pNodeReporter: Summarizing a Node's Position and Status

June 2018

Michael Benjamin, mikerb@mit.edu
Department of Mechanical Engineering
MIT, Cambridge MA 02139

1 Overview

The `pNodeReporter` MOOS application runs on each vehicle (real or simulated) and generates node-reports (as a proxy for AIS reports) for sharing between vehicles, depicted in Figure 1. The process serves one primary function - it repeatedly gathers local platform information and navigation data and creates an AIS like report in the form of the MOOS variable `NODE_REPORT_LOCAL`. The `NODE_REPORT` messages are communicated between the vehicles and the shore or shipside command and control through an inter-MOOSDB communications process such as pShare or via acoustic modem. Since a node or platform may both generate and receive reports, the locally generated reports are labeled with the `_LOCAL` suffix and bridged to the outside communities without the suffix. This is to ensure that processes running locally may easily distinguish between locally generated and externally generated reports.

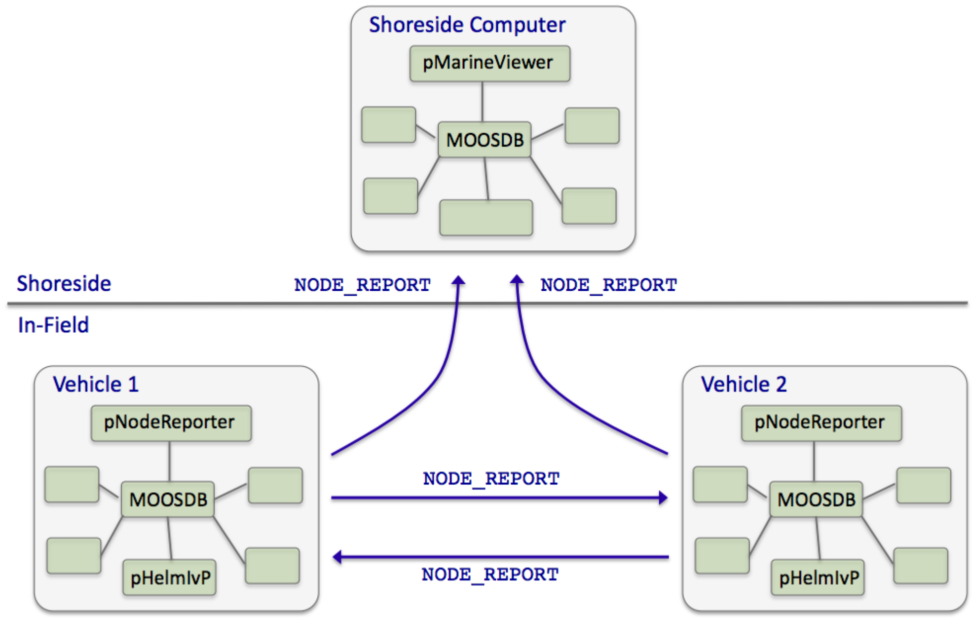

Figure 1: **Typical pNodeReporter usage:** The pNodeReporter application is typically used with pShare or acoustic modems to share node summaries between vehicles and to a shoreside command-and-control GUI.

To generate the local report, pNodeReporter registers for the local NAV_* vehicle navigation data and creates a report in the form of a single string posted to the variable NODE_REPORT_LOCAL. An example of this variable is given in below in Section 2.1. The pMarineViewer and pHelmIvP applications are two modules that consume and parse the incoming NODE_REPORT messages.

The pNodeReporter utility may also publish a second report, the PLATFORM_REPORT. While the NODE_REPORT summary consists of an immutable set of data fields described later in this section, the PLATFORM_REPORT consists of data fields configured by the user and may therefore vary widely across applications. The user may also configure the frequency in which components of the PLATFORM_REPORT are posted within the report.

