pMediator: Mediating Inter-Vehicle Messages

Nov 2023

Michael Benjamin, mikerb@mit.edu
Department of Mechanical Engineering
MIT, Cambridge MA 02139
`project-pavlab/appdocs/app_pmediator`

1 Overview

The `pMediator` application is a tool for mediating messages between vehicles in situations without guaranteed message arrivals. Outgoing messages may request a return acknowledgement message (an "ack"), and periodically repeat the message until an ack is received or time limit or re-send limit has been reached. The policy for repeating a message may be configured individually per message type to account for vital messages and nice-to-have messages. The `pMediator` app runs on each vehicle and stores local information to handle outgoing and incoming messages. Handling of incoming messages involves both sending return acknowledgement messages and ensuring that no incoming message is posted twice.

The setup overview is depicted in Figure 1:

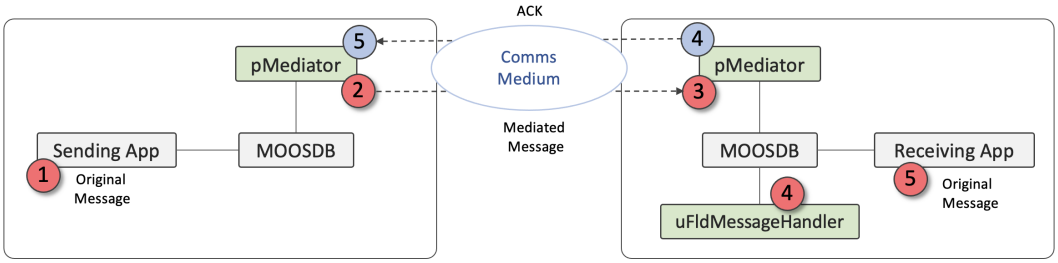

Figure 1: **Mediated Messaging:** An outgoing message to another node originates within any application. The `pMediator` app adds a tracking ID. An app specific to whichever comms medium is in use will send the mediated message to the destination vehicle(s). A `pMediator` app on the receiving vehicle will process the message by (a) sending an acknowledgement message back to the sender, and (b) posting the incoming message locally. The incoming message will be unpacked by the `uFldMessageHandler` app, resulting in a MOOS variable publication received by the destination appp.

The addition of `pMediator` to our autonomy toolbox is based on the following observations and assumptions:

- Colloborative autonomy software needs to assume that inter-vehicle communications will be lossy, resulting dropped, delayed or out-of-order messages. This includes comms between the shoreside comand-and-control and vehicles.
- Duplicated messages can be reliably prevented but arguably should also be accounted for and tested.
- The `pMediator` should not only improve but also monitor and measure inter-vehicle message performance.

Complementing `pMediator`, our simulation environment should also be able to simulate lossy comms with a variety of manners of performance drop-off, based on inter-vehicle range, message rate, and message size. The existing app, `uFldNodeComms`, is augmented to simulate lossy comms and support the `pMediator` inter-vehicle messaging scheme.

2 Overview Issues with Inter-Vehicle Messaging

Messages may be sent between vehicles in many different manners, e.g, WiFi, satellite, underwater acoustic modems and so on. The issues addressed with `pMediator` are independent of *how* messages are sent. Instead it is concerned with *what* messages are sent, whether or not their receipt by the receiver can be verified, and what to do if it is not received.

First it is worth reminding ourselves that not all messages are the same with respect to their importance and with respect to the policy of handling dropped messages. They can be lumped into three general categories as depicted in Figure 2

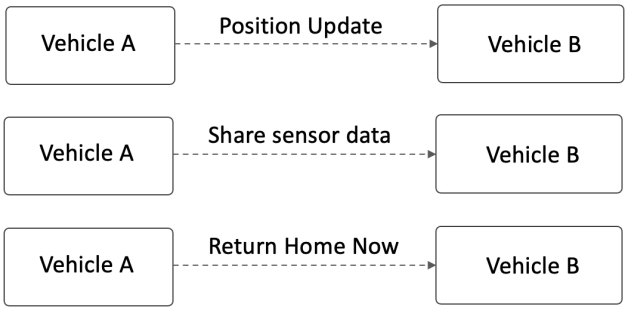

Figure 2: **Categories of Messages:** Messages may vary from (a) fire-and-forget, (b) repeat if bandwidth allows, and (c) critical, to be repeated perhaps an unlimited number of times until an acknowledgement is received.

