UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

NavUP
Longsword Testing Report on Broadsword Data

Compiled By

Lucian Sargeant - u15225560
Ritesh Doolabh - u15075754
Peter Boxall - u14056136
Claude Greeff - u13153740
Harris Leshaba - u15312144
Hristian Vitrychenko - u15006442

GitHub Repository: COS 301 Team Longsword Data GitHub Repository(Phase 4)
Slack Group: COS 301 Team Longsword Data Slack Group(Phase 4)
Scrum-board: COS 301 Team Longsword Data Scrum-board(Phase 4)
Burn-down Chart: COS 301 Team Longsword Data Burn-down Chart(Phase 4)

2017
TEAM LONGSWORD (DATA)

Contents

1 Introduction

For this phase we will be testing the Data module of the BroadSword Team. We have split the testing phase according to Functional Requirements, Non-Functional Requirements, Use Cases and Testing Cases.

Their code was primarily coded in Python and used a NSQ message processing system. We will be testing the various cases and giving a brief description of how we tested followed by an explanation of the mark that was given to them.

2 Testing Methods

The testing of the Data module becomes more complex due to the fact that multiple independent components are used to achieve the purpose of data streaming. This then means that we had to integrate code within the files given us so that we could effectively and accurately test the system given to us. With the regard to the aforementioned, it can be clearly seen that the use of any form of a unit testing framework would be difficult to implement and thus we opted for the integrated code test. This form of testing also shows which part/s of the code given to us performed its actions to the necessary pedigree and which part/s did not.

3 Service Contracts

3.1 Retrieving and passing device MAC address.

MARK: 10

The access module sends a location request to the data module via the NSQ server,the access module has to subscribes to a topic.The main entry point of the system, query resolver.py, sends the mac address to the Aruba server in a query to get the location of the device. In query resolver.py, the handler function receives a request, it then passes the mac address to the searcher function-which will create a query to get the location for the device.

This functional requirement was fulfilled by the Broadsword team.

```python
73   def handler(message):
74     # validator=malformed_test.validateRequest()
75     # validator.validate(message)
76     obj = json.loads(message.body)
77     if (obj['src'] == 'data' and obj['msgType'] == 'request'):
78       #print obj['content']['mac']
79       location = Searcher(obj['content']['mac'])
80       src = obj['src']
81       dest = obj['dest']
82       msgtype = "response"
83       content = location
84       m=publish(src, dest, msgtype, content)
85       tornado.ioloop.PeriodicCallback(pub_message(m,dest), 1000).start()
86       return True
87   def pub_message(message,destination):
88     writer.pub(destination,str(message), finish_pub)
89   def finish_pub(conn, data):
90     print(data)
```

Figure 1: Handler function.

3.2 Logging in and maintaining a session with Aruba ALE.

MARK: 4

The Broadsword data team does indeed log in to the Aruba ALE and they do this by making use of the python urllib2 library. Their query_resolver.py entry point takes arguments for the various ALE login fields, the arguments are passed between classes until reaching the Aruba class which is instantiated with the details. This Aruba class contains a method called get() which effectively creates a URL, containing the address for accessing Aruba appended by the current request. The login is then handled with a password manager and finally the function returns the JSON formatted response from Aruba. The function can be seen in the figure below.

```python
class Aruba:

    def __init__(self, host_name, port, username, password):
        self.host_name = host_name
        self.port = port
        self.username=username
        self.password=password

    "TODO: add exception handling"
    def get(self,query_string):
        url=self.host_name+"/"+query_string
        p = urllib2.HTTPPasswordMgrWithDefaultRealm()
        p.add_password(None, url, self.username, self.password)
        handler = urllib2.HTTPBasicAuthHandler(p)
        opener = urllib2.build_opener(handler)
        urllib2.install_opener(opener)
        return urllib2.urlopen(url).read()
```

Figure 2: Implementation of the Aruba Wrapper class.

