating inter-vehicle communications on a network
- uFieldNodeComms
- uFieldMessageHandler
- Indicators of successful messaging
- Debugging dropped messages
- Lab Preview

6

Inter-Vehicle Communications

How do two vehicles / robots / machines talk to each other?

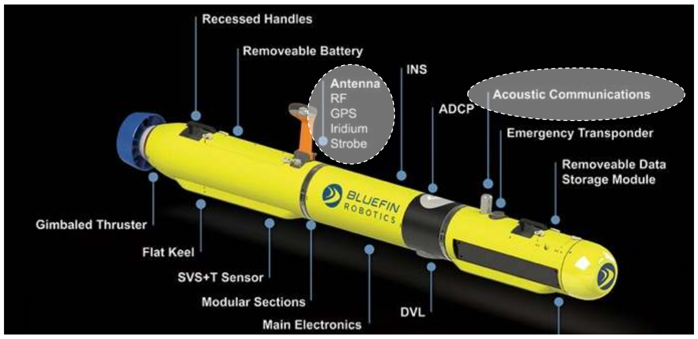
- Depends on how far away they are from each other
- Depends on what is between them (air, water, or both)
- Messages may be sent directly between robots, or over a network


Recovery Strobe Light
GPS/Iridium/WiFi Antenna
Acoustic Modem Transducer
Conductivity, Temperature and Depth Sensor
Recovery Float and Pop-Off Float
HYDROID
Side Scan Sonar
Navigation
Electronics/Batteries
Acoustic Doppler Current Profiler (ADCP)

Inter-Vehicle Communications

How do two vehicles / robots / machines talk to each other?

- Depends on how far away they are from each other
- Depends on what is between them (air, water, or both)
- Messages may be sent directly between robots, or over a network



MITMECHE

Inter-Vehicle Communications

How do two vehicles / robots / machines talk to each other?

- Depends on how far away they are from each other
- Depends on what is between them (air, water, or both)
- Messages may be sent directly between robots, or over a network

Assumptions for now:

- All robots have a unique name with known network address
- Messages may be sent to an individual robot, all robots, or a group of robots
- A message may or may not received by the target robot
- No acknowledgement is built into the messaging structure (although you can do this yourself)
- A message may be range-limited (the receiving robot is too far away)
- A message may be band-width limited (the message has a maximum length)
- A message may be frequency limited (there may be a minimum wait time between messages)
- A message may have latency (it's arrival time at the receiving robot is not guaranteed)

Focus of 2.680

How can we use robot mobility and autonomy to overcome these limitations?

MIT 2.680 Spring 2024 – Marine Autonomy – "Lecture 11: Inter-Vehicle Messaging"

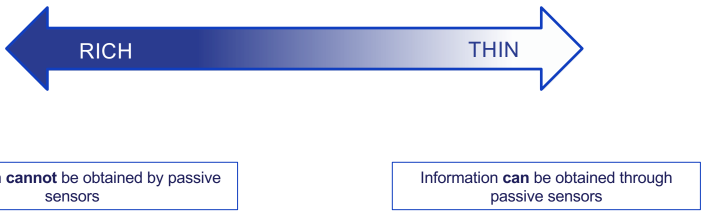
9

MITMECHE

Message Content

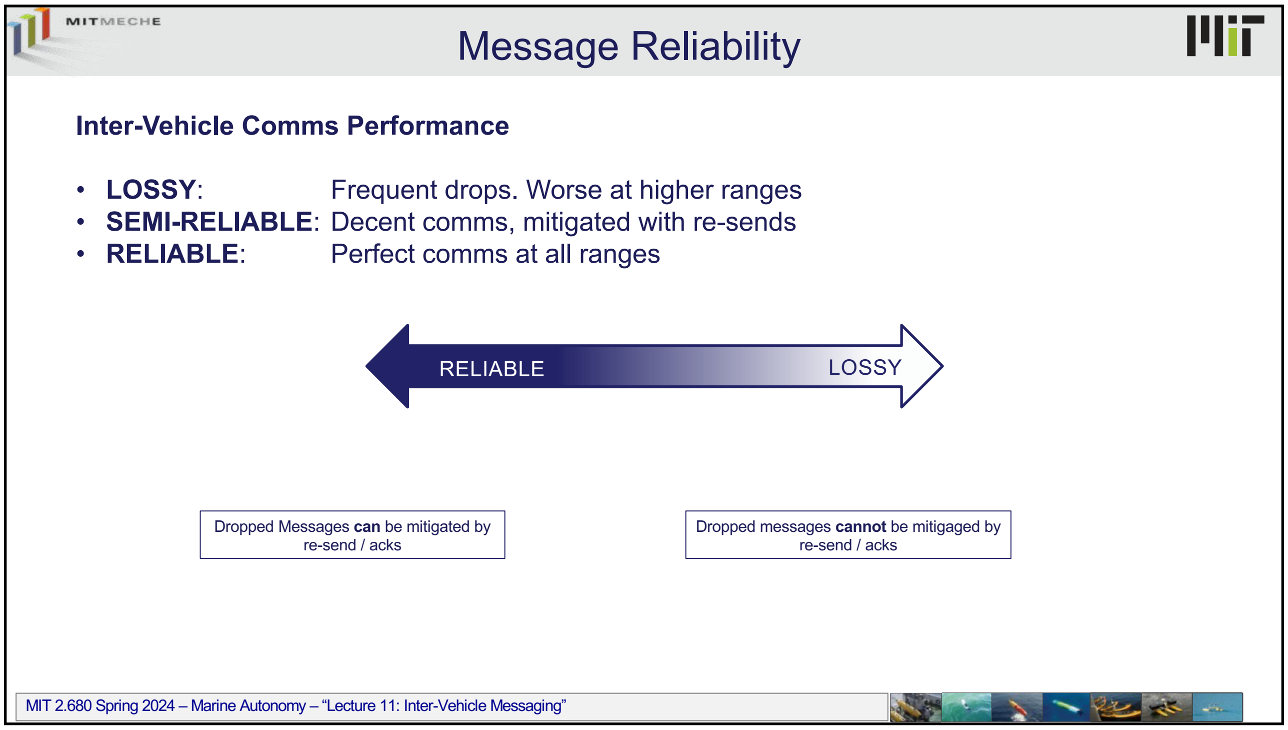
Inter-Vehicle Comms Message Content

- THIN: Position/Pose
- SEMI-RICH: Position/Pose + Status or Intent
- RICH: Unlimited Data Types, plus acknowledgements


RICH
THIN
Information **cannot** be obtained by passive sensors
Information **can** be obtained through passive sensors

MIT 2.680 Spring 2024 – Marine Autonomy – "Lecture 11: Inter-Vehicle Messaging"