For messages that are important enough, an acknowledgment return message can be requested (an "ack" for short). As depicted below, if the acknowledgement has been received, the message is considered successful. An unsuccessful message may be due to either the original message being dropped, or the acknowledgment message being dropped.

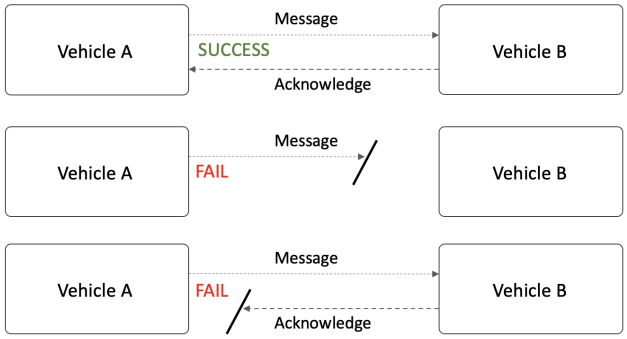

Figure 3: **Messages with Acknowledgements:** Using an acknowledgement return message, the sender can be assured that the mesage was received (top). The lack of an ack may be due to the failure of the original message (middle) or the failure of the acknowledgment message to reach the sender (bottom).

3 Vanilla Inter-Vehicle Messaging

Inter-vehicle message passing in the MOOS-IvP environment follows a *convention* with two parts. First, all inter-vehicle messages are wrapped in the MOOS variable `NODE_MESSAGE_LOCAL`, arriving at the destination vehicle(s) as `NODE_MESSAGE`. The `uFldMessageHandler` unpacks the arriving message and posts the intended MOOS variable and value. Second, message passing is achieved through the `pShare` application between vehicles, during which the name of the messaged changes from `NODE_MESSAGE_LOCAL` to simply `NODE_MESSAGE`.

Either or both parts of this convention may be swapped out and replaced with some other implementation. The idea is shown in Figure 4 below.

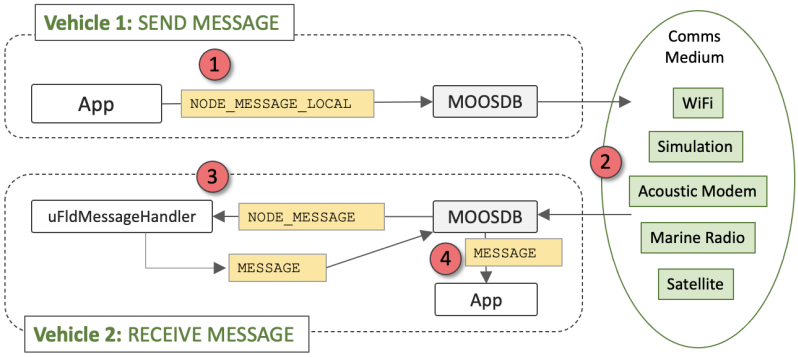

Figure 4: **A Typical Message Path:** (1) An application generates a messaged to be shared to another vehicle by packing it within a NODE_MESSAGE_LOCAL posting. (2) The message is shared to the other vehicle using some comms medium. In simulation or in WiFi this is done with the pShare app. (3) The packed message is received on the destination vehicle and unpacked to a simple message posted to the vehicle's local MOOSDB. (4) The app on the destination vehicle meant to be the consumer of this message, will finally receive the message.

As we will see, the pMediator app will augment this convention with another stage to handle acknowledgments and repeating real or perceived dropped messages.