There is however a grave issue with this approach as each time the get() function is called, which occurs for every request, the login to Aruba needs to happen again as a session is not maintained. This results in a massive time overhead especially considering that the program is intended to be scaled to handle tens of thousands of requests a second. This is simply not possible with Broadswords code as it stands, as each request-response pair to Aruba is taking roughly a second to complete. This can been seen in the diagram below where only 73 requests were processed in a minute.

```
ocation","content":{ "mac_address": "EC:1F:72:B7:5D:39" ,"x": 41.874733, "y": 20
.70608, "building_name": "HF4228-IT Building", "floor_name": "Floor 4"} }
        70          58.4424219131
{"src":"data","dest":"navigation","msgType":"response",,"queryType":"getCurrentL
ocation","content":{ "mac_address": "EC:1F:72:B7:5D:39" ,"x": 41.874733, "y": 20
.70608, "building_name": "HF4228-IT Building", "floor_name": "Floor 4"} }
        70          58.4424219131
{"src":"data","dest":"navigation","msgType":"response",,"queryType":"getCurrentL
ocation","content":{ "mac_address": "EC:1F:72:B7:5D:39" ,"x": 41.874733, "y": 20
.70608, "building_name": "HF4228-IT Building", "floor_name": "Floor 4"} }
        71          59.0637218952
{"src":"data","dest":"navigation","msgType":"response",,"queryType":"getCurrentL
ocation","content":{ "mac_address": "EC:1F:72:B7:5D:39" ,"x": 41.874733, "y": 20
.70608, "building_name": "HF4228-IT Building", "floor_name": "Floor 4"} }
        71          59.0637218952
{"src":"data","dest":"navigation","msgType":"response",,"queryType":"getCurrentL
ocation","content":{ "mac_address": "EC:1F:72:B7:5D:39" ,"x": 41.874733, "y": 20
.70608, "building_name": "HF4228-IT Building", "floor_name": "Floor 4"} }
        72          59.6855230331
{"src":"data","dest":"navigation","msgType":"response",,"queryType":"getCurrentL
ocation","content":{ "mac_address": "EC:1F:72:B7:5D:39" ,"x": 41.874733, "y": 20
.70608, "building_name": "HF4228-IT Building", "floor_name": "Floor 4"} }
        72          59.6855230331
73 requests processed in 60.1202399731 seconds
```

Figure 3: The number of Aruba requests processed in one minute.

3.3 Processing the request and retrieving location.

MARK: 10

In the python file query_resolver.py, Broadsword provides the options of using mock data to test their program or to follow standard procedure and use real data from Aruba by connecting to it through location_lookup.py, building_lookup.py and floor_lookup.py. The previously mentioned python files each connect to Aruba and all separately log in by utilising Aruba_wrapper.py which establishes and maintains a session with Aruba. Each class also filters their own JSON objects that are returned by Aruba and return only the necessary data thus fulfilling the requirement of processing requests and retrieving location.

This functional requirement was not fully fulfilled by the Broadsword team.

```python
import json
import aruba_wrapper

class LocationLookup:
        def __init__(self, hostname, port, username, password):
                self.aruba_handle = aruba_wrapper.Aruba(hostname,port,username,password)

        def get_json(self,mac_addr):
                raw_json = self.aruba_handle.get('/api/v1/location?sta_eth_mac='+mac_addr)
                return json.loads(raw_json)


        def lookup(self,mac_addrress):
                obj = self.get_json(mac_addrress)
                for field in obj['Location_result']:
                        if 'msg' in field:
                                return ("{\"x\": "+str(field['msg']['sta_location_x'])+", \"y\": "+str(field['msg']['sta_location_y'])+


class LocationLookupTest:
        def __init__(self):
                self.loc = LocationLookup("127.0.0.1","80","","")

        def get_mock_json(self,mac_addr):
                raw_json = open('mock_location_json', 'r').read()
                return json.loads(raw_json)

        def test_lookup(self):
                self.loc.get_json = self.get_mock_json
                if(self.loc.lookup("58:48:22:a7:84:6b")=="{\"x\": "+str(55.566734)+", \"y\": "+str(42.82108)+", \"building_id\": \"08BE
                        print "Location lookup test passed"
                else:
                        print "Location lookup test failed"
```

Figure 4: Aruba_wrapper.py

```
import json
import aruba_wrapper

class LocationLookup:
        def __init__(self, hostname, port, username, password):
                self.aruba_handle = aruba_wrapper.Aruba(hostname,port,username,password)

        def get_json(self,mac_addr):
                raw_json = self.aruba_handle.get('/api/v1/location?sta_eth_mac='+mac_addr)
                return json.loads(raw_json)


        def lookup(self,mac_addrress):
                obj = self.get_json(mac_addrress)
                for field in obj['Location_result']:
                        if 'msg' in field:
                                return ("{\"x\": "+str(field['msg']['sta_location_x'])+", \"y\": "+str(field['msg']['sta_location_y'])+", \

class LocationLookupTest:
        def __init__(self):
                self.loc = LocationLookup("127.0.0.1","80","","")

        def get_mock_json(self,mac_addr):
                raw_json = open('mock_location_json', 'r').read()
                return json.loads(raw_json)

        def test_lookup(self):
                self.loc.get_json = self.get_mock_json
                if(self.loc.lookup("58:48:22:a7:84:6b")=="{\"x\": "+str(55.566734)+", \"y\": "+str(42.82108)+", \"building_id\": \"08BEB4DE
                        print "Location lookup test passed"
                else:
                        print "Location lookup test failed"
```

Figure 5: Mock JSON

3.4 Returning a location to the source of the request.

MARK: 8

After the Aruba ALE connection is made by the Aruba-wrapper class, the results of the location query are then delt with in the LocationLookup class. This class instantiates and executes the Aruba-wrapper implementation and then processes the JSON results to filter out what is needed.