2 Using pNodeReporter

2.1 Overview Node Report Components

The primary output of pNodeReporter is the node report string. It is a comma-separated list of key-value pairs. The order of the pairs is not significant. The following is an example report:

```
NODE_REPORT_LOCAL = "NAME=alpha,TYPE=UUV,TIME=1252348077.59,X=51.71,Y=-35.50,
                    LAT=43.824981,LON=-70.329755,SPD=2.00,HDG=118.85,YAW=118.84754,
                    DEP=4.63,LENGTH=3.8,MODE=MODE@ACTIVE:LOITERING"
```

The TIME reflects the Coordinated Universal Time as indicated by the system clock running on the machine where the MOOSDB is running. Speed is given in meters per second, heading is in degrees in the range $[0, 360)$, depth is in meters, and the local x-y coordinates are also in meters. The source of information for these fields is the NAV_* navigation MOOS variables such as NAV_SPEED. The report also contains several components describing characteristics of the physical platform, and the state of the IvP Helm, described next.

If desired, pNodeReporter may be configured to use a different variable than NODE_REPORT_LOCAL for its node reports, by setting the configuration parameter node_report_output to MY_REPORT for example. Most applications that subscribe to node reports, subscribe to two variables, NODE_REPORT_LOCAL and NODE_REPORT. This is because node reports are meant to be bridged to other MOOS communities (typically with pShare but not necessarily). A node report should be broadcast only from the community that generated the report. In practice, to ensure that node reports that arrive in one community are not then sent out to other communities, the node reports generated locally have the _LOCAL suffix, and when they are sent to other communities they are sent to arrive with the new variable name, minus the suffix.

2.2 Helm Characteristics

The node report contains one field regarding the current mode of the helm, MODE. Typically the pNodeReporter and pHelmIvP applications are running on the same platform, connected to the same MOOSDB. When the helm is running, but disengaged, i.e., in manual override mode, the MODE field in the node report simply reads "MODE=DISENGAGED". When or if the helm is detected to be not running, the field reads "MODE=NOHELM-SECS", where SECS is the number of seconds since the last time pNodeReporter detected the presence of the helm, or "MODE=NOHELM-EVER" if no helm presence has ever been detected since pNodeReporter has been launched.

How does pNodeReporter know about the health or status of the helm? It subscribes to two MOOS variables published by the helm, IVPHELM_STATE and IVPHELM_SUMMARY. These are described more fully in helm documentation, but below are typical example values:

```
IVPHELM_STATE   = "DRIVE"

IVPHELM_SUMMARY = "iter=72,ofnum=1,warnings=0,time=127349406.22,solve_time=0.00,
                   create_time=0.00,loop_time=0.00,var=course:209.0,var=speed:1.2,
                   halted=false,running_bhvs=none,modes=MODE@ACTIVE:LOITERING,
                   active_bhvs=loiter$17.8$100.00$9$0.04$0/0,completed_bhvs=none
                   idle_bhvs=waypt_return$17.8$0/0:station-keep$17.8$n/a
```

The IVPHELM_STATE variable is published on each iteration of the pHelmIvP process regardless of whether the helm is in manual override ("PARK") mode or not, and regardless of whether the value of this variable has changed between iterations. It is considered the ”heartbeat” of the helm. This is the variable monitored by pNodeReporter to determine whether a "NOHELM" message is warranted. By default, a period of five seconds is used as a threshold for triggering a "NOHELM" warning. This value may be changed by setting the nohelm_threshold configuration parameter.

When the helm is indeed engaged, i.e., not in manual override mode, the value of IVPHELM_STATE posting simply reads "DRIVE", but the helm further publishes the IVPHELM_SUMMARY variable similar to the above example. If the user has chosen to configure the helm using hierarchical mode declarations (as described in helm documentation), the IVPHELM_SUMMARY posting will include a component such as "modes=MODE@ACTIVE:LOITERING" as above. This value is then included in the node report by pNodeReporter. If the helm is not configured with hierarchical mode declarations, the node report simply reports "MODE=DRIVE".