3.1 Structure of an Outgoing Inter-Vehicle Message

Each outgoing inter-vehicle message is originated by posting to the variable NODE_MESSAGE_LOCAL. This message has four key parts:

- *Source Node:* The name of the vehicle originating the message
- *Destination Node:* The name of the destination vehicle
- *Variable Name:* The MOOS variable of the message
- *Variable Value:* The value of the MOOS variable for the message

Below are a couple example messages:

```
NODE_MESSAGE_LOCAL = src_node=abe, dest_node=ben, var_name=RETURN, string_val=true
NODE_MESSAGE_LOCAL = src_node=eve, dest_node=fin, var_name=DIST, double_val=1984
```

Since MOOS messages are either numerical (type double) values or string values, the value component of a NODE_MESSAGE_LOCAL posting has either the field `double_val` or `string_var` as in the examples above.

Note that a message is sent without naming the app on the receiving vehicle. Like any other interaction in a publish-subscribe architecture, the receiving app receives this information only because it registered for mail on this variable.

3.2 Messages Sent to Groups of Other Vehicles

The sender of a message is an app that publishes the original outgoing message NODE_MESSAGE_LOCAL as described above. It is not uncommon for the sender to want to send the message to "everyone", or "everyone in my group". Such designations are generally supported in outgoing messages.

The first of the following two messages request the message is sent to ”all” other vehicles, and the second requests that all vehicles in the group ”blue” receive the message.

```
NODE_MESSAGE_LOCAL = src_node=abe, dest_node=all, var_name=RETURN, string_val=true
NODE_MESSAGE_LOCAL = src_node=abe, dest_group=blue, var_name=RETURN, string_val=true
```

The questions remain:

- What is the list of ”all” vehicles or the list of vehicles in group ”blue”?
- Where does this information come from? Another app? Configuration parameters?
- How does the message get routed to these vehicles specifically?

4 The Role of pMediator

A design goal of pMediator is that it can be enabled alongside the ”normal” messaging conventions, i.e., requiring no changes to how messages are generated by the sender, unpacked by the receiver, or related at all to the inter-vehicle messaging medium. The design of pMediator is described here:

4.1 The Role of pMediator in Sending Outgoing Messages

When pMediator is used, outgoing messages generated by any app are constructed in the usual manner, through the generation of a NODE_MESSAGE_LOCAL publication. Rather than exporting this message directly to other vehicles, e.g., using pShare, this message is instead read by pMediator, as shown in Figure 5. Upon receipt of NODE_MESSAGE_LOCAL, a new message is created with augmented information, by pMediator. The new message uses the MOOS variable MEDIATED_MESSAGE_LOCAL, and adds an ID unique to each outgoing message. It is comprised of the source vehicle name and an integer incremented upon each message. This mediated message (not NODE_MESSAGE_LOCAL) is then shared offboard to the destination node(s).