10



Message Reliability

Inter-Vehicle Comms Performance

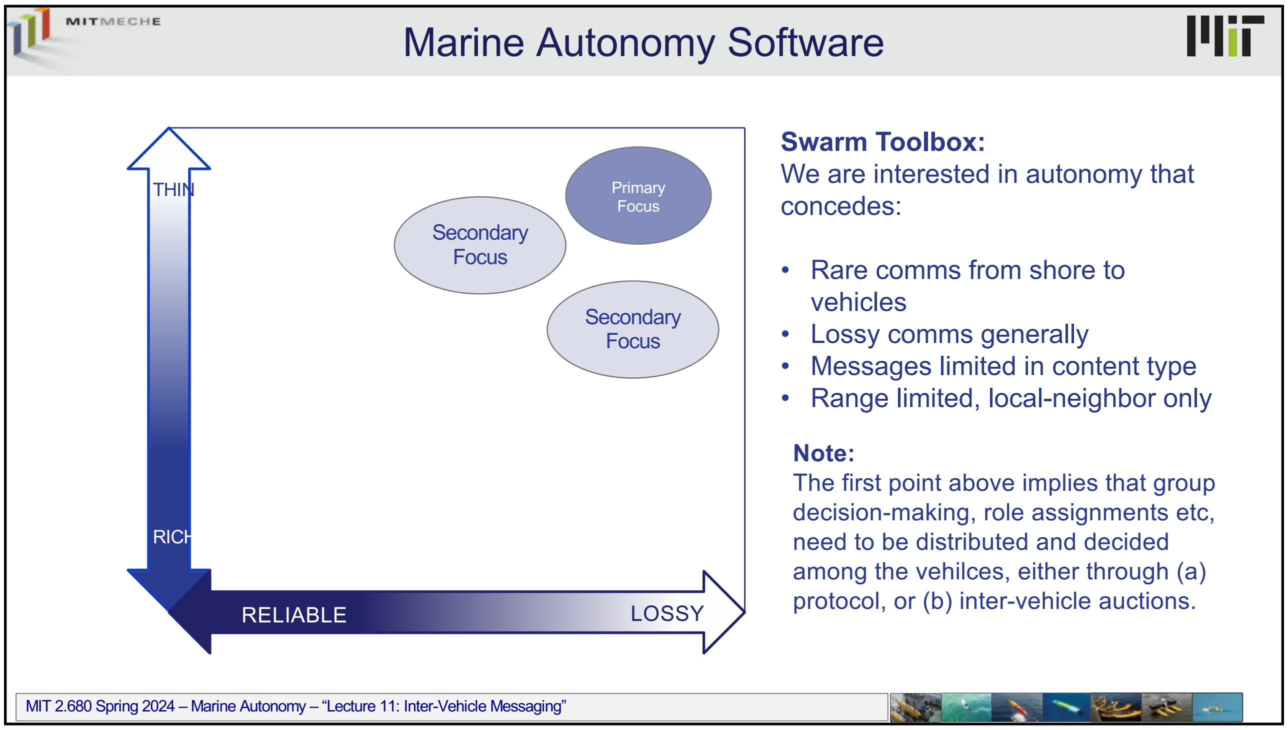
• **LOSSY**:	Frequent drops. Worse at higher ranges
• **SEMI-RELIABLE**:	Decent comms, mitigated with re-sends
• **RELIABLE**:	Perfect comms at all ranges

RELIABLE
LOSSY

Dropped Messages **can** be mitigated by re-send / acks

Dropped messages **cannot** be mitigaged by re-send / acks

11



Marine Autonomy Software

THIN
RICH
Primary Focus
Secondary Focus
Secondary Focus
RELIABLE
LOSSY

Swarm Toolbox:
We are interested in autonomy that concedes:

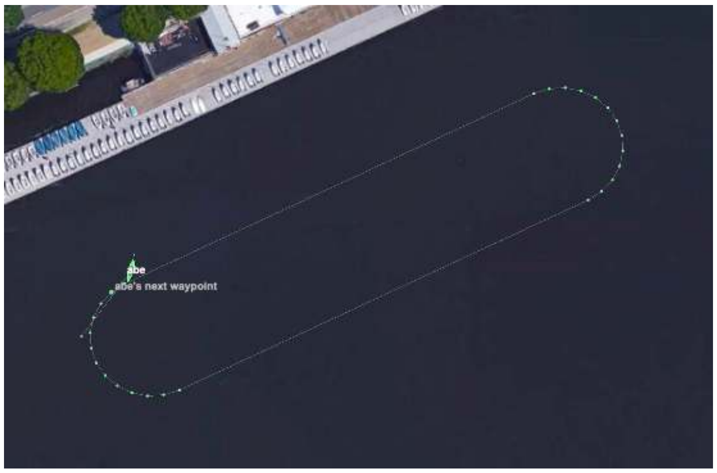
- Rare comms from shore to vehicles
- Lossy comms generally
- Messages limited in content type
- Range limited, local-neighbor only

Note:
The first point above implies that group decision-making, role assignments etc, need to be distributed and decided among the vehilces, either through (a) protocol, or (b) inter-vehicle auctions.

12

Mission Progression

- **Level 0** - single scripted
- **Level 1** - single adaptive
- **Level 2** - collaborative independent
- **Level 3** - collaborative contacts
- **Level 4** - collaborative tactical
- **Level 5** – Collaborative tactical/human



13

Mission Progression

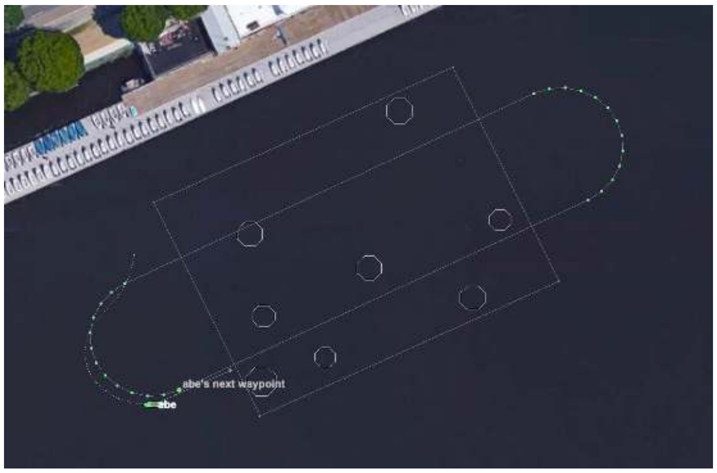
- **Level 0** - single scripted
- **Level 1** - single adaptive
- **Level 2** - collaborative independent
- **Level 3** - collaborative contacts
- **Level 4** - collaborative tactical
- **Level 5** – Collaborative tactical/human

abe
abe's next waypoint

14

Mission Progression

- **Level 0** - single scripted
- **Level 1** - single adaptive
- **Level 2** - collaborative independent
- **Level 3** - collaborative contacts
- **Level 4** - collaborative tactical
- **Level 5** – Collaborative tactical/human