This functional requirement was fulfilled by the Broadsword team.

Figure 6: JSON returned

4 Non-Functional Requirements

4.1 Level of concurrency of the task.

MARK: 0

Broadsword made use of a server known as NSQ which is widely known for its message passing based procedures. Whilst the NSQ core was built in Go (a programming language with built in concurrent), Broadsword did not seem to make use of any of those concurrent features in their code. This was made apparent in testNavigationConsumer.py where each NSQ action was made on after the other with no concurrency apparent anywhere in terms of mac address processing and message passing. In terms of concurrent programming in general, there is none in any of the code presented by Broadsword.

This non-functional requirement was not fulfilled by the Broadsword team.

```
    f = nsq.Reader(message_handler=h, lookupd_http_addresses=[address_port],
                   topic='navigation', channel='navup', lookupd_poll_interval=15)
    #tornado.ioloop.IOLoop.instance().run_sync(do_pub(m))
    #print("wrote one message to nsq")
    nsq.run()
```

Figure 7: NSQ Use Without Concurrency

```
    writer = nsq.Writer(['127.0.0.1:4150'])
    tornado.ioloop.PeriodicCallback(publish_destination,1000).start()
    nsq.run()

    ##########################################################################
```

Figure 8: NSQ Writer

4.2 Performance of the request processing.

MARK: 2

Broadsword used a NSQ server which is a message passing system that primarily relies on queue structures to store information. Although NSQ does allow for concurrent programmability, these features were not used efficiently enough to allow large numbers of requests to be "streamed" at high speeds. We simply used a timer in their code to count the number of instances of the location object that was returned for a specific MAC address. This terminated the code after 60 seconds and gave the final number of instances returned.

As can be seen from the screen-shots below, within the duration of 60 seconds, the system only managed to return 73 Location objects for a single mac address. In terms of real-world functionality, this would mean a significant bottleneck on the system. This is because the system would need at a minimum roughly 30 000 requests to be fulfilled in a matter of 60 seconds. In the regard of high speed data streaming, the Broadsword Data module has failed.

This non-functional requirement was not fulfilled by the Broadsword team.

Figure 9: Number of requests processed in 60 Seconds

Figure 10: Code inserted to test the number of requests processed in 60 seconds

4.3   Maintainability and modularity of the code and repository.

MARK: 4

For someone not intimately familiar with the code, it is fairly difficult to understand the flow of the program. There is decent documentation on getting the code operational however the explanation of how the program actually works is lacking and to make matters worse there are also no comments in the code to explain what is happening. This could make maintaining the code an issue as if the developers need to return to it after a long time even they may have issues remembering exactly how the program flows and functions.

The code is very modular having been separated effectively into separate classes based on function, which is good however once again it makes the program difficult to follow without any commenting.

Finally, the GitHub repository is disorganised. It has been divided into a number of branches that do each each have an explained function however, as the repository stands, the branches are unsynchronised and not fulfilling their intended function.

All of the above considered Broadsword receives the mark of 4/10 for this section.

4.4   Integrability and ease of transfer into a final system.

MARK: 5

Whilst the code is very modular in format allowing for certain pieces to be easily matched to other systems, the code appears to be fragmented to a stage where there are a lot of inter-dependency between these classes. Meaning It would be harder to break the system up and mould it to a new system. Also the lack of commenting makes it difficult for one to see where things are implemented and what certain aspects of the code does. This makes it harder for a developer who is required to integrate the code who may not have necessarily written it.

