**Offline data storage mechanism for Liota**

If the client faces network disconnectivity, publish message can be stored as a persistent storage or in a temporary offline queue in which publish data will be added to an internal queue until the number of queued-up requests reaches the size limit of the queue. If the size of the queue is defined as negative integer it will act as an infinite queue. One can also choose the queue behavior after it reaches its specified size. If drop\_oldest behavior is set to be true, oldest publish message is dropped else the newest publish messages are dropped. One should specify the draining frequency in each case, which implies how data which has been stored will be published once the network connectivity is established.

You can also specify data\_drain\_size which specifies ow much data will be drained at once after the internet connectivity is established again. By default, both are set to 1.

**Example:**

Suppose we want to create a persistent storage, while creating instance of DCC, we would pass an instance of Buffering class along with it.

By default, persistent\_storage parameter is set as False:

buffering\_params = BufferingParams (persistent\_storage=True, data\_drain\_size=10, draining\_frequency=1)

graphite = Graphite(SocketDccComms(ip=config['GraphiteIP'], port=8080), buffering\_params=buffering\_params)

Here, data\_drain\_size is 1 and draining\_frequency is 1 which specifies 10 messages will be sent per second.

For persistent storage, a sqlite database will be created by the name of storage.db which will store all the messages while network connectivity is broken.

Once network connectivity is back messages will be removed from database as they get published.

In case of persistent\_storage as False which is default the queueing, mechanism will be used, you can specify queue\_size and other parameters like drop\_oldest, data\_drain\_size and draining\_frequency:

buffering\_params = BufferingParams (queue\_size=-1, data\_drain\_size=10, draining\_frequency=1)

will create a queueing mechanism with infinite size and drop\_behaviour by default is set to true, data\_drain\_size and draining\_frequency can be any positive integer.

For queue with size 3 and drop\_oldest behaviour set to true,

buffering\_params = BufferingParams (queue\_size=3, drop\_oldest=True, draining\_frequency=1)

As the publish message arrives the queue will be like this after 3 publish message arrive:

['msg1', 'msg2', 'msg3']

As the fourth publish message arrives:

['msg2', 'msg3', 'msg4']

For the fifth publish message:

['msg3', 'msg4', 'msg5']

Similarly, if the drop\_oldest behaviour is set to False:

['msg1', 'msg2', 'msg3']

After this any new coming publish message will be dropped.

**Testing of buffering mechanism:**

Initial condition:

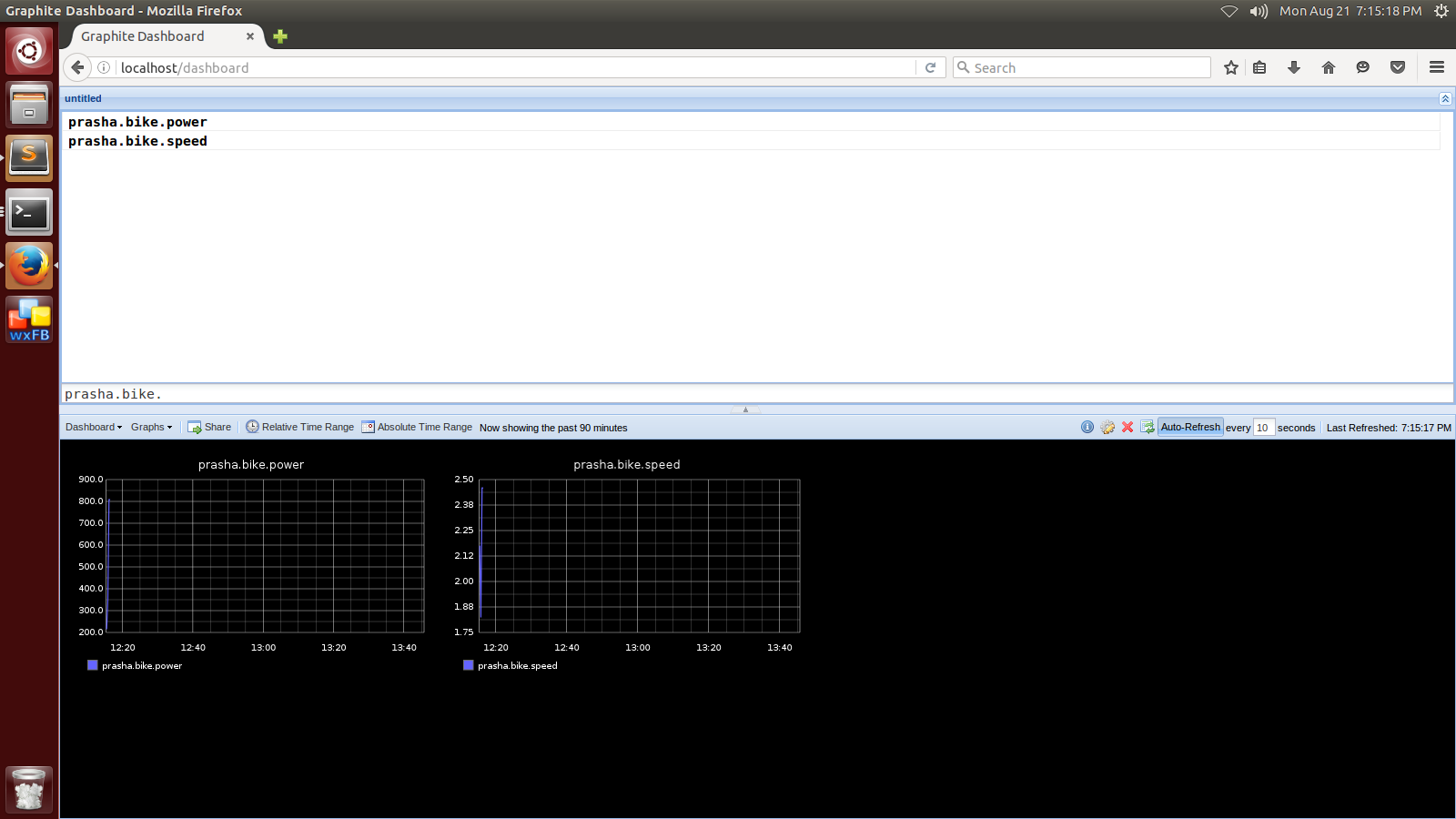
Ubuntu 14.04 Virtual machine running on Windows 10.

The interval of data publish set to 2 for both bike.power and bike.speed metric. queue\_size is infinite persistent\_storage =False, data\_drain\_size is set to 10 and draining\_frequency is 1.

Network is disconnected starts at 17:45:17:

CPU Utilization: 8.93

Memory free: 89.96

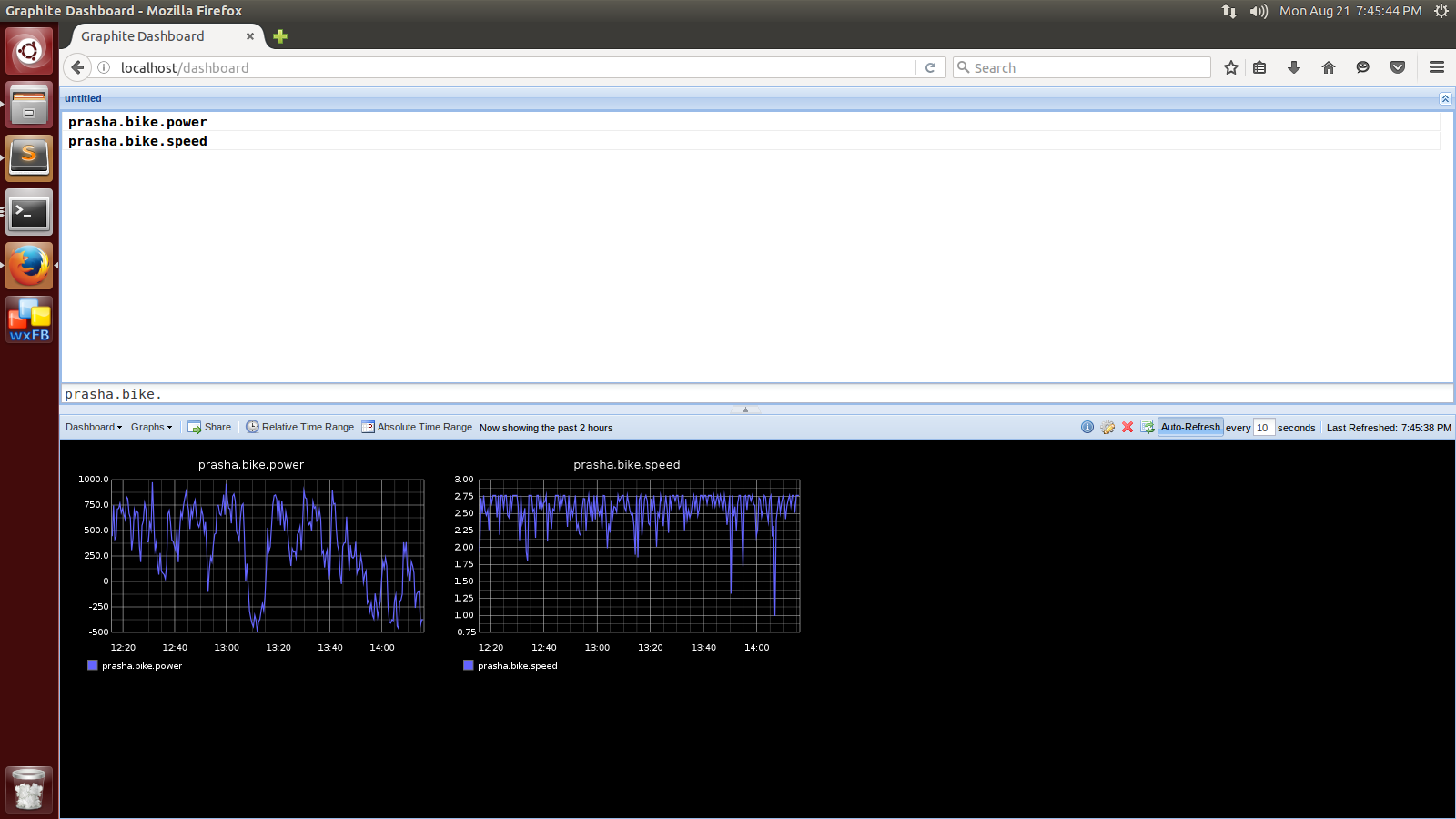


Network connectivity resumed at 19:15:17:

Draining starts, initially queue length is 5348.

Draining ends at: 19:33:36 with 10 messages being published per second which are stored in the queue.

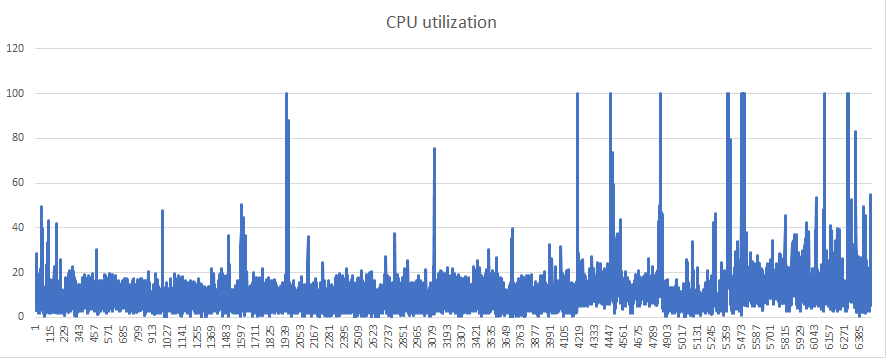
Graphite browser at end of draining finish:



CPU utilization during the whole test time(1 hour 50 min):

From 1-5370: No network connectivity was there

From 5370-6464: Draining was occurring



CPU Memory Free:

From 1-5370: No network connectivity was there

From 5370-6464: Draining was occurring