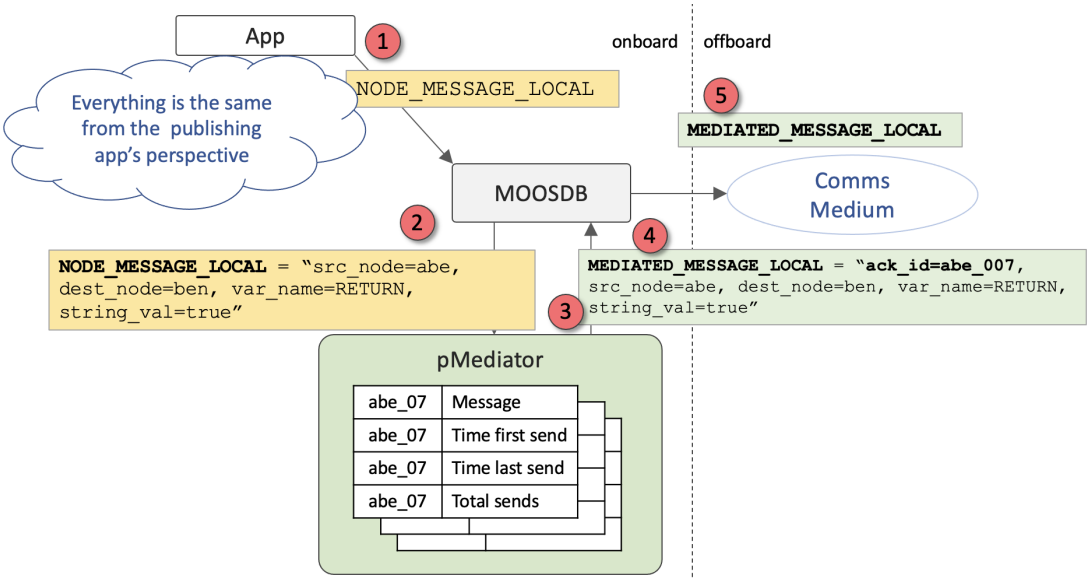

Figure 5: **Messages Sent with pMediator:** (1) The outgoing message is generate by the sending app as normal. (2) The original outgoing messaged is received by `pMediator`. (3) An internal table is augmented with the new message, assigning the message a unique ID field. (4) The outgoing message is replaced with `MEDIATED_MESSAGE_LOCAL`, augmented with the unique ID. (5) The mediated message is sent off-board to the destination node(s).

A table of outgoing messages is maintained by `pMediator`, keyed on the unique message identifier. For messages where an acknowledgment return message is requested, this table is used for keeping track of which messages have been acknowledged. The table also keeps track of the original message content in case it needs to be re-sent. It also holds the time since the message was last sent, for each message, to allow control over the wait time between re-sending messages. A total count of message sends is maintained for each message for cases where the number of re-send attempts is limited.

4.2 The Role of pMediator in Receiving Incoming Messages

When pMediator is used, incoming offboard messages from other nodes arrive in the variable MEDIATED_MESSAGE, perhaps via the local pShare app or another comms medium. The only app registering for this variable is pMediator. Upon receipt, it will unpack the message into a NODE_MESSAGE, removing any ack information from the mediated messge. If the mediated message had contained the field `ack=true`, then pMediator will also generate a return acknowledgement message in the variable ACK_MESSAGE_LOCAL. See Figure 6.

Not all messages require an ack return message. On the sender side, pMediator can be configured to name one or more MOOS variable messages that do not require and acknowledgment. Regardless of whether an ack is requested, the `ack_id` field will be included in all mediated messages. This ensures the uniqueness of each message, and allows the receiving pMediator to ensure that no message will be published locally twice.