15

Mission Progression

- **Level 0** - single scripted
- **Level 1** - single adaptive
- **Level 2** - collaborative independent
- **Level 3** - collaborative contacts
- **Level 4** - collaborative tactical
- **Level 5** – Collaborative tactical/human

Henry Gilda Baseline

MIT 2.680

16

Mission Progression

- **Level 0** - single scripted
- **Level 1** - single adaptive
- **Level 2** - collaborative independent
- **Level 3** - collaborative contacts
- **Level 4** - collaborative tactical
- **Level 5** – Collaborative tactical/human

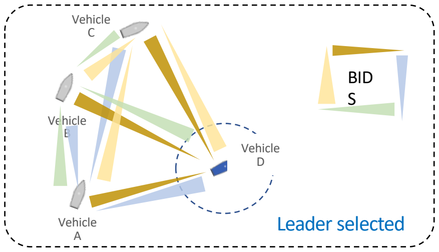
The Berta Mission

(Collision Avoidance)

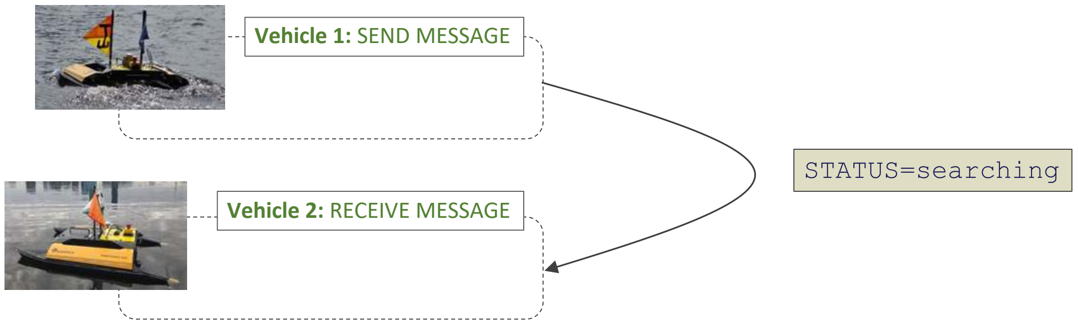
MIT 2.680

Mission Progression

- **Level 0** - single scripted
- **Level 1** - single adaptive
- **Level 2** - collaborative independent
- **Level 3** - collaborative contacts
- **Level 4** - collaborative tactical
- **Level 5** – Collaborative tactical/human



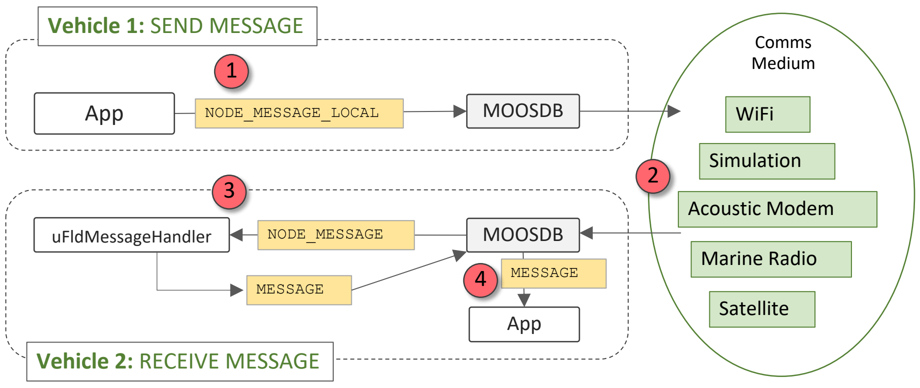
Sending Messages Between Vehicles in MOOS



19

uField Toolbox – Sending a Message Between Vehicles

```
NODE_MESSAGE_LOCAL = "src_node=alpha,dest_node=bravo,
                      var_name=STATUS,string_var=searching"
```



1. App generates a msg to be shared to another vehicle packing it in a NODE MESSAGE LOCAL posting.
2. Msg shared to other vehicle using a comms medium. In sim this is pShare.
3. Packed msg received on destination vehicle and unpacked to simple msg posted to the MOOSDB.
4. The app on destination vehicle meant to be the consumer of the msg, will receive the msg.

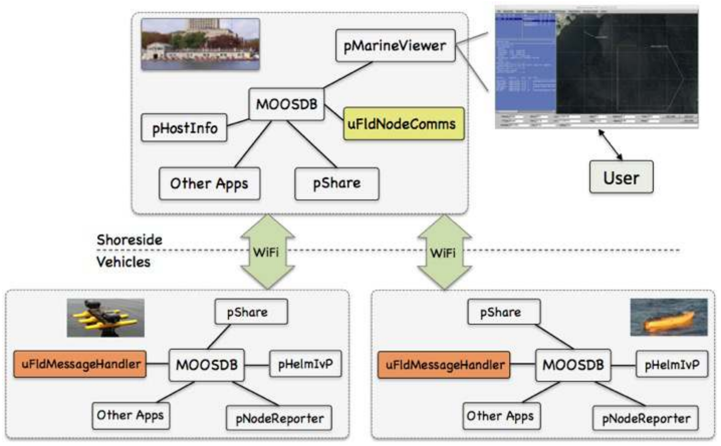
```
NODE_MESSAGE = "src_node=alpha,dest_node=bravo,
                var_name=STATUS,string_var=searching"
```

```
STATUS= "searching"
```

20

uField Toolbox – Sending a Message Between Vehicles

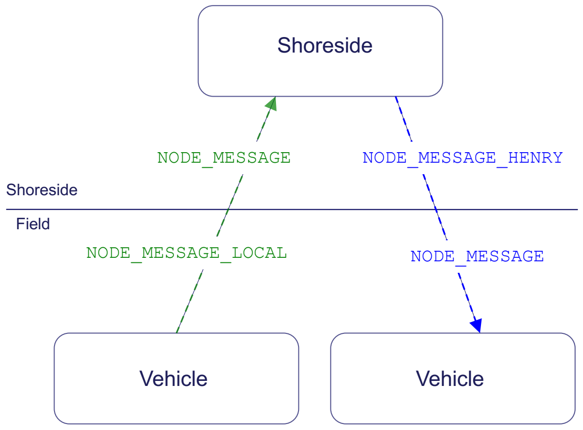
The uFldMessageHandler app is running on all vehicles wishing to receive messages.



