

# National University of Singapore

## School of Computing

### CS3103 – Computer Networks Practice

#### **Experiment 5: CoAP with Raspberry Pi [for Phyton Users]**

**Learning Objective:** Implement, understand and analyse CoAP protocol in action including CoAP service discovery, CoAP observer mode. Experiment with Raspberry PI GPIO, Monitoring IoT network using raspberry PI.

**Note 1: Bring your laptop with Python and a remote desktop s/w installed. No lab questionnaire for this lab.**

Remote Desktop Options:

MAC/Linux: Real VNC <https://www.realvnc.com/raspberrypi/>

Windows: MobaXterm (preferred) MobaXterm (<https://mobaxterm.mobatek.net/>) or

Putty Putty (<https://www.putty.org/>)

**Note 2: Demonstrate output of the following to your TA for instant grading. (No lab questionnaire for this Lab).**

- **(6 marks) parts B and C**
- **(4 marks) part D**
- **(0 marks) parts F and H [Skip, if there is no time].**

<b>A</b>	<b>Installing txThings and all dependencies [Skip, if it is already Installed]</b>
	<p>In this tutorial we will be using txThings library for CoAP.</p> <ul style="list-style-type: none"> <li>- <i>txThings</i> - CoAP library for Twisted framework</li> <li>- <i>txThings</i> is a Python implementation of Constrained Application Protocol (CoAP):</li> <li>- <i>txThings</i> is based on Twisted - asynchronous I/O framework and networking engine written in Python. <i>txThings</i> uses MIT License (like Twisted itself).</li> </ul> <ol style="list-style-type: none"> <li>1. Install txThings. <ul style="list-style-type: none"> <li>- ssh to raspberry pi</li> <li>- create handsON2 folder and change directory to handsON2</li> <li>- install txThings: <code>git clone</code>  <code>git://github.com/siskin/txThings.git</code>  (note: check the examples in txThings/examples folder.)</li> </ul> </li> <li>2. Install twisted package. <a href="https://pypi.org/project/Twisted/">https://pypi.org/project/Twisted/</a> <ul style="list-style-type: none"> <li>- <code>pip install twisted</code></li> </ul> </li> </ol> <p style="text-align: center;">Note: If pip is not working , update pip using...</p> <pre style="background-color: #f0f0f0; padding: 10px;">\$wget https://bootstrap.pypa.io/get-pip.py -O ./get-pip.py \$python ./get-pip.py (for python 2) \$python3 ./get-pip.py (for python 3)</pre>