2.3 Platform Characteristics

The node report contains three fields regarding the platform characteristics, NAME, TYPE, and LENGTH. The name of the platform is equivalent to the name of the MOOS community within which pNodeReporter is running. The MOOS community is declared as a global MOOS parameter (outside any given process' configuration block) in the .moos mission file. The TYPE and LENGTH parameters are set in the pNodeReporter configuration block. They may alternatively derive their values from a MOOS variable posted elsewhere by another process. The user may configure pNodeReporter to use this external source by naming the MOOS variables with the platform_length_src and platform_type_src parameters. If both the source and explicit values are set, as for example:

```
platform_length     = 12              // meters
platform_length_src = SYSTEM_LENGTH   // A MOOS Variable
```

then the explicit length of 12 would be used only if the MOOS variable SYSTEM_LENGTH remained unwritten to by any other MOOS application connected to the MOOSDB. The platform length and type may be used by other platforms as a parameter affecting collision avoidance algorithms and protocol. They are also used in the pMarineViewer application to allow the proper platform icon to be displayed at the proper scale.

If the platform type is known, but no information about the platform length is known, certain rough default values may be used if the platform type matches one of the following: "kayak" maps to 4 meters, "mokai" maps to 4 meters, "uuv" maps to 4 meters, "auv" maps to 4 meters, "ship" maps to 18 meters, "glider" maps to 3 meters.

A color may also be associated with a vehicle, which can be useful in applications like pNodeReporter or alogview to indicate a team, or a certain vehicle configuration. This configuration parameter is completely optional. To specify the color:

```
platform_color     = dodgerblue
```

2.4 Dealing with Local versus Global Coordinates

A primary component of the node report is the current position of the vehicle. The pNodeReporter application subscribes for the following MOOS variables to garner this information: NAV_X, NAV_Y in local coordinates, and the pair NAV_LAT, NAV_LONG in global coordinates. These two pairs should be consistent, but what if they are not? And what if pNodeReporter is receiving mail for one pair but not the other? Three distinct policy choices are supported:

- The default policy: node reports include exactly what is given. If NAV_X and NAV_Y are being received only, then there will be no entry in the node report for global coordinates, and vice versa. If both pairs are being received, then both pairs are reported. No attempt is made to check or ensure that they are consistent. This is the default policy, equivalent to the configuration crossfill_policy=literal.

- If one of the two pairs is not being received, pNodeReporter will fill in the missing pair from the other. This policy can be chosen with the configuration cross_fill_policy=fill-empty.
- If NAV_LAT and NAV_LONG are being received, pNodeReporter will use these in its report, *and* convert the global coordinates to local coordinates, and use these converted coordinates in its report, essentially disregarding NAV_X and NAV_Y if they are also being received. This policy can be chosen with the configuration cross_fill_policy=global. This is available in the next release after Release 19.8.x.
- If one of the two pairs has been received more recently, the older pair is updated by converting from the other pair. The older pair may also be in a state where it has never been received. This policy can be chosen with the configuration cross_fill_policy=fill-latest.

2.5 Processing Alternate Navigation Solutions

Under normal circumstances, node reports are generated reflecting the current navigation solution as defined by the incoming NAV_* variables. The pNodeReporter application can handle the case where the vehicle also publishes an alternate navigation solution, as defined by a sister set of incoming MOOS variables separate from the NAV_* variables. In this case pNodeReporter will monitor both sets of variables and may generate *two* node reports on each iteration. The following two configuration parameters are needed to activate this capability:

```
alt_nav_prefix = <prefix>      // example: NAV_GT_
alt_nav_name   = <node-name>   // example: _GT
alt_nav_group  = <group-name>  // example: ground_truth (Post R.19.8.x)
```

The configuration parameter, alt_nav_prefix, names a prefix for the alternate incoming navigation variables. For example, alt_nav_prefix=NAV_GT_ would result in pNodeReporter subscribing for NAV_GT_X, NAV_GT_Y and so on. A separate vehicle state would be maintained internally based on this alternate set of navigation information and a second node report would be generated. The alt_nav_group allows the alt nav node report to have a different group name than the baseline node report. If not specified, the group names will match. This feature was introduced after Release 19.8.x.

A second node report would be published under the same MOOS variable, NODE_REPORT_LOCAL, but the NAME component of the report would be distinct based on the value provided in the alt_nav_name parameter. If a name is provided that does not begin with an underscore character, that name is used. If the name does begin with an underscore, the name used in the report is the otherwise configured name of the vehicle plus the suffix.