21

uField Message Routing

- Message routing is handled on the shoreside.
- But it's not the case that all messages make it through
- They are handled by uFldNodeComms



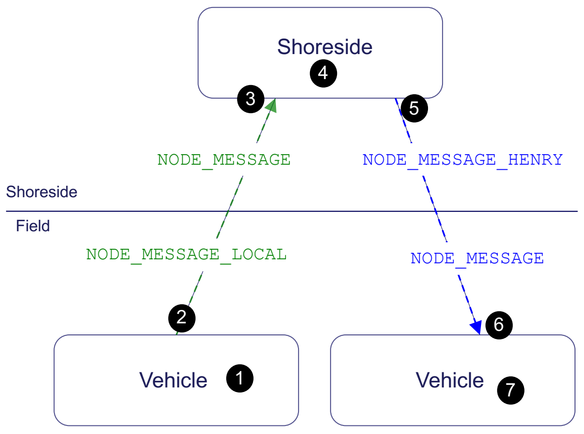
```
ProcessConfig = uFieldShoreBroker
{
   qbridge= NODE_MESSAGE
}
```

```
ProcessConfig = uFieldNodeBroker
{
   bridge = src=NODE_MESSAGE_LOCAL,
                alias=NODE_MESSAGE
}
```

22

uField Message Routing Sequence

- The sequence of events, from the generation of the message all the way to the receipt on the destination robot(s)



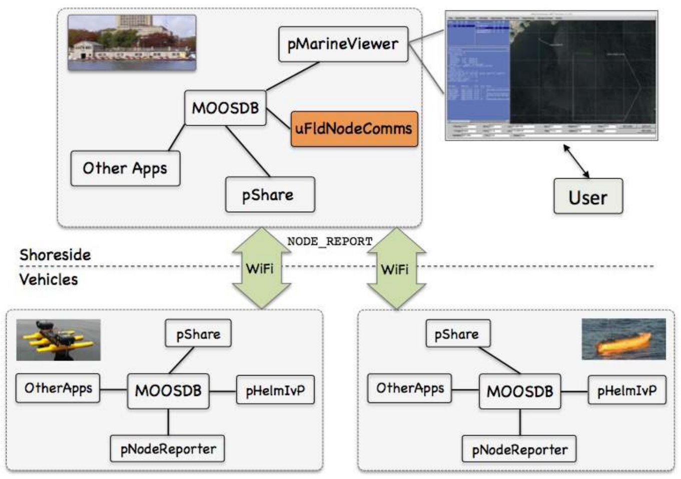
1. Some app on the source vehicle publishes an outoing message in the form of **`NODE_MESSAGE_LOCAL`**
2. The source vehicle shares it via pShare to the Shoreside computer
3. It arrives at Shoreside as **`NODE_MESSAGE`**
4. Shoreside uFldNodeComms examines the message, location of vehicles and other range, bandwidth criteria and may decide to send it.
5. If uFldNodeComms decides to send it, it is published as **`NODE_MESSAGE_VNAME`** which is only shared to the robot named `VNAME`
6. It arrives on the destination vehicle simply as the MOOS variable **`NODE_MESSAGE`**
7. The final message is unpacked by uFldMessageHandler and posted as a MOOS variable-value pair to the local MOOSDB.

The uFldNodeComms App

Typical Application Toplogy

The uFldNodeComms app runs on the shoreside, limits intervehicle messaging.

- It subscribes for the the **`NODE_REPORT`** messages arriving from all vehicles.
- It knows the position of all vehicles.
- It shares vehicle position information to all other vehicles. To support collision avoidance. Separate from message passing.
- It knows the range between any pair of vehicles and may use that to block a message.
- It keeps track of when each vehicle sent its previous message, to perhaps limit message frequency.
- It publishes visual objects for pMarineViewer to indicate comms status.
- It keeps track of all sent and dropped messages for viewing and debugging in its AppCast output, viewable in pMarineViewer.



The uFldNodeComms Basic Configuration

The uFldNodeComms configuration parameters:

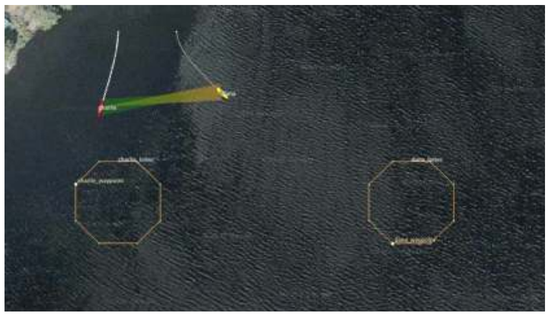
```
ProcessConfig = uFieldNodeComms
{
   comms_range      = 200
   min_msg_interval = 60
   max_msg_length   = 100

   view_node_report_pulses = true

   stale_time       = 5
   groups           = true

   critical_range   = 1000

}
```

Distance in meters between vehicles (default is 100m)

Min time in seconds between messages from a vehicle (default is 30 sec)

Max chars in a string message (default is 1,000 characters)

Boolean indicating whether visual artifacts are to be generated indicating that node reports are being shared between vehicles



The uFldNodeComms Handling Stale Vehicles

The uFldNodeComms configuration parameters:

```
ProcessConfig = uFieldNodeComms
{
   comms_range      = 200
   min_msg_interval = 60
   max_msg_length   = 100

   view_node_report_pulses = true

   stale_time       = 5
   groups           = true

   critical_range   = 1000

}
```

stale_time: Time in seconds after which a vehicle will not receive node reports or messages unless a node report has been received by that vehicle. The default is 5 seconds.

- Since up-to-date inter-vehicle range information is used as part of the criteria in determining whether a vehicle receives a new node report from another, the position of the candidate recipient vehicle needs to reasonably up-to-date.
- If a recipient vehicle becomes stale, it will not receive `NODE_MESSAGE` messages.

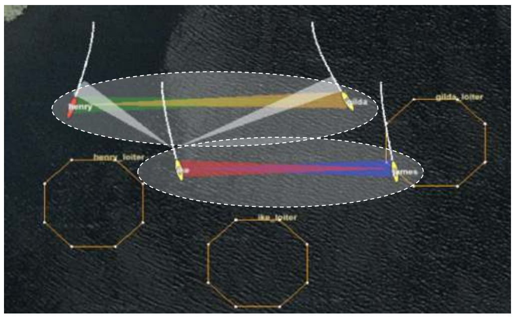
The uFldNodeComms Support for Groups

The uFldNodeComms configuration parameters:

```
ProcessConfig = uFieldNodeComms
{
   comms_range      = 200
   min_msg_interval = 60
   max_msg_length   = 100

   view_node_report_pulses = true

   stale_time       = 5
   groups           = true

   critical_range   = 1000

}
```

groups: If true, inter-vehicle node reports are shared only if two vehicles are in the same group. Default is false.

- The group name is a field contained in the node report itself, so the onus is on the vehicle to include this information as part of its report.
- pNodeReporter can be configured with **`group=<group-name>`** where the group information is declared for inclusion in all node reports.



- Motivation for groups: to support multi-vehicle competitions where some vehicles want to convey positions to teammates, but not adversaries.