	<ol style="list-style-type: none"> <li>3. Install other Dependencies for txThings <ul style="list-style-type: none"> <li>- Change to txThings folder.</li> <li>- <code>sudo python setup.py install</code></li> </ul> </li> </ol>
<b>B</b>	<b>Creating CoAP Server</b> <ol style="list-style-type: none"> <li>1. Create a file named coapserver.py</li> <li>2. Copy the source code provided below (adapted from sample codes). [You can download coapserver.py from: <a href="https://www.comp.nus.edu.sg/~bhojan/resources/iot/coapserver.py">https://www.comp.nus.edu.sg/~bhojan/resources/iot/coapserver.py</a>]</li> <li>3. Run the server: <code>python coapserver.py</code></li> </ol> <pre> ''' Created on 30-05-2019 CoAP server runs in official IANA assigned CoAP port 5683. @author: Bhojan Anand (adapted from sample codes) '''  import sys import datetime  from twisted.internet import defer from twisted.internet.protocol import DatagramProtocol from twisted.internet import reactor from twisted.python import log  import txthings.resource as resource import txthings.coap as coap  class CounterResource (resource.CoAPResource):    #extends CoAPResource class     """     Example Resource which supports only GET method. Response is a     simple counter value.      Name render_&lt;METHOD&gt; is required by convention. Such method should     return a Deferred. [Python-differ module provides a small framework for     asynchronous programming. The Deferred allows to chain callbacks.     There are two type of callbacks: normal callbacks and errbacks, which     handle an exception in a normal callback.] If the result is available     immediately it's best     to use Twisted method defer.succeed(msg).     """      def __init__(self, start=0):         resource.CoAPResource.__init__(self)         self.counter = start         self.visible = True </pre>

```

        self.addParam(resource.LinkParam("title", "Simple Counter
Resource"))

    def render_GET(self, request):
        response = coap.Message(code=coap.CONTENT, payload='%d' %
(self.counter,))
        self.counter += 1
        return defer.succeed(response)

class BlockResource (resource.CoAPResource):
    """
    Example Resource which supports GET, and PUT methods. It sends large
    responses, which trigger blockwise transfer (>64 bytes for normal
    settings).

    As before name render_<METHOD> is required by convention.
    """

    def __init__(self):
        resource.CoAPResource.__init__(self)
        self.visible = True
        self.addParam(resource.LinkParam("title", "Simple Message String
Resource"))

    def render_GET(self, request):
        payload=" Welcome to NUS school of computing. In this course you
will learn Secured Internet of Things application development and
Analytics."
        response = coap.Message(code=coap.CONTENT, payload=payload)
        return defer.succeed(response)

    def render_PUT(self, request):
        log.msg('PUT payload: %s', request.payload)
        payload = "Just do it!."
        response = coap.Message(code=coap.CHANGED, payload=payload)
        return defer.succeed(response)

class SeparateLargeResource(resource.CoAPResource):
    """
    Example Resource which supports GET method. It uses callLater
    to force the protocol to send empty ACK first and separate response
    later. Sending empty ACK happens automatically after
coap.EMPTY_ACK_DELAY.
    No special instructions are necessary.

    Notice: txThings sends empty ACK automatically if response takes too
    long.

    Method render_GET returns a deferred. This allows the protocol to

```

do other things, while the answer is prepared.

Method `responseReady` uses `d.callback(response)` to "fire" the deferred, and send the response.

"""

```
def __init__(self):
    resource.CoAPResource.__init__(self)
    self.visible = True
    self.addParam(resource.LinkParam("title", "Some Large Text
Resource."))

def render_GET(self, request):
    d = defer.Deferred()
    reactor.callLater(3, self.responseReady, d, request)
    return d

def responseReady(self, d, request):
    log.msg('response ready. sending...')
    payload = "Seek what you know as the highest. It does not matter
whether it is going to happen or not - living with a vision itself is a
very elevating process."
    response = coap.Message(code=coap.CONTENT, payload=payload)
    d.callback(response)
```

```
class TimeResource(resource.CoAPResource):
    def __init__(self):
        resource.CoAPResource.__init__(self)
        self.visible = True
        self.observable = True

        self.notify()
        self.addParam(resource.LinkParam("title", "Time now"))
```

```
def notify(self):
    log.msg('TimeResource: trying to send notifications')
    self.updatedState()
    reactor.callLater(60, self.notify)
```

```
def render_GET(self, request):
    response = coap.Message(code=coap.CONTENT,
payload=datetime.datetime.now().strftime("%Y-%m-%d %H:%M"))
    return defer.succeed(response)
```

```
class CoreResource(resource.CoAPResource):
    """
```

Example Resource that provides list of links hosted by a server.  
Normally it should be hosted at `/.well-known/core`

Resource should be initialized with "root" resource, which can be used  
to generate the list of links.

For the response, an option "Content-Format" is set to value 40, meaning "application/link-format". Without it most clients won't be able to automatically interpret the link format.

