Robotarium Backend Software Architecture

Robotarium Backend: Paul Goltfelter
Documentation: Ian Buckley

Init. 19 April 2018, Compiled July 1, 2018

1 Introduction

The Robotarium backend uses MQTT, a lightweight publish/subscribe messaging protocol, to execute experiments. The purpose of this document is to detail the communication aspects of the Robotarium backend in sufficient detail that the Robotarium can be re-implemented and adapted to serve as an underwater testbed.[1] This document is structured as follows. First, a brief description of MQTT, the underlying messaging protocol used by the Robotarium, is presented. Then, the messaging framework Vizier, which leverages MQTT and facilitates information transfer, is documented. Lastly, the topology of the MQTT clients/nodes of the Robotarium backend is presented.

2 A Brief Description of MQTT

In this section, the salient features of MQTT, with respect to communication, are briefly introduced. For a full description of the MQTT specification, see [1]. From the specification, MQTT is a light-weight Client Server publish/subscribe messaging transport protocol. The protocol runs over TCP/IP, or other network protocols that provide *ordered, lossless,* and *bidirectional* connections.

2.1 Control Packet Format

MQTT control packets are formatted as in Figure 1.

Fixed Header The fixed header consists of two parts. The first byte specifies the MQTT control packet type, e.g. publish or subscribe, and any flags on those types. From the second byte on, the remaining length in bytes of the MQTT control packet is listed, excluding the bytes used to encode the remaining length. The remaining length can take up to 4 bytes.

[1]The Robotarium is under constant revision, so the details of the Robotarium backend, and thus, the accuracy of the information contained in this document, are subject to change.

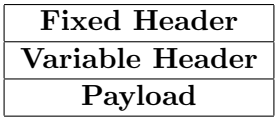

Figure 1: MQTT Control Packets include a fixed header, a variable header, and the payload.

Variable Header Depending on the packet type indicated in the fixed header, certain packet types have a variable header, which further specifcy the behaviour of the packet. The variable header is not used for certain packet types.

Payload After the fixed and variable headers, the remaining bytes in the packet are the payload.

2.2 Data Representation

MQTT relies on two data types: 16-bit integers and UTF-8 encoded strings. Integers data values are 16 bits in big-endian order, i.e. the most significant byte is presented first, followed by the least significant byte. UTF-8 strings are composed of a 2-byte prefix followed by the sequence of 8-bit characters; the two byte prefix limits the string size to 2^{16} bits, equivalently 8192 characters.

3 Vizier Messaging Framework

Vizier is a custom messaging protocol for designed for the Robotarium to facilitate communication between the different MQTT clients [2]. Vizier is implemented as a Python 3 module; Vizier provides convenient wrappers around the Paho-MQTT module, whose `mqtt` class and methods are used explicitly. Vizier requires a running MQTT broker, but is independent of the implementation of the broker.

3.1 Messages

Messages between Vizier nodes can be any byte-encoded data, which is a requirement inherited through the Paho-MQTT module. In particular, strings are a convenient data type for messages. In practice, messages are passed as JSON formatted strings. JSON is used for several reasons, not least of which are the Python tools for parsing JSON formatted string and the syntax, which is similar to Python's built-in dictionary syntax.

3.2 Topic Types

Vizier uses two topic types: `STREAM` and `DATA`. The messages of `STREAM` topics are published immediately. The Vizier `node` class offers two self-explained methods for handling `STREAM` topics: `node.publish()` and `node.subscribe()`. In contrast, the messages of `DATA` topics are published on-demand. For handling `DATA` topics, two class methods are provided: `node.post()`, which posts the message on the topic, and `node.get_data()`, which requests data from the topic. Intuitively, the `STREAM` topic type should be used for messages that are always relevant, such as control inputs or pose information, and the `DATA` topic type should be used for messages that are sometimes useful, such as battery levels.

3.3 Nodes and the Vizier Server

In addition to simplifying messaging between the MQTT clients, a primary function of Vizier is to ensure that all topic subscriptions are satisfied, which is crucial for operation of the Robotarium. To do this, a special Vizier node, the Vizier Server, is launched first. The Vizier Server has a special `DATA` topic, `vizier/setup`, which subscribes to the dedicated `topic_name/node_descriptor` (`DATA`)topics of the Vizier nodes. On the `/node_descriptor` topics, each Vizier node publishes its node descriptor file, a JSON formatted file that lists the topics (and their types) to which the node

publishes and subscribes. The Vizier Server uses the information from each of the nodes' descriptor files to verify that all subscriptions are filled.

3.4 A Worked Example

To ground the previous descriptions of the key components of the Vizier messaging framework, example code, included with the Vizier module, is provided to verify installation of the module and to understand the procedure for its use. In this section, installation and basic usage is detailed. This worked example will require a Linux operating; in particular, this example was tested using Ubuntu 18.04.

3.4.1 Installation

Vizier has requires Python 3 and the python package manager, `pip`. On Ubuntu 18.04, Python 3 is installed by default, and pip can be installed using `apt`:

```
sudo apt-get update
sudo apt-get install python3-pip
```

Vizier requires installation of two dependencies: `paho-mqtt` and `testresources`. Using `pip`, these are easily install by running:

```
sudo python3 -m pip install testresources
sudo python3 -m pip install paho-mqtt
```

Installation of the Vizier module is simple. First, clone the git repository:

```
git clone https://github.com/user2552/vizier.git
```

Then, run execute the `setup.py` script found in the newly created `Vizier/python` directory:

```
python3 ./vizier/python/setup.py
```

Vizier requires an MQTT broker to be running. Vizier is agnostic to which broker is used, so any broker can be used. For use on Ubuntu 18.04, the MQTT broker `mosquitto` is available using the `apt` package repository. To install `mosquitto`, run the following in the terminal:

```
sudo apt-get update
sudo apt-get install mosquitto
```

3.4.2 Scripted Start Up

First, change directories to the `worked_example` directory. Vizier requires a running MQTT broker. Launching `mosquitto` on a specific port is performed via

```
mosquitto -p 1884
```

Next, the Vizier Server is launched by running the shell script:

```
./test_vizier.sh
```

Lastly, the Vizier nodes are launched by running the following shell script:

```
./test_nodes.sh
```

In the terminal, the following should be printed

```
The
quick
brown
fox
jumps
over
the
lazy
dog.
```

3.4.3 Inside the Code

Two python scripts are used: `test_vizier.py` for the Vizier Server and `test_node.py` for the Vizier nodes. The `test_vizier.py` script is an extensible example for starting the Vizier Server: it parses the command line options to identify the host, MQTT broker port, and the Vizier Server configuration file; and it performs the checks to ensure that all of the node descriptor files are available and fill the subscriptions. The `test_node.py` is executed for both nodes, `a` and `b`. The nodes receive index values from each other, print the string at that index in a list, and publish the next increment of the index. The nodes stop when the the entire list has been printed.

References

[1] A. Banks and R. Gupta, "MQTT version 3.1.1," October 2014.

[2] P. Glotfelter. (2016) Vizier: An mqtt networking framework. [Online]. Available: https://github.com/robotarium/vizier