3 The Optional Blackout Interval Option

Under normal circumstances, the pNodeReporter application will post a node report once per iteration, the gap between postings being determined solely by the app_tick parameter (Figure 2). However, there are times when it is desirable to add an artificial delay between postings. Node reports are typically only useful as information sent to another node, or to a shoreside computer rendering fielded vehicles, and there are often dropped node report messages due to the uncertain nature of communications in the field, whether it be acoustic communications, WiFi, or satellite link.

Applications receiving node reports usually implement provisions that take dropped messages into account. A collision-avoidance or formation-following behavior, or a contact manager, may extrapolate a contact position from its last received position and trajectory. A shoreside command-and-control GUI such as pMarineViewer may render an interpolation of vehicle positions between node reports. To test the robustness of applications needing to deal with dropped messages, a way of simulating the dropped messages is desired. One way is to add this to the simulation version of whatever communications medium is being used. For example, there is an acoustic communications simulator where the dropping of messages may be simulated, where the probability of a drop may even be tied to the range between vehicles. Another way is to simply simulate the dropped message at the source, by adding delay to the posting of reports by pNodeReporter.

By setting the blackout_interval parameter, pNodeReporter may be configured to ensure that a node report is not posted until *at least* the duration specified by this parameter has elapsed, as shown in Figure 3.

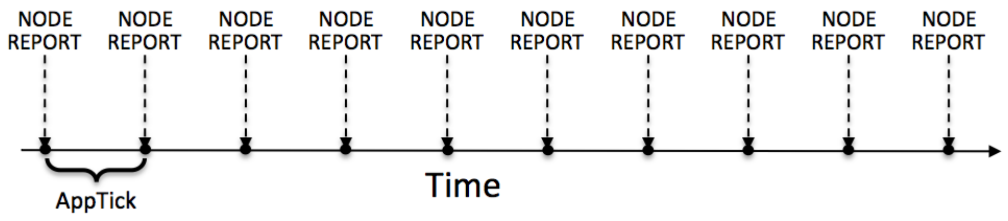

Figure 2: **Normal schedule of node report postings**: The pNodeReporter application will post node reports once per application iteration. The duration of time between postings is directly tied to the frequency at which pNodeReporter is configured to run, as set by the standard MOOS AppTick parameter.

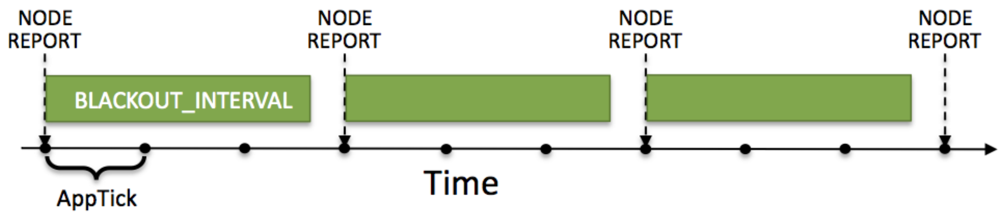

Figure 3: **The optional blackout interval parameter:** The schedule of node report postings may be altered by the setting the BLACKOUT_INTERVAL parameter. Reports will not be posted until at least the time specified by the blackout interval has elapsed since the previous posting.

An element of unpredictability may be added by specifying a value for the blackout_variance parameter. This parameter is given in seconds and defines an interval $[-t, t]$ from which a value is chosen with uniform probability, to be added to the duration of the blackout interval. This variation is re-calculated after each interval determination. The idea is depicted in Figure 4.

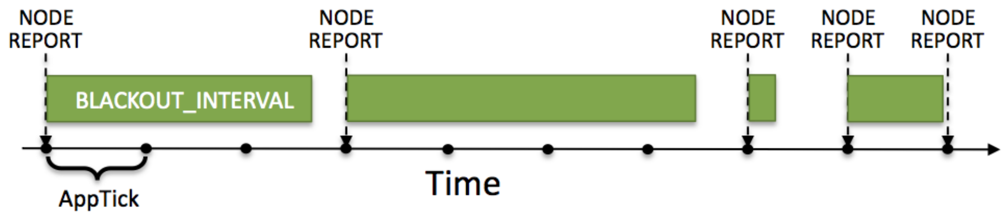

Figure 4: **Blackout intervals with varying duration:** The duration of a blackout interval may be configured to vary randomly within a user-specified range, specified in the `blackout_variance` parameter.