Notice that `self.visible` is not set – that means that resource won't be listed in the link format it hosts.  
"""

```
def __init__(self, root):
    resource.CoAPResource.__init__(self)
    self.root = root

def render_GET(self, request):
    data = []
    self.root.generateResourceList(data, "")
    payload = ",".join(data)
    log.msg("%s", payload)
    response = coap.Message(code=coap.CONTENT, payload=payload)
    response.opt.content_format =
coap.media_types_rev['application/link-format']
    return defer.succeed(response)

# Resource tree creation
log.startLogging(sys.stdout)
root = resource.CoAPResource()

well_known = resource.CoAPResource()
root.putChild('.well-known', well_known)
core = CoreResource(root)
well_known.putChild('core', core)

counter = CounterResource(5000)
root.putChild('counter', counter)

time = TimeResource()
root.putChild('time', time)

other = resource.CoAPResource()
root.putChild('other', other)

block = BlockResource()
other.putChild('block', block)

separate = SeparateLargeResource()
other.putChild('separate', separate)

endpoint = resource.Endpoint(root)
reactor.listenUDP(coap.COAP_PORT, coap.Coap(endpoint)) #, interface=":::")
reactor.run()
```

C	<b>Creating CoAP Client</b>
	<ol style="list-style-type: none"> <li>1. Create a file named coapclient.py</li> <li>2. Copy the source code provided below (adapted from sample codes). [You can download coapclient.py from: <a href="https://www.comp.nus.edu.sg/~bhojan/resources/iot/coapclient.py">https://www.comp.nus.edu.sg/~bhojan/resources/iot/coapclient.py</a>]</li> <li>3. Run the server: python coapclient.py</li> </ol> <pre> ''' Created on 30-05-2019 CoAP client connects to CoAP server at default port 5683. @author: Bhojan Anand (adapted from sample codes) '''  import sys from ipaddress import ip_address  from twisted.internet import reactor from twisted.python import log  import txthings.coap as coap import txthings.resource as resource  class Agent:     """     Example class which performs single GET request to coap.me     port 5683 (official IANA assigned CoAP port), URI "test".     Request is sent 1 second after initialization.      Remote IP address is hardcoded – no DNS lookup is performed.      Method requestResource constructs the request message to     remote endpoint. Then it sends the message using protocol.request().     A deferred 'd' is returned from this operation.      Deferred 'd' is fired internally by protocol, when complete response is     received.      Method printResponse is added as a callback to the deferred 'd'. This     method's main purpose is to act upon received response (here it's     simple print).     """      def __init__(self, protocol):         self.protocol = protocol         reactor.callLater(1, self.requestResource)      def requestResource(self):         request = coap.Message(code=coap.GET) </pre>

```

# Send request to "coap://coap.me:5683/test"

request.opt.uri_path = ['counter']
request.opt.observe = 0
request.remote = (ip_address("127.0.0.1"), coap.COAP_PORT)
#Change to appropriate remote IP
d = protocol.request(request,
observeCallback=self.printLaterResponse)
d.addCallback(self.printResponse)
d.addErrback(self.noResponse)

def printResponse(self, response):
    print('First result: ' + response.payload)
    # reactor.stop()

def printLaterResponse(self, response):
    print('Observe result: ' + response.payload)

def noResponse(self, failure):
    print('Failed to fetch resource:')
    print(failure)
    # reactor.stop()

log.startLogging(sys.stdout)

endpoint = resource.Endpoint(None)
protocol = coap.Coap(endpoint)
client = Agent(protocol)

reactor.listenUDP(61616, protocol) # , interface=":::")
reactor.run()

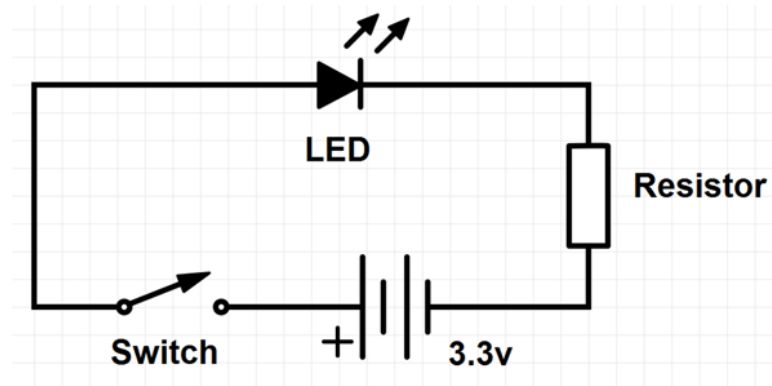
```

**Additional Practice 1:** In your free time, try to run the client program in your laptop instead of Raspberry Pi.

## D Raspberry Pi GPIO: Playing with LEDs

### Introduction

One of the simplest electrical circuits that you can build is a battery connected to a light source and a switch (the resistor is there to protect the LED):



The Raspberry Pi replaces both the switch and the battery in the above diagram.