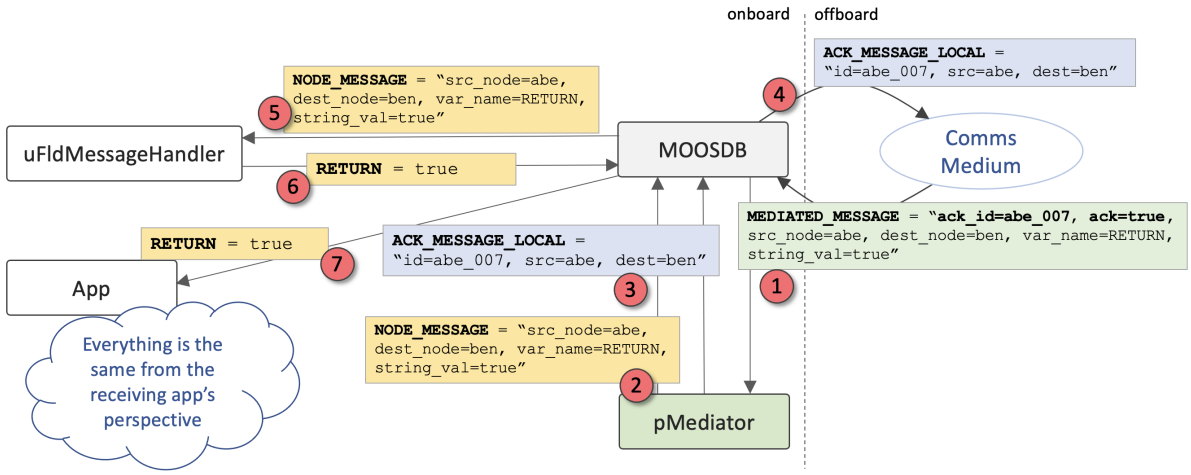

Figure 6: **Messages Received with pMediator:** (1) The incoming message is received as a MEDIATED_MESSAGE, to the local MOOSDB and received by pMediator. (2) The mediated message is unpacked, removing any `ack` fields, to create a NODE_MESSAGE published to the local MOOSDB. (3) If `ack=true` was contained in the mediated message, an acknowledgement message is generated by the mediator to be delivered back to the sending node (4) via the same comms medium. (5) The NODE_MESSAGE is unpacked by uFldMessageHandler, and (6) posted with the intended MOOS variable and value to the local MOOSDB. (7) Finally the original message, RETURN=true is consumed by the intended app on the receiving node.

After pMediator has unpacked the mediated message and generated and posted NODE_MESSAGE to the local MOOSDB, the normal (unmediated) inter-vehicle messaging pipeline continues. The node message is unpacked by uFldMessageHandler and the actual contained message is published to the local MOOSDB where presumably some client application is waiting to receive this information.

5 Handling Dropped Messages

By default, pMediator will request an acknowledgement message for each outgoing message, indicated by the inclusion of `ack=true` in the message. See Figure 6. Messages that do not received an ack message in return, may be re-sent to the node that did not acknowledge receipt. The ack and re-send policy has a number of configuration options.

Note: If a message is being sent to a *group* of vehicles, a distinct message is created and tracked for destination vehicle. For example if a message is being sent from `abe` to `ben` and `cal`, two distinct ack IDs will be created, e.g., `abe_123` and `abe_124`. If an ack is received from `ben` but not `cal`, the message will be resent only to `cal`.

5.1 Messages that Do Not Require and Acknowledgment

Some inter-vehicle messages may not require an acknowledgement. Typically such messages are *status* messages that are sent on a regular interval with new messages obviating old (or dropped) messages. In this case the message variable is named with the configuration parameter:

```
no_ack_vars = FUEL_STATUS, PROP_STATUS
```

The parameter accepts a comma-separated list of MOOS variables, with perhaps multiple configuration lines. Duplicates are simply ignored.

5.2 Setting an Upper Limit on the Number of Re-send Attempts

By default, a message will be re-sent a maximum of 5 times before giving up. This can be changed with the configuration parameter, `max_tries`. For example:

```
max_tries = 10
```

If a variable is on the `no_ack_vars` list, then no further attempts to send will be made after the first attempt, and this parameter does not apply to that variable.

5.3 Time Interval Between Re-send Attempts

If a message is being sent with an ack requested, a re-send will be attempted if no ack has been received after a configured interval. The default for this interval is 2 seconds and may be customized, for example to 5 seconds, with:

```
resend_thresh = 5
```

Currently it is not possible to set the re-send interval per message type.

5.4 A Bare-Bones Example pMediator Configuration

Listing 1 below shows a bare-bones configuration.

Listing 5.1: Example bare-bones configuration of pMediator.

```
1  ProcessConfig = pMediator
2  {
3    resend_thresh = 3
4    max_tries     = 6
5    no_ack_vars   = MUSTER_STATUS
6    group         = blue_team
7    mates         = deb,fin
8  }
```

6 Configuration Parameters of pMediator

The following parameters are defined for pMediator. For some parameters, more detailed description are provided in other sections. Parameters having default values are indicated so.

Listing 6.2: Configuration Parameters for pMediator.

vname:	The name of ownship. This will be used in creating the `ack_id` field in outgoing mediated messages. 4.1
no_ack_vars:	A comma-separted list of outgoing message variables that will not request an acknowledgement return message. A fire-and-forget policy.
no_ack_var:	Same as no_ack_vars, but typically only containing one variable.
group:	Name of the ownship group.
resend_thresh:	Time between re-send attempts unless an ack message has been received. Section 5.3.
max_tries:	Number of times a variable will be re-sent before an ack has been received until giving up. Section 5.2.
mates:	An explicit list of vehicle names to be considered part of our group. Section **??**.
mate:	Same as mates, but typically containing only vehicle name when using this parameter.