Message dropping is typically tied semi-predictably to characteristics of the environment, such as range between nodes, water temperature or platform depth, an so on. This method of simulating dropped messages captures none of that. It is however simple and allows for easily proceeding with the testing of applications that need to deal with the dropped messages.

4 Configuration Parameters for pNodeReporter

The following parameters are defined for `pNodeReporter`. A more detailed description is provided in other parts of this section. Parameters having default values are indicated so in parentheses.

Listing 4.1: Configuration Parameters for pNodeReporter.

`alt_nav_prefix`:	Source for processing alternate nav reports. Section 2.5.
`alt_nav_group`:	Group associated with node reports generated for alternate nav node reports. Introduced after Release 19.8.x. Section 2.5.
`alt_nav_name`:	Node name in posting alternate nav reports. Section 2.5.
`blackout_interval`:	Minimum duration, in seconds, between reports (0). Section 3.
`blackout_variance`:	Variance in uniformly random blackout duration. Legal values: any non-negative value. The default is zero. Section 3.
`crossfill_policy`:	Policy for handling local versus global nav reports (`"literal"`). Section 2.4.
`node_report_output`:	MOOS variable used for the node report (`NODE_REPORT_LOCAL`). Section 2.1.
`nohelm_threshold`:	Seconds after which a quiet helm is reported as AWOL. Legal values: any non-negative value. The default is 5 seconds. Section 2.2.
`platform_length`:	The reported length of the platform in meters. Legal values: any non-negative value. The default is zero. Section 2.3.
`plat_report_output`:	The Platform report MOOS variable. Legal values: conventions for MOOS variable names. The default is `PLATFORM_REPORT_LOCAL`. Section 6.
`plat_report_input`:	A component of the optional platform report. Section 6.
`platform_color`:	A hint on how to render the vehicle in a GUI application like `pMarineViewer` or `alogview`. Introduced in Release 16.5. Section 2.3.
`platform_type`:	The reported type of the platform. Legal values: any string. The default is `"unknown"`. Section 2.3.
`paused`:	If true, posting of reports is suspended. The default is `"false"`. With release 15.3.

An Example MOOS Configuration Block

An example MOOS configuration block is provided in Listing 2 below. To see an example MOOS configuration block from the console, enter the following:

```
$ pNodeReporter --example or -e
```

This will show the output shown in Listing 2 below.

Listing 4.2: Example configuration of pNodeReporter.

```
===============================================================
pNodeReporter Example MOOS Configuration
===============================================================
Blue lines: Default configuration

ProcessConfig = pNodeReporter
{
  AppTick   = 4
  CommsTick = 4

  // Configure key aspects of the node
  platform_type      = glider   // or {uuv,auv,ship,kayak}
  platform_length    = 8        // meters.  Range [0,inf)

  // Configure optional blackout functionality
  blackout_interval  = 0        // seconds. Range [0,inf)

   // Configure the optional platform report summary
  plat_report_input  = COMPASS_HEADING, gap=1
  plat_report_input  = GPS_SAT, gap=5
  plat_report_input  = WIFI_QUALITY, gap=1
  plat_report_output = PLATFORM_REPORT_LOCAL

  // Configure the MOOS variable containg the node report
  node_report_output = NODE_REPORT_LOCAL

  // Threshold for conveying an absense of the helm
  nohelm_threshold   = 5        // seconds

  // Policy for filling in missing lat/lon from x/y or v.versa
  // Valid policies: [literal], fill-empty, use-latest, global
  crossfill_policy   = literal

  // Configure monitor/reporting of dual nav solution
  alt_nav_prefix     = NAV_GT
  alt_nav_name       = _GT
}
```

5 Publications and Subscriptions for pNodeReporter

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

```
$ pNodeReporter --interface or -i
```

5.1 Variables Published by pNodeReporter

The primary output of pNodeReporter to the MOOSDB is the node report and the optional platform report:

- 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.
- NODE_REPORT_LOCAL: Primary summary of the node's navigation and helm status. Section 2.1.
- PLATFORM_REPORT_LOCAL: Optional summary of certain platform characteristics. Section 2.3.