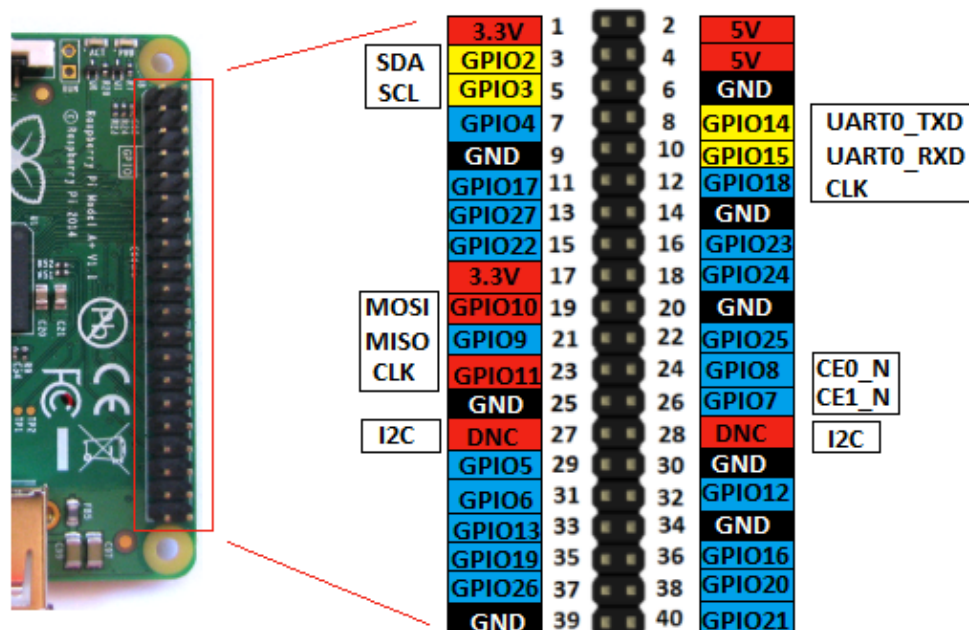
### Requirements

Raspberry Pi board with Raspbian OS

1 x LED light

2 x female to female jumper cable wires

### Raspberry Pi 3 B+ Model – PIN Configuration:

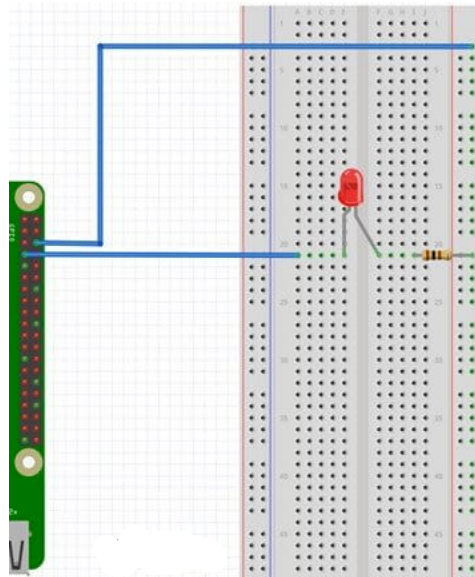


### Raspberry Pi 3 B+ Pinout



There are actually two labels for all of the pins, Broadcom (BCM) and board (BOARD). The board option will let you refer to the pin's actual number on the board, and the Broadcom number is the actual pin number that the Broadcom chip considers it to be. It seems to be that BCM specification is the \*actual\* pin number, so we'll use that.