The uFldNodeComms Critical Range

The uFldNodeComms configuration parameters:

```
ProcessConfig = uFieldNodeComms
{
   comms_range      = 200
   min_msg_interval = 60
   max_msg_length   = 100

   view_node_report_pulses = true

   stale_time       = 5
   groups           = true

   critical_range   = 1000

}
```

critical range: Range in meters within which inter-vehicle node reports will be shared even if group membership would otherwise disallow. The default is 30 meters.

- When the two vehicles are within a range deemed critical, as set by the **critical_range** configuration parameter, node reports are shared between vehicles regardless of the comms_range parameter and the groups parameter.
- The default for this parameter is 30 meters.
- The thought behind this feature is that, while it may be advantageous to not broadcast your own vehicle position to non group members for the purposes of a competition, it may be a good idea to share this information for the sake of collision avoidance.

The uFldMessageHandler Configuration

The uFldMessageHandler configuration parameters:

```
ProcessConfig = uFieldMessageHandler
{
   strict_addressing = false

   appcast_trunc_msg = 60
}
```

strict_addressing: If true, only messages with a destination specified by dest_node, matching the local community name are processed. Other messages with a destination specified by a group designation are ignored. The default is false.

appcast_trunc_msg: Number of characters allowed in the appcast report for each line reporting a successful message. The default is 75. Setting it to zero means no truncating will be applied.

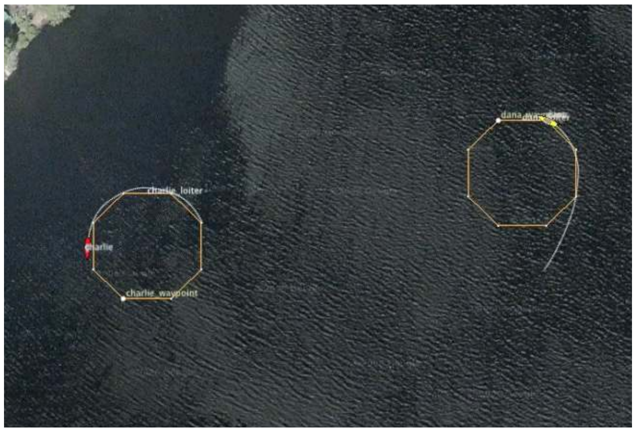
29

Signs of Healthy Messaging

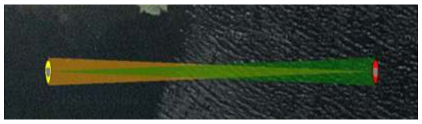
30

Visual Signs of Healthy Messaging

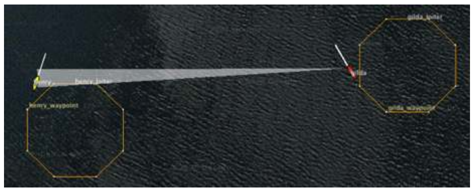
From the charlie_dana_messaging mission in Lab 9



NODE_REPORT messages are being shared



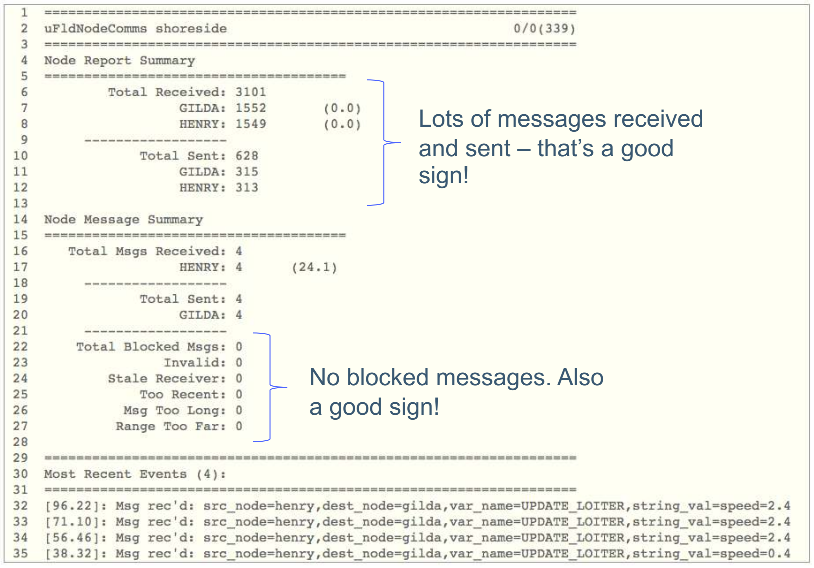
NODE_MESSAGE messages are being shared



AppCasting Signs of Healthy Messaging

Status of uFldNodeComms is contained in its AppCast output.

There are several fields confirming healthy messaging.



```
==================================================================
uFldNodeComms shoreside                                    0/0(339)
==================================================================
Node Report Summary
======================================
        Total Received: 3101
                 GILDA: 1552       (0.0)
                 HENRY: 1549       (0.0)
     ------------------
            Total Sent: 628
                 GILDA: 315
                 HENRY: 313

Node Message Summary
======================================
   Total Msgs Received: 4
                 HENRY: 4      (24.1)
     ------------------
            Total Sent: 4
                 GILDA: 4
     ------------------
    Total Blocked Msgs: 0
               Invalid: 0
        Stale Receiver: 0
            Too Recent: 0
          Msg Too Long: 0
        Range Too Far: 0

==================================================================
Most Recent Events (4):
==================================================================
[96.22]: Msg rec'd: src_node=henry,dest_node=gilda,var_name=UPDATE_LOITER,string_val=speed=2.4
[71.10]: Msg rec'd: src_node=henry,dest_node=gilda,var_name=UPDATE_LOITER,string_val=speed=2.4
[56.46]: Msg rec'd: src_node=henry,dest_node=gilda,var_name=UPDATE_LOITER,string_val=speed=2.4
[38.32]: Msg rec'd: src_node=henry,dest_node=gilda,var_name=UPDATE_LOITER,string_val=speed=0.4
```

Lots of messages received and sent – that's a good sign!

No blocked messages. Also a good sign!

AppCasting Signs of Healthy Messaging

Status of uFldMessageHandler is contained in its AppCast output

- Totals valid messages
- Invalid messages are ill-formed
- Rejected messages failed one of the range, bandwidth etc. criteria

  - Per source summary, one line per other robot.