5.2 Variables Subscribed for by pNodeReporter

Variables subscribed for by pNodeReporter are summarized below. A more detailed description of each variable follows. In addition to these variables, any MOOS variable that the user requests to be included in the optional PLATFORM_REPORT will also be automatically subscribed for.

- APPCAST_REQ: A request to generate and post a new apppcast report, with reporting criteria, and expiration.
- IVPHELM_STATE: A indicator of the helm state produced by pHelmIvP, e.g., "PARK", "DRIVE" "DISABLED", or "STANDBY".
- IVPHELM_ALLSTOP: A indicator of the helm allstop produced by pHelmIvP, e.g., "clear", or a reason why the vehicle is at zero speed.
- IVPHELM_SUMMARY: A summary report produced by the IvP Helm (pHelmIvP).
- NAV_X: The ownship vehicle position on the x axis of local coordinates.
- NAV_Y: The ownship vehicle position on the y axis of local coordinates.
- NAV_LAT: The ownship vehicle position on the y axis of global coordinates.
- NAV_LONG: The ownship vehicle position on the x axis of global coordinates.
- NAV_HEADING: The ownship vehicle heading in degrees.
- NAV_YAW: The ownship vehicle yaw in radians.
- NAV_SPEED: The ownship vehicle speed in meters per second.
- NAV_DEPTH: The ownship vehicle depth in meters.
- PNR_PAUSE: Allows the paused state to be set or toggled. Acceptable values are "true" "false" "toggle". With release 15.3.

If pNodeReporter is configured to handle a second navigation solution as described in Section 2.5, the corresponding additional variables described in that section will also be automatically subscribed for.

5.3 Command Line Usage of pNodeReporter

The pNodeReporter application is typically launched as a part of a batch of processes by pAntler, but may also be launched from the command line by the user. The basic command line usage for the pNodeReporter application is the following:

Listing 5.3: Command line usage for the pNodeReporter *application.*

```
Usage: pNodeReporter file.moos [OPTIONS]

Options:
  --alias=<ProcessName>
      Launch pNodeReporter with the given process name
      rather than pNodeReporter.
  --example, -e
      Display example MOOS configuration block.
  --help, -h
      Display this help message.
  --version,-v
      Display the release version of pNodeReporter.
```

6 The Optional Platform Report Feature

The pNodeReporter application allows for the optional reporting of another user-specified list of information. This report is made by posting the PLATFORM_REPORT_LOCAL variable. An alternative variable name may be used by setting the PLAT_REPORT_SUMMARY configuration parameter. This report may be configured by specifying one or more components in the pNodeReporter configuration block, of the following form:

```
plat_report_input = <variable>, gap=<duration>, alias=<variable>
```

If no component is specified, then no platform report will be posted. The `<variable>` element specifies the name of a MOOS variable. This variable will be automatically subscribed for by pNodeReporter and included in (not necessarily all) postings of the platform report. If the variable BODY_TEMP is specified, a component of the report may contain "BODY_TEMP=98.6". An alias for a MOOS variable may be specified. For example, `alias=T`, for the BODY_TEMP component would result in "T=98.6" in the platform report instead.

How often is the platform report posted? Certainly it will not be posted any more often than the apptick parameter allows, but it may be posted far more infrequently depending on the user configuration and how often the values of its components are changing. The platform report is posted only when one or more of its components requires a re-posting. A component requires a re-posting only if (a) its value has changed, *and* (b) the time specified by its gap setting has elapsed since the last platform report that included that component. When a PLATFORM_REPORT_LOCAL posting is made, only components that required a posting will be included in the report.