### Circuit:



### Code:

- 1) Install python GPIO library for raspberry Pi

```
$ sudo apt-get install python-rpi.gpio
```

- 2) Type the following code using nano.

Alternatively you can download the code from:

<https://www.comp.nus.edu.sg/~bhojan/resources/iot/gpioled.py>

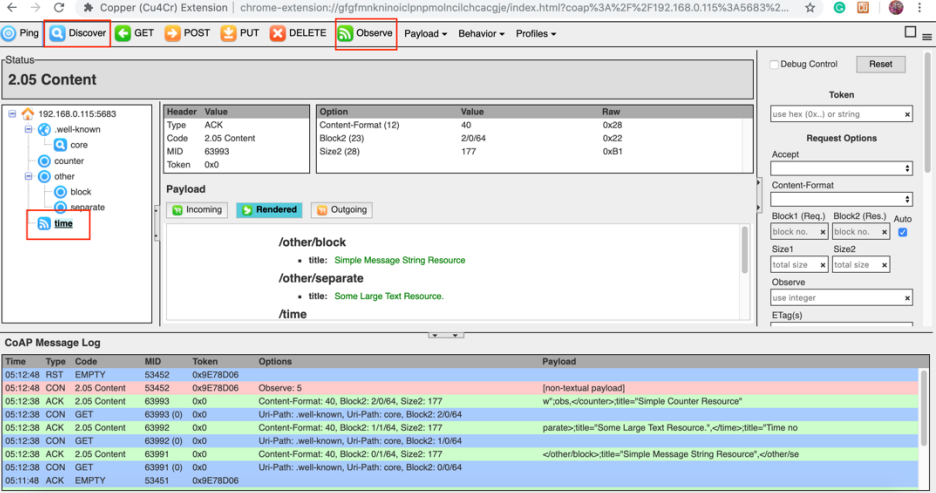
```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)    #we are using Broadcom (BCM) labels to identify
                           #the PINs.

GPIO.setup(4, GPIO.OUT)   #GPIO pin 4 is used, in OUTPUT mode. [In board, it
                           #is pin number 7]

while (1):
    GPIO.output(4, GPIO.HIGH)
    time.sleep(1)
    GPIO.output(4, GPIO.LOW)
```

	<pre>time.sleep(1)  GPIO.cleanup()</pre>
<b>E</b>	<b>CoAP Service Discovery</b> <p><b>Discovery:</b> In terms of communications protocols, discovery is the process whereby clients can “discover” the various properties provided by the server. Discovery is absent from some of the older telemetry protocols (e.g. MODBUS), and it’s a very desirable feature. Without automated discovery, the client will have to be informed of the server’s capabilities by some other means, e.g. manual provisioning or reading in a configuration file for the device etc. Automatic provisioning by discovery is much quicker and reliable.</p> <p>CoAP Support For Discovery</p> <p>A client can discover properties about a server by issuing a CoAP GET to the URL:</p> <pre>.well-known/core</pre> <ol style="list-style-type: none"> <li>1. Amend the CoAP client code discussed previously to know the services available at the server.</li> <li>2. Change the resource list to [ '.well-known', 'core' ]</li> </ol>
<b>F</b>	<b>CoAP Observable Resource</b> <p><b>Observable Resource:</b> You can make a resource observable to get updates regularly or when the state changes.</p> <ol style="list-style-type: none"> <li>1. Create the resource as usual, and set the <i>observable</i> property to True. Then, each time you want to broadcast a new state, just call <i>self.notify()</i> as shown the CoAP server code discussed in previous hands-on exercise.</li> </ol> <pre>class TimeResource(resource.CoAPResource):     def __init__(self):         resource.CoAPResource.__init__(self)         self.visible = True         self.observable = True          self.notify()         self.addParam(resource.LinkParam("title", "Time now"))      def notify(self):         log.msg('TimeResource: trying to send notifications')         self.updatedState()         reactor.callLater(60, self.notify)      def render_GET(self, request):         response = coap.Message(code=coap.CONTENT,                                 payload=datetime.datetime.now().strftime("%Y-%m-%d %H:%M"))         return defer.succeed(response)</pre>