7 Publications and Subscriptions for uFldObstacleSim

The interface for pMeditor, in terms of publications and subscriptions, is described below. This same information may also be obtained from the terminal with:

```
$ pMediator --interface or -i
```

7.1 Variables Published by pMediator

- APPCAST: Contains an appcast report identical to the terminal output. Appcasts are posted only after an appcast request is received from an appcast viewing utility. Section 8
- ACK_MESSAGE_LOCAL: For incoming mediated messages requesting an acknowledgement return message, this variable is posted to contain the acknowledgment.
- NODE_MESSAGE: For incoming mediated messages, the contained node message will be republished locally in this variable.
- MEDIATED_MESSAGE_LOCAL: For outgoing node messages. The original node message will be packed in this message. Section 4.1.
- BCM_REPORT_REQUEST: Posted upon startup, this messages requests from the contact manager a list of vehicles with 10K range.

Example postings:

```
MEDIATED_MESSAGE_LOCAL = ack_id=abe_007,ack=true,src_node=abe,dest_node=ben,
                         var_name=RETURN,string_val=true

    ACK_MESSAGE_LOCAL  = id=abe_007,src=abe,dest=ben

          NODE_MESSAGE = src_node=abe,dest_node=ben, var_name=RETURN,
                         string_val=true

    BCM_REPORT_REQUEST = var=MEDIATED_GROUP_10K, range=10000
```

7.2 Variables Subscribed for by pMediator

The uFldObstcleSim application will subscribe for the following MOOS variables:

- APPCAST_REQ: A request to generate and post a new apppcast report, with reporting criteria, and expiration.
- NODE_MESSAGE_LOCAL: When pMediator handles outgoing node messages, it looks for publications to this variable.
- MEDIATED_MESSAGE: When pMediator handles incoming messages, these messages arrive in this variable.
- ACK_MESSAGE: Incoming acknowledgements of previously sent messages.
- MEDIATE_GROUP_10K: A report from the contact manager listing the names of all vehicles in our group with ten kilometers. This variable can be configured or disabled. Section **??**.

Example postings:

```
NODE_MESSAGE_LOCAL = src_node=abe,dest_node=ben,var_name=RETURN,
                     string_val=true
  MEDIATED_MESSAGE = ack_id=abe_007,ack=true,src_node=abe,dest_node=ben,
                     var_name=RETURN,string_val=true
       ACK_MESSAGE = id=abe_007,src=abe,dest=ben
MEDIATED_GROUP_10K = abe,ben,deb
```

8 Terminal and AppCast Output

The pMediator application produces some useful information to the terminal and identical content through appcasting. An example is shown in Listing 3 below.

On line 2, the name of the local community, typically the shoreside community, is listed on the left. On the right, "0/0(1582) indicates there are no configuration or run warnings, and the current iteration of pMediator is 1582. Lines 4-9 show the configuration settings. On Line 7, the list of named mates is empty, which means the list of mates on line 8, were inferred from an incoming message from the contact manager, identifying the teammates ben, cal, and deb.

Lines 11-13 provide an overall recap on total messages sent to all other vehicle for all message types (MOOS variables). Note that the total number of messages sent exceeds the number of acknowledgement messages received in return, indicating there is significant lossy comms at play.

Lines 15-17 summarize the incoming message count from all vehicles, for all message types (MOOS variables). A bad message is one that is improperly formed or missing required fields.

Lines 19-26 provide outgoing message statistics *per vehicle*. Note the total messages sent in the second column is 114, the same total number given oin line 12. Note that comms from abe to deb seems to result in far fewer messages requiring a re-send. Column five shows the average time for a message to be sent and received *on the first try*. This value may be useful in setting the configuration parameter resend_thresh, discussed in Section 5.3.