This non-functional requirement was partially fulfilled by the Broadsword team.

```python
parser = argparse.ArgumentParser(description='Serving location requests on the an NSQ topic',formatter_class=arg
parser.add_argument('--nsqlookupd_hostname',
                    help='The http hostname of the NSQ lookupd daemon', default='127.0.0.1', metavar='')
parser.add_argument('--nsqlookupd_port',
                    help='The port number of the NSQ lookupd daemon', default='4161', metavar='')
parser.add_argument('--nsqlookupd_polling_interval',
                    help='The amount of time in seconds between querying all of the supplied nsqlookupd instance
parser.add_argument('--nsqd_hostname',
                    help='The http hostname of the NSQ daemon', default='127.0.0.1', metavar='')
parser.add_argument('--nsqd_port',
                    help='The http port of the NSQ daemon', default='4150', metavar='')
parser.add_argument('--subscribe_topic',
                    help='The nsq topic to listen for requests', default='data', metavar='')
parser.add_argument('--nsq_channel',
                    help='The channel within an NSQ topic to listen for requests', default='navup', metavar='')
parser.add_argument('--aruba_hostname',
                    help='The hostname of the aruba location engine', default='https://137.215.6.208', metavar='
parser.add_argument('--aruba_port',
                    help='The port of the aruba location engine', default='80', metavar='')
parser.add_argument('--aruba_username',
                    help='The username to use the aruba location engine', default='', metavar='')
parser.add_argument('--aruba_password',
                    help='The username to use the aruba location engine', default='', metavar='')
parser.add_argument('--test', action='store_true', help='The username to use the aruba location engine')
args = parser.parse_args()

if(args.test):
    building_lookup.BuildingLookupTest().test_lookup()
    floor_lookup.FloorLookupTest().test_lookup()
    location_lookup.LocationLookupTest().test_lookup()
    exit()

logging.basicConfig(filename='error.log',level=logging.WARNING)
writer = nsq.Writer([args.nsqd_hostname+':'+args.nsqd_port])

def publish(src, dest, msgtype, content):
    result="{\"src\":\""+src+"\",\"dest\":\""+dest+"\",\"msgType\":\""+msgtype+"\",,\"queryType\":\"getCurrentLoca
    return result

def Searcher(mac_string):
    locationL= location_lookup.LocationLookup(args.aruba_hostname,args.aruba_port,args.aruba_username,args.aruba_p
    location_json=json.loads(locationL.lookup(mac_string))
    buildingID=location_json['building_id']
    floorID=location_json['floor_id']
    x=location_json['x']
    y=location_json['y']


    floorL=floor_lookup.FloorLookup(args.aruba_hostname,args.aruba_port,args.aruba_username,args.aruba_password)
    floor_name=floorL.lookup(buildingID,floorID)

    buildingL= building_lookup.BuildingLookup(args.aruba_hostname,args.aruba_port,args.aruba_username,args.aruba_p
    building_name=buildingL.lookup(buildingID)

    final_content="{ \"mac_address\": \""+mac_string+"\" ,\"x\": "+str(x)+", \"y\": "+str(y)+", \"building_name\"
    return final_content

m="";
def handler(message):
    # validator=malformed_test.validateRequest()
    # validator.validate(message)
    obj = json.loads(message.body)
    if (obj['src'] == 'data' and obj['msgType'] == 'request'):
        #print obj['content']['mac']
        location = Searcher(obj['content']['mac'])
        src = obj['src']
        dest = obj['dest']
        msgtype = "response"
    content = location
```

Figure 11: No Comments

5 Use Cases

Broadsword has made use of a NSQ. A 'nsqd' instance is designed to handle multiple streams of data at once. In NSQ terms, streams are called "topics" where a topic can have 1 or more "channels". A channel maps to a downstream service consuming a topic. The method broadsword used was to create a topic, "data", through subscribing to channel "navup", which is a channel on the "data" topic. The topic is created through the first subscription. This method lets channels and topics buffer their data independently. This allows for fast downstream, preventing a slow consumer to cause a bottleneck or delay.

Successful upstream and downstream communication ticks the use case boxes, although more efficient changes could be made for concurrent processing, a mark of 8 out of 10 for downstream and upstream seems fair.

5.1 Upstream communication.

MARK: 8

The file named "publisher.py" will write to nsqd port 4150, this is their upstream communication channel.

```python
writer = nsq.Writer(['127.0.0.1:4150'])
tornado.ioloop.PeriodicCallback(publish_destination,1000).start()
nsq.run()
```

Figure 12: Upstream channel

5.2 Downstream communication.

MARK: 8

Consumers make use of a HTTP /lookup endpoint. Consumers are introduced to topics through making use of the addresses of Broadsword's nsqlookupd instance. In their case it would be host name '127.0.0.1' and port '4161'. This would be their downstream communication.

```python
parser.add_argument('--nsqlookupd_port',
                    help='The port number of the NSQ lookupd daemon', default='4161', metavar='')
```