- Finite list of most recent messages.
- Automatically truncated in number.
- Truncated in length as per set by the user with the appcast_trunc_msg parameter.

```
==================================================================
uFldMessageHandler gilda 0/0(841)
==================================================================
Overall Totals Summary
=====================================
Total Received Valid: 3
Invalid: 0
Rejected: 0
Time since last Msg: 101.3

Per Source Node Summary
=====================================
Source Total Elapsed Variable Value
------ ----- ------- -------- -----
henry  3     101.3   RETURN   true

Last Few Messages: (oldest to newest)
=====================================
Valid Mgs:
  src_node=henry,dest_node=gilda,var_name=UPDATE_LOITER,string_val=speed
  src_node=henry,dest_node=gilda,var_name=UPDATE_LOITER,string_val=speed
  src_node=henry,dest_node=gilda,var_name=RETURN,string_val=true
Invalid Mgs:
  NONE
Rejected Mgs:
  NONE
```

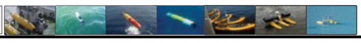
ALog Signs of Healthy Messaging

We can check the log files *on the Shoreside:*

```
$ cd LOG_SHORESIDE_18_3_2018_____16_36_17
$ aloggrep *.alog NODE_MESSAGE_CHARLIE NODE_MESSAGE_DANA
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% LOG FILE:       ./LOG_SHORESIDE_18_3_2018_____16_36_18/LOG_SHORESIDE_18_3_2018_____16_36_18.alog
%% FILE OPENED ON  Wed Dec 31 19:00:00 1969
%% LOGSTART               22821080675.4
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
120.863         NODE_MESSAGE_CHARLIE uFldNodeComms   src_node=dana,dest_node=all,var_name=UP_LOITER,string_val=ycenter_assign=-43.075
127.423         NODE_MESSAGE_DANA    uFldNodeComms   src_node=charlie,dest_node=all,var_name=UP_LOITER,string_val=ycenter_assign=-19.65
553.160         NODE_MESSAGE_CHARLIE uFldNodeComms   src_node=dana,dest_node=all,var_name=UP_LOITER,string_val=ycenter_assign=-24.35
572.299         NODE_MESSAGE_DANA    uFldNodeComms   src_node=charlie,dest_node=all,var_name=UP_LOITER,string_val=ycenter_assign=-69.675
Total lines retained: 9 (0.01%)
Total lines excluded: 60558 (99.99%)
Total chars retained: 839 (0.01%)
Total chars excluded: 9972138 (99.99%)
  Variables retained: (2) NODE_MESSAGE_CHARLIE, NODE_MESSAGE_DANA
```

`NODE_MESSAGE_CHARLIE` entries indicate outgoing messages to charlie. In this case we can also see they are coming from dana.



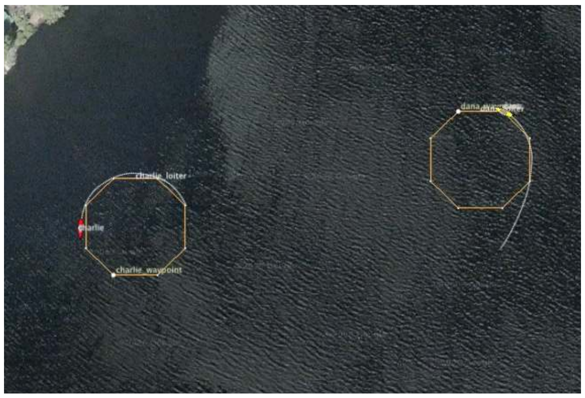
When Things Go Wrong

(How to Debug)

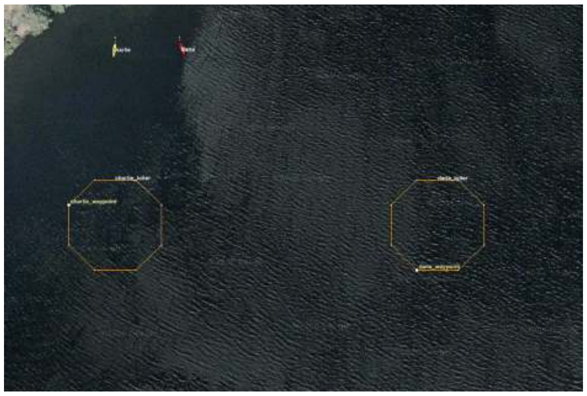
35

When Things Go Wrong

You were expecting:



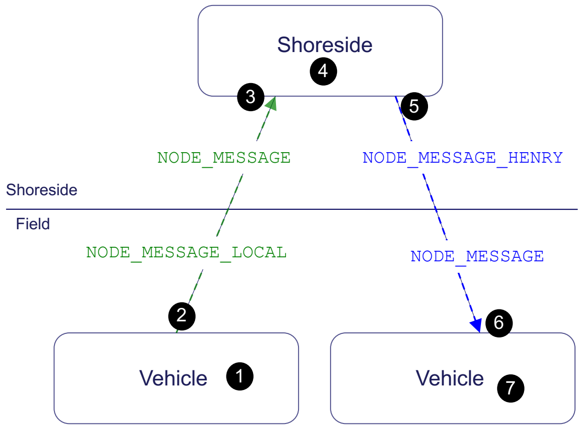
But you’re seeing this instead (no comms pulses)



36

Debugging Broken Messaging

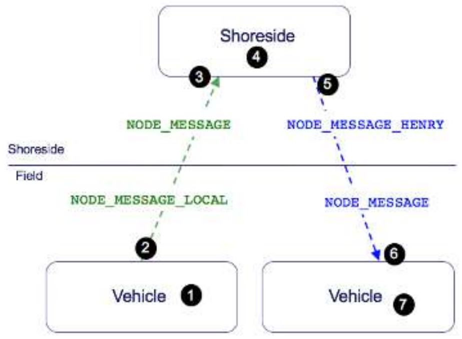
- Re-visiting the message passing "pipeline":



1. Some app on the source vehicle publishes an outoing message in the form of **NODE_MESSAGE_LOCAL**
2. The source vehicle shares it via pShare to the Shoreside computer
3. It arrives at Shoreside as **NODE_MESSAGE**
4. Shoreside uFldNodeComms examines the message, location of vehicles and other range, bandwidth criteria and may decide to send it.
5. If uFldNodeComms decides to send it, it is published as **NODE_MESSAGE_VNAME** which is only shared to the robot named VNAME
6. It arrives on the destination vehicle simply as the MOOS variable **NODE_MESSAGE**
7. The final message is unpacked by uFldMessageHandler and posted as a MOOS variable-value pair to the local MOOSDB.

Debugging Broken Messaging (Stage 1)

- Re-visiting the message passing "pipeline":



1. Some app on the source vehicle publishes an outoing message in the form of **NODE_MESSAGE_LOCAL**

```
-----------------------------------
DEBUGGING STEPS
-----------------------------------
```

- Was **NODE_MESSAGE_LOCAL** ever actually posted to the MOOSDB on the vehicle?
- You can check while running the mission by running a scope:

```
$ uXMS mission.moos NODE_MESSAGE_LOCAL
```

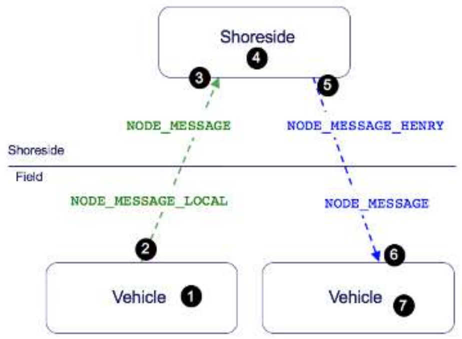
- You can check after running the mission by examining the alog file:

```
$ aloggrep file.alog NODE_MESSAGE_LOCAL
```

- If it was never posted, re-examine what was supposed to generate this posting.