The wide variation in configurations of the platform report allow for reporting information about the node that may be very specific to the platform, not suitable for a general-purpose node report. As an example, consider a situation where a shoreside application is running to monitor the platform's battery level and whether or not the payload compartment has suffered a breach, i.e., the presence of water is detected inside. A platform report could be configured as follows:

```
plat_report_input = ACME_BATT_LEVEL, gap=300, alias=BATTERY_LEVEL
plat_report_input = PAYLOAD_BREACH
```

This would result in an initial posting of:

```
PLATFORM_REPORT_LOCAL = "platform=alpha,utc_time=1273510720.99,BATTERY_LEVEL=97.3,
                         PAYLOAD_BREACH=false"
```

In this case, the platform uses batteries made by the ACME Battery Company and the interface to the battery monitor happens to publish its value in the variable ACME_BATT_LEVEL, and the software on the shoreside that monitors all vehicles in the field accepts the generic variable BATTERY_LEVEL, so the alias is used. It is also known that the ACME battery monitor output tends to fluctuate a percentage point or two on each posting, so the platform report is configured to include a battery level component no more than once every five minutes, (gap=300). The MOOS process monitoring the indication of a payload breach is known to have few false alarms and to publish its findings in the variable PAYLOAD_BREACH. Unlike the battery level which has frequent minor fluctuations and degrades slowly, the detection of a payload breach amounts to the flipping of a Boolean value and needs to be conveyed to the shoreside as quickly as possible. Setting gap=0, the default, ensures that a platform report is posted on the very next iteration of pNodeReporter, presumably to be read by a MOOS process controlling the platform's outgoing communication mechanism.

7 An Example Platform Report Configuration Block for pNodeReporter

Listing 4 below shows an example configuration block for pNodeReporter where an extensive platform report is configured to report information about the autonomous kayak platform to support a "kayak dashboard" display running on a shoreside computer. Most of the components in the platform report are specific to the autonomous kayak platform, which is precisely why this information is included in the platform report, and not the node report.

Listing 7.4: An example pNodeReporter configuration block.

```
//------------------------------------------------------------------
// pNodeReporter config block

ProcessConfig = pNodeReporter
{
  AppTick = 2
  CommsTick = 2

  platform_type       = KAYAK
  platform_length     = 3.5     // Units in meters
  nohelm_thresh       = 5       // The default
  blackout_interval   = 0       // The default
  blackout_variance   = 0       // The default

  node_report_output  = NODE_REPORT_LOCAL        // The default
  plat_report_output  = PLATFORM_REPORT_LOCAL    // The default

  plat_report_input = COMPASS_PITCH, gap=1
  plat_report_input = COMPASS_HEADING, gap=1
  plat_report_input = COMPASS_ROLL, gap=1
  plat_report_input = DB_UPTIME, gap=1
  plat_report_input = COMPASS_TEMPERATURE, gap=1, alias=COMPASS_TEMP
  plat_report_input = GPS_MAGNETIC_DECLINATION, gap=10, alias=MAG_DECL
  plat_report_input = GPS_SAT, gap=5
  plat_report_input = DESIRED_RUDDER, gap=0.5
  plat_report_input = DESIRED_HEADING, gap=0.5
  plat_report_input = DESIRED_THRUST, gap=0.5
  plat_report_input = GPS_SPEED, gap=0.5
  plat_report_input = DESIRED_SPEED, gap=0.5
  plat_report_input = WIFI_QUALITY, gap=0.5
  plat_report_input = WIFI_QUALITY, gap=1.0
  plat_report_input = MOOS_MANUAL_OVERRIDE, gap=1.0
}
```

8 Measuring the Odometry Extent Per Mission Hash

A new feature, *after* Release 22.8, is introduced to enable pNodeReporter to post useful information to be logged with the vehicle alog file. This coincides with the introduction of a *mission hash*. The mission hash is published by the shoreside and shared to all vehicles and logged. It has the form:

```
MISSION_HASH = mhash=221119-1255K-NEAR-MILK,utc=1668880538.69
```

The posting contains both the hash itself, 221119-1255K-NEAR-MILK, and the UTC timestamp when the hash was created on the shoreside. The first part of the hash reflects the time and date, 221119-1255, Nov 19th, 2022, 12:55. The second part of the hash is comprised of randomly configured words that are easier to remember. The final two-word part of the hash can be used as an alias in some applications.