Figure 13: Downstream channel initialize

```python
r = nsq.Reader(message_handler=handler, lookupd_http_addresses=[args.nsqlookupd_hostname+':'+args.nsqlookupd_port],
topic=args.subscribe_topic, channel=args.nsq_channel, lookupd_poll_interval=args.nsqlookupd_polling_interval)
nsq.run()
```
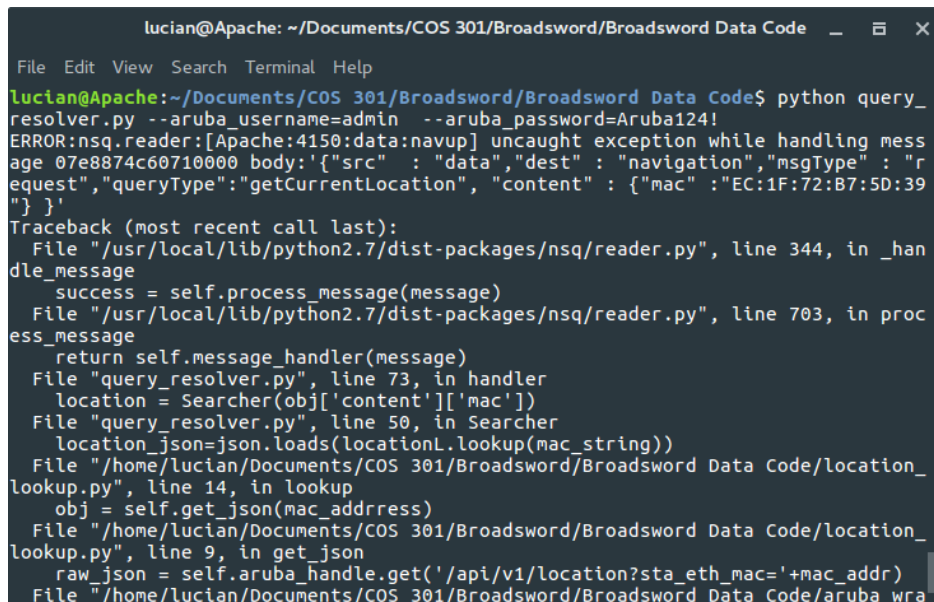
Figure 14: Downstream channel execute

5.3 Security.

MARK: 8

Broadsword requires that the login credentials for Aruba be passed into their program via parameters rather than being specified directly in the code. This creates a layer of security as had the details been hard coded, anyone could access and discover them simply by looking at the code. This is especially true as the project is currently open source and available freely on GitHub.

This was a very mindful decision on Broadswords part and a nice touch, however they do not handle an incorrect password very effectively. Having actively made the decision to pass in the details via function arguments one could expect that they would handle an incorrect password however this is not the case and this is demonstrated in the figure below. An incorrect password results in an uncaught exception in the program. This is not desirable as one cannot decipher the cause simply by examining the error message.

Figure 15: Exception occurring when an incorrect password is used.

The non-functional requirement is there however and as a result Broadsword receives the mark of 8/10 for this section.

6   Evaluation of Broadsword Test Cases

MARK: 6

The Broadsword team makes use of extensive testing cases that caver all the information necessary for the Aruba connection as well as cover the mock information For the GIS module. This good coverage provides utility for successful test cases showing results that would be used in the final NAVUP system.

The MAC address that is handed to the 'aruba_wrapper' class, involved in the Aruba Ale connection and communication, are chosen randomly from a group of 4 available test cases. This provides some sense of variety in the testing of the system.

The issue with the current test cases is that although the coverage may seem extensive, the fulfillment of the non-functional requirement of concurrency is not tested in any form. Concurrency is an important part of the data streaming service for the final NAVUP system. In conclusion for good test-case coverage, variety in MAC address testing as well as extensive mock input data and classes whilst keeping in mind the lack of concurrency and speed and its importance to the project, a mark of 6 is given.

Figure 16: The random selection of MAC addresses

Figure 17: An image showing tests on different parts of the data system

7   Conclusion

In conclusion, as a team we felt that the Broadsword Data team only implemented the core functionality of the module at hand and did not focus the majority of their efforts into making sure that high speed data streaming takes place at all times. This could have been misleading by the fact that requests can be issued to the NSQ Server at a very high rate, but this is merely adding to the queue and these requests are not serviced in ample time so as to serve the task at hand.

After taking into consideration all the tests performed with quantitative and qualitative data, and after critical assessment of their code, we as a team have come to a conclusion of a mark of 73/120 , which translates to a percentage of 60.8%.