The output in lines 24-26 in column six is the most direct confirmation that no messages have been lost, even though a re-send was required in several instances.

Listing 8.3: Example UFldObstacleSim *console output.*

```
 1  ================================================================
 2  pMediator abe                                          0/0(1582)
 3  ================================================================
 4  Config:
 5    ReSend thresh: 3
 6        Max Tries: 6
 7      Named mates:
 8            mates: ben,cal,deb
 9      No_ack_vars: CONVOY_STAT_RECAP_ALLY
10
11  Stats (outgoing):
12    Out Msg Count: 114
13    Ack Msg Count: 66
14
15  Stats (incoming):
16    Rcv Msg Count: 107
17    Bad Msg Count: 0
18
19  Outgoing stats: (0)
20  ---------------------------
21        Msgs  Msgs     Avg Msgs  Avg First  Msgs
22  Mate  Sent  Re-Sent  Re-Sent   Interval   Dropped
23  ----  ----  -------  --------  ---------  -------
24  ben   39    11       0.282     1.821      0
25  cal   36    12       0.333     1.799      0
```

```
26  deb   39    1        0.026     1.672      0
27
28  Outgoing stats: (Per Variable)
29  -------------------------------
30  Snds  ReSnds  Freq  Var                      Dest(s)
31  ----  ------  ----  -----------------------  -----------
32  48    0*      6.95  CONVOY_STAT_RECAP_ALLY   ben,cal,deb
33  6     1       1.37  MUSTER                   ben,cal,deb
34  36    11      5.91  MUSTER_COLLAB            ben,cal,deb
35  18    9       2.49  TASK_BID                 ben,cal,deb
36  6     3       1.37  UP_MUSTER                ben,cal,deb
37
38  Incoming stats:
39  ---------------------------
40        Acks  Acks     Avg Acks
41  Mate  Sent  Re-sent  Re-sent
42  ----  ----  -------  --------
43  ben   27    1        0.037
44  cal   31    4        0.129
45  deb   35    9        0.257
46
47  ================================================================
48  Most Recent Events (8):
49  ================================================================
50  [424.82]: Completed msg id: [abe_107], elapsed=1.705
51  [418.54]: Completed msg id: [abe_105], elapsed=4.562
52  [416.01]: Completed msg id: [abe_106], elapsed=2.03
53  [405.94]: Completed msg id: [abe_104], elapsed=1.752
54  [405.94]: Completed msg id: [abe_103], elapsed=1.752
55  [405.36]: Completed msg id: [abe_102], elapsed=1.174
56  [395.36]: Completed msg id: [abe_96], elapsed=8.266
57  [389.05]: Completed msg id: [abe_97], elapsed=1.957
```

In Lines 28-36, a summary of statistics is shown sorted on the message type (MOOS variable). In the first column the total attempted sends is given, followed by the number of re-sends that were needed. (If `max_tries=5` then column two can be at most five times greater than column one.) The frequency value shown in column three is *sends per minute*. The frequency calculation does *not* include re-sends. In other words, the frequency value will be unchanged under perfect or lossy comms since it only counts the initial attempted send for each unique inter-vehicle message. Not the asterisk in column 2 of line 32. This indicates that the zero re-sends value is due to this variable being on the `no_ack_vars` list.

Lines 38-45 summarize the acks sent to other vehicles upon incoming messages. Column one is the name of the vehicle to which the ack is sent. Column two is the number of acks that needed to be sent (one per incoming unique message), and column three is the number of additional acks that needed to be sent. Recall that re-sending an ack is triggered by the original message arriving as a re-send.

9 Implementation Notes

o The NodeMessage class is in `lib_mbutil`. The serialization functions for converting to a string and unpacking from a string are defined on this class. There is no MediatedMessage class. The NodeMessage class also serves this purpose and had two additional fields, the ack id, and the ack=bool fields. They are simply unused and not serialized if they have not been set.