Debugging Broken Messaging (Stage 2/3)

- Re-visiting the message passing "pipeline":



2 The source vehicle shares it via pShare to the Shoreside computer

3 It arrives at Shoreside as **`NODE_MESSAGE`**

```
-----------------------------------
DEBUGGING STEPS
-----------------------------------
```

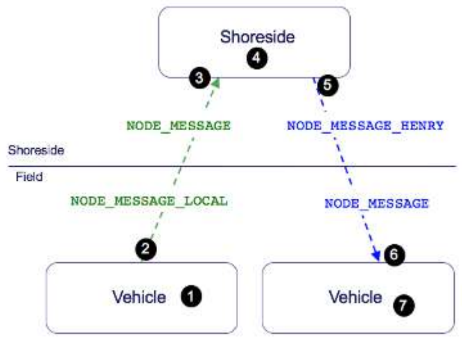
- Did **`NODE_MESSAGE`** arrive in the Shoreside?
- You can check after running the mission by examining the *Shoreside* alog file:

```
$ aloggrep shoreside.alog NODE_MESSAGE
```

- If it was never posted, things to check:
- Was pShare running on vehicle? Shoreside?
- Did the vehicle uFldNodeBroker config block include sharing for **`NODE_MESSAGE_LOCAL`**?

Debugging Broken Messaging (Stage 4)

- Re-visiting the message passing "pipeline":



4 Shoreside uFldNodeComms examines the message, location of vehicles and other range, bandwidth criteria and may decide to send it.

```
-----------------------------------
DEBUGGING STEPS
-----------------------------------
```

- In this stage uFldNodeComms will ingest a **`NODE_MESSAGE`** and post a **`NODE_MESSAGE_VNAME`** if all goes well. Was **`NODE_MESSAGE_VNAME`** `posted?`
- You can check after running the mission by examining the *Shoreside* alog file:

```
$ aloggrep shoreside.alog NODE_MESSAGE_HENRY
```

- If it was never posted, things to check:
- Was the message blocked because it was ill-formed?
- Was the message blocked due to range between vehicles?
- Was the message blocked due to message length?
- Was the message blocked due to a stale receiving vehicle?
- Was the message blocked due to frequency constraints?

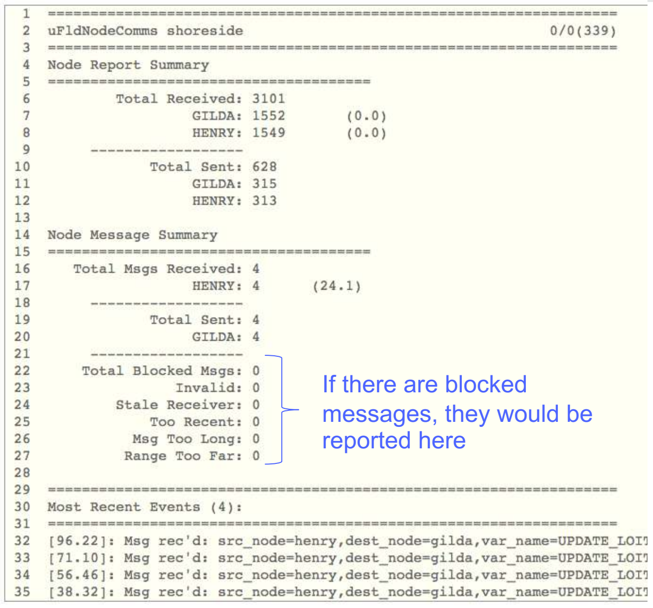
For debugging blocked messages, the AppCasting output of uFldNodeComms is your most powerful debugging tool.

Debugging Blocked Messages at the Shoreside

A *blocked message* at the Shoreside is a one where uFldNodeComms has ingested a **`NODE_MESSAGE`** but has not made a corresponding **`NODE_MESSAGE_VNAME`** post.

Possible reasons for blocking:

- The message was ill-formed.
- The message was blocked due to range between vehicles.
- The message was blocked due to message length.
- The message was blocked due to frequency constraints. (too soon since the previous successful message)
- The message was blocked due to a stale receiver vehicle, or the receiver vehicle is not known to uFldNodeComms.
- Re-run the mission and check the AppCast output of uFldNodeComms (see right).
- As of now, uFldNodeComms does not produce similar output to debugging MOOS variables for logging.

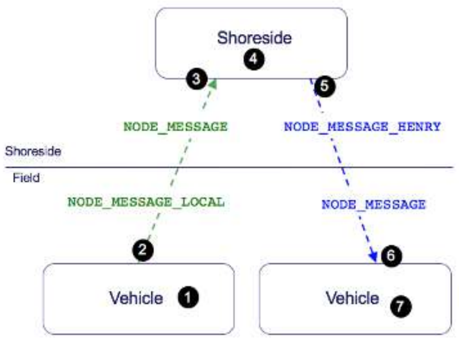


```
======================================================================
uFldNodeComms shoreside                                      0/0(339)
======================================================================
Node Report Summary
======================================
        Total Received: 3101
                 GILDA: 1552       (0.0)
                 HENRY: 1549       (0.0)
     ------------------
            Total Sent: 628
                 GILDA: 315
                 HENRY: 313

Node Message Summary
======================================
   Total Msgs Received: 4
                 HENRY: 4      (24.1)
     ------------------
            Total Sent: 4
                 GILDA: 4
     ------------------
    Total Blocked Msgs: 0
               Invalid: 0
        Stale Receiver: 0
            Too Recent: 0
          Msg Too Long: 0
        Range Too Far: 0

======================================================================
Most Recent Events (4):
======================================================================
[96.22]: Msg rec'd: src_node=henry,dest_node=gilda,var_name=UPDATE_LOIT
[71.10]: Msg rec'd: src_node=henry,dest_node=gilda,var_name=UPDATE_LOIT
[56.46]: Msg rec'd: src_node=henry,dest_node=gilda,var_name=UPDATE_LOIT
[38.32]: Msg rec'd: src_node=henry,dest_node=gilda,var_name=UPDATE_LOIT
```

If there are blocked messages, they would be reported here

Debugging Broken Messaging (Stage 5/6)

- Re-visiting the message passing "pipeline":



5 If uFldNodeComms decides to send it, it is published as **`NODE_MESSAGE_VNAME`** which is only shared to the robot named `VNAME`