The shoreside generates the hash, and it cannot be ruled out that the shoreside generates multiple hashes for a single vehicle alog file. This can happen for example if the shoreside app, e.g., pMarineViewer, generating the mission hash, is re-started during the vehicle deployment. In this case the vehicle alog file may have multiple mission hashes, leading to confusion. This is where pNodeReporter can help.

pNodeReporter will register for MISSION_HASH and use an odometer to track the maximum vehicle extent associated with this hash. The *extent* is the maximum distance from the odometer starting point, i.e., the farthest linear distance from the starting point at any time in the mission. The node reporter will publish to PNR_MHASH whenever this maximum extent has reached a new max value. If the MISSION_HASH value should change mid-mission, the PNR_MHASH value will also change.

If the mission hash changes mid-mission, we may see a set of alog entries from pMarineViewer along the lines of:

```
14.853   PNR_MHASH  mhash=221119-1412M-GOLD-EPIC,ext=2.6
15.874   PNR_MHASH  mhash=221119-1412M-GOLD-EPIC,ext=3.9
16.897   PNR_MHASH  mhash=221119-1412M-GOLD-EPIC,ext=5.4
22.538   PNR_MHASH  mhash=221119-1412M-GOLD-EPIC,ext=6.9
23.560   PNR_MHASH  mhash=221119-1412M-GOLD-EPIC,ext=8.3
24.587   PNR_MHASH  mhash=221119-1412M-GOLD-EPIC,ext=9.5
26.119   PNR_MHASH  mhash=221119-1412M-GOLD-EPIC,ext=10.6
177.068  PNR_MHASH  mhash=221119-1413E-MAIN-KHAN,ext=12.1
178.093  PNR_MHASH  mhash=221119-1413E-MAIN-KHAN,ext=13.5
179.117  PNR_MHASH  mhash=221119-1413E-MAIN-KHAN,ext=14.9
180.129  PNR_MHASH  mhash=221119-1413E-MAIN-KHAN,ext=16.3
181.159  PNR_MHASH  mhash=221119-1413E-MAIN-KHAN,ext=17.8
182.172  PNR_MHASH  mhash=221119-1413E-MAIN-KHAN,ext=19.2
183.207  PNR_MHASH  mhash=221119-1413E-MAIN-KHAN,ext=20.4
184.228  PNR_MHASH  mhash=221119-1413E-MAIN-KHAN,ext=21.8
218.643  PNR_MHASH  mhash=221119-1413E-MAIN-KHAN,ext=23.2
219.682  PNR_MHASH  mhash=221119-1413E-MAIN-KHAN,ext=24.3
220.693  PNR_MHASH  mhash=221119-1413E-MAIN-KHAN,ext=25.4
222.219  PNR_MHASH  mhash=221119-1413E-MAIN-KHAN,ext=26.9
223.762  PNR_MHASH  mhash=221119-1413E-MAIN-KHAN,ext=28.3
225.317  PNR_MHASH  mhash=221119-1413E-MAIN-KHAN,ext=29.8
226.876  PNR_MHASH  mhash=221119-1413E-MAIN-KHAN,ext=31.1
```

In this case the mission hash was changed sometime between the offset times of 26.119 and 177.068. When `pNodeReporter` noticed that the max extent distance accumulated while the mission hash was `"MAIN-KHAN"` had exceeded 10.6, a new `PNR_MHASH` posting was made.

In a nutshell, if there was unfortunately more than one mission hash received by a vehicle and logged in a single alog file, this information can help determine which mission hash was the ”real” one. It should always be the hash associated with the final posting to `PNR_MHASH`. And this information can be obtained using a few `aloggrep` switches:

```
  $ aloggrep file.alog --final --no_report
  226.876  PNR_MHASH    pNodeReporter   mhash=221119-1413E-MAIN-KHAN,ext=31.1
```

Or, to isolate the hash itself:

```
  $ aloggrep file.alog --final --format=val --subpat=mhash
  221119-1413E-MAIN-KHAN
```

The latter example is essentially the whole point of this feature: to enable `aloggrep` to determine the most likely single mission hash for a give alog file, even if multiple hashes exist. This tool can then be used as a component of other scripts that are performing post-mission log file archiving tasks.