CCOM Data Shuttle

Val Schmidt

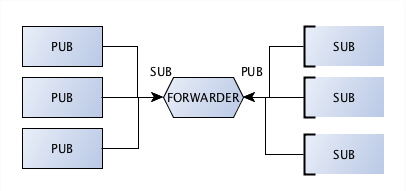
Center for Coastal and Ocean Mapping

University of New Hampshire

The Data Shuttle (name to be changed) is a package of software for pushing real-time or near-real-time data streams over a possibly tenuous IP link. The purpose of DS is to provide a mechanism to push data from various field sensors and platforms back to our campus for near-real-time processing and display.

DS is built on the python implementation of zmq which is a message passing library. This document is written to provide some guidance regarding use of the software.

The basic architecture is that of the “Forwarder” reproduced below from the documentation:



<http://learning-0mq-with-pyzmq.readthedocs.org/en/latest/pyzmq/devices/forwarder.html#forwarder>

Field sensors and systems will publish data to a “forwarder” living in the Center’s network DMZ. The forwarder will then provide this data to any number of subscribers internal to the CCOM network (or possibly elsewhere). The zmq library is robust to losses of network connectivity between publishers and the forwarder or between the forwarder and the subscribers, reconnecting automatically when the network is restored. [Note, publishers and subscribers do not seem to be able to recover after a computer goes to sleep and subsequently wakes.] The zmq library will buffer messages (up to 1000 by default) before dropping them, fifo style. This is true for each publisher, when the forwarder is not available, and for the forwarder itself when a previously connected subscriber is not available. When a subscriber is disconnected for an extended period it’s subscription is removed.

The forwarder (for the moment) operates on marvel.ccom.unh.edu, accessible from within CCOM as graywhale.ccom.unh.edu

The *dataproxy* module

Dataproxy is a module aimed at making it easy to write publishers and subscribers and to implement the forwarder. It contains three classes (publisher, forwarder and subscriber).

The forwarder is easiest and only requires one to define input and output network ports to which publishers and subscribes connect respectively. The script forwarder.py implements this basic setup with default ports of 7777 input and 7778 output by default.

The publisher class has methods for getting data (get\_data() )from a sensor and sending that data to the forwarder, with a few options regarding pre-processing the data before sending (more on that in a moment). The get\_data() method is an empty method by default and requires being over-ridden for each publisher implementation. For example, if one desired to send GPS data form a GPS connected via serial port, one would write your own publisher.get\_data() method to read data from the serial port and return it. Similarly the process\_data() is an empy method by default and requires over-ridden for each subscriber implementation. For example, if one wanted to write the incoming GPS data sent by the publisher and received by the subscriber to a file, one would write subscriber.process\_data() to take the incoming data and write it to a file.

The one-the-wire format of the message may be in one of 4 forms currently supported by the dataproxy module, ASCII string, compressed binary, compressed python pickle or numpy array. The module takes care of all the details of reading and writing these formats. One need only choose one appropriate to the task.

Topics:

The zmq library supports the idea of “topics” which allow a subscriber to subscribe to only those topics it desires. This is implemented by matching a topic name, which is passed along with each message as an ASCII string. When passing ASCII string messages, the topic may be included in the message, delimited by white space form the message itself. However, this method cannot work for binary messages and so multipart messages called “envelopes” are used.

A zmq envelop is a multipart message in which the first part is the ASCII topic and the second part is the message itself. (There may be additional parts if the data must be sent in multiple messages. See the zmq guide for additional details). The details of this is all handled behind the scenes by the dataproxy module.

When a subscriber connects to the forwarder it does so having defined a “topic filter”, that is used to subscribe to only specific messages having a matching topic. A topic filter will match if it matches the beginning of the topic. For example “TEST” will match “TEST1” and “TEST 123”. So while there is no explicit wild-card or regular expression matching, with careful naming of topics one can subscribe to sets of data. This fact guides the following naming convention used in this messaging setup.

Topic Naming:

The topic naming convention will be as follows:

PLATFORM\_SENSOR\_NN\_ENC

PLATFORM specifies the platform on which the data is coming from. It may be one of the following:

RVCS

RVGS

CML

SENSOR is a description of the type of sensor sending the data. It may be one of:

GPS

CAMERA

MET

INS

NN is a 2-digit, zero-padded index, allowing for the sending of multiple sensors of the same time.

01

02

03

ENC specifies the encoding of the data. The zmq library can encode the data as ASCII text, compressed python pickle or numpy array with any of the following:

STR

PKL

NPY

For example, a publisher sending GPS ASCII data from the R/V Gulf Surveyor using the compressed binary pickle format would specify its topics as

RVGS\_GPS\_01\_PKL