6 It arrives on the destination vehicle simply as the MOOS variable **`NODE_MESSAGE`**

```
-----------------------------------
DEBUGGING STEPS
-----------------------------------
```

- uFldNodeComms has published a **`NODE_MESSAGE_VNAME`** and it should have resulted in **`NODE_MESSAGE`** on the vehicle.
- You can the *vehicle* alog file:

```
$ aloggrep vehicle.alog NODE_MESSAGE
```

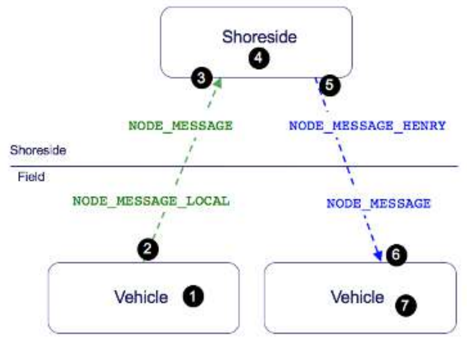
If it was never posted, things to check:

- Was pShare running on the Shoreside
- Was pShare running on the vehicle?
- If you were able to deploy the vehicles and see their positions updated on pMarineViewer, then very likely pShare was running on both vehicles.
- Was the Shoreside pShare configured to share **`NODE_MESSAGE_VNAME`** and to **`NODE_MESSAGE`**? Check the configuration block for uFldShoreBroker and look for a configuration like like:

```
qbridge = NODE_MESSAGE
```

Debugging Broken Messaging (Stage 7)

- Re-visiting the message passing "pipeline":



7 The final message is unpacked by uFldMessageHandler and posted as a MOOS variable-value pair to the local MOOSDB.

```
-----------------------------------
DEBUGGING STEPS
-----------------------------------
```

- A **`NODE_MESSAGE`** has arrived on the vehicle, but the contents of the message have not been posted.
- Again, you can verify that **`NODE_MESSAGE`** has been received on the vehicle by checking the *vehicle* alog file:

```
$ aloggrep vehicle.alog NODE_MESSAGE
```

If the contents of the message was not posted, things to check:

- The message was invalid (ill-formed syntactically)
- The message was rejected, perhap because the "addressee" was set to "all", and message handler was configured to require strict matching of vehicle name.
- **For debugging blocked messages, the AppCasting output of uFldMessageHandler is your most powerful debugging tool.**

Debugging Broken Messaging (Stage 7)

Possible reasons for unposted messages from an incoming NODE_MESSAGE on a vehicle:

- The message was invalid (ill-formed syntactically)
- The message was rejected, perhap because the "addressee" was set to "all", and message handler was configured to require strict matching of vehicle name.

```
===============================================================
uFldMessageHandler gus 0/0(841)
===============================================================
Overall Totals Summary
======================================
Total Received Valid: 3
Invalid: 0
Rejected: 0
Time since last Msg: 101.3

Per Source Node Summary
======================================
Source Total Elapsed Variable Value
------ ----- ------- -------- -----
hal    3     101.3   RETURN   true

Last Few Messages: (oldest to newest)
======================================
Valid Mgs:
  src_node=hal,dest_node=gus,var_name=UP_LOITER,string_val=speed
  src_node=hal,dest_node=gus,var_name=UP_LOITER,string_val=speed
  src_node=hal,dest_node=gus,var_name=RETURN,string_val=true
Invalid Mgs:
  NONE
Rejected Mgs:
  NONE
```

If there are invalid or rejected messages, they would be reported here

Contents of recent invalid or rejected messages, are shown here

MITMECHE

Lab 9 Preview

Inter-vehicle Messaging

In today's lab, we will build a simple two-vehicle mission:

- Each vehicle is loitering in its half of an east-west op-area.
- Each vehicle periodically sends a message to the other vehicle to switch its region

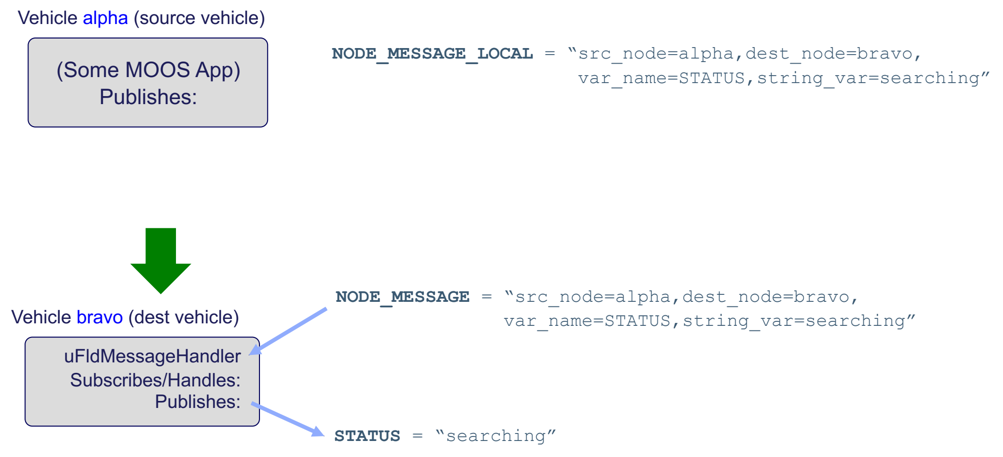
45

MITMECHE

END

46

uField Toolbox – Sending a Message Between Vehicles



Vehicle alpha (source vehicle)

(Some MOOS App)
Publishes:

```
NODE_MESSAGE_LOCAL = "src_node=alpha,dest_node=bravo,
                      var_name=STATUS,string_var=searching"
```

Vehicle bravo (dest vehicle)

uFldMessageHandler
Subscribes/Handles:
Publishes:

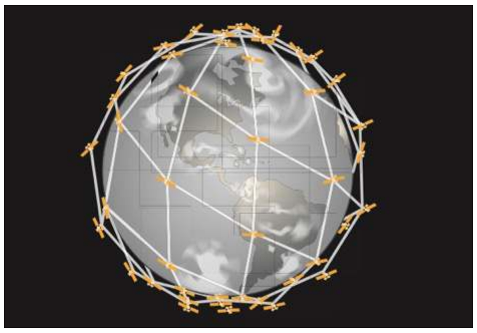
```
NODE_MESSAGE = "src_node=alpha,dest_node=bravo,
                var_name=STATUS,string_var=searching"
```

```
STATUS = "searching"
```

47

Iridium Satellites

- Iridium's constellation consists of 66 low-earth orbiting (LEO), cross-linked satellites operating as a fully meshed network and supported by multiple in-orbit spares. Iridium has gateways in Arizona and Hawaii and additional telemetry, tracking and control facilities in Alaska, Canada and Norway. It is the largest commercial satellite constellation in the world [1].



[1] http://www.wlnet.com/pdfs/OP-Install.pdf

48