G	<h2>Additional Practice 2: Viewing Observable Resources in Chrome browser (Do it in your free time!)</h2>
	<ol style="list-style-type: none"> <li>1. Add copper extension for chrome browser as stated in   <a href="https://github.com/anuflora/Copper4Cr">https://github.com/anuflora/Copper4Cr</a> </li> <li>2. Browse to the observable 'time' resource (e.g. coap://192.168.1.100:5683/time).          &lt;OR&gt; Browse all available resources (e.g. coap://192.168.1.100:5683/) and press the <i>Discover</i> button, and you should see all the resources available.           Now select 'time' resource (which is an observable resource) and click 'observe' to see the time message coming every minute.       </li> </ol>  <ol style="list-style-type: none"> <li>3. For an alternate example, you can try with simulated temperature, which has an observable temperature resource:   <a href="https://www.comp.nus.edu.sg/~bhojan/resources/iot/temperatureserver.py">https://www.comp.nus.edu.sg/~bhojan/resources/iot/temperatureserver.py</a> </li> </ol>
H	<h2>Observable Resources</h2> <ol style="list-style-type: none"> <li>1. txThings client for receiving observable recourse.</li> <li>2. The code is given below. You can also download from :  <a href="https://www.comp.nus.edu.sg/~bhojan/resources/iot/coapclient_observe.py">https://www.comp.nus.edu.sg/~bhojan/resources/iot/coapclient_observe.py</a> </li> </ol> <pre> ''' Created on 30-05-2019 CoAP client for observable blocks @author: Bhojan Anand (adapted from sample codes) '''  import sys  from twisted.internet.defer import Deferred from twisted.internet.protocol import DatagramProtocol from twisted.internet import reactor from twisted.python import log  import txthings.coap as coap import txthings.resource as resource </pre>

```

from ipaddress import ip_address

class Agent():
    """
    Example class which performs single GET request to 'time' resource that
    supports observation
    """

    def __init__(self, protocol):
        self.protocol = protocol
        reactor.callLater(1, self.requestResource)

    def requestResource(self):
        request = coap.Message(code=coap.GET)
        #Send request to "coap://iot.eclipse.org:5683/obs-large"
        request.opt.uri_path = ['time',]
        request.opt.observe = 0
        request.remote = (ip_address('192.168.0.115'), coap.COAP_PORT)
        d = protocol.request(request,
            observeCallback=self.printLaterResponse,
                                observeCallbackArgs=('*** OBSERVE NOTIFICATION
BEGIN ***',),
                                observeCallbackKeywords={'footer':'*** OBSERVE
NOTIFICATION END ***'})
        d.addCallback(self.printResponse)
        d.addErrback(self.noResponse)

    def printResponse(self, response):
        print '*** FIRST RESPONSE BEGIN ***'
        print response.payload
        print '*** FIRST RESPONSE END ***'

    def printLaterResponse(self, response, header, footer):
        print header
        print response.payload
        print footer

    def noResponse(self, failure):
        print '*** FAILED TO FETCH RESOURCE'
        print failure
        #reactor.stop()

log.startLogging(sys.stdout)

endpoint = resource.Endpoint(None)
protocol = coap.Coap(endpoint)
client = Agent(protocol)

reactor.listenUDP(0, protocol)
reactor.run()

```

<b>I</b>	<b>Additional Practice 3: GPIO Input mode. (Do it in your free time!)</b>
	<p>In the exercise above, we have used GPIO pin in OUTPUT mode. Develop a circuit and write necessary code to use a GPIO pin in INPUT mode to read data from a sensor and print it in your screen. [There are few sensors in the tool kit box, you can anyone of the sensors].</p> <p><b>Submit</b> screen capture of the sensor data with your code.</p>
<b>J</b>	<b>Additional Practice 4: Raspberry PI as a Network Monitoring Module. (Do it in your free time!)</b>
	<ol style="list-style-type: none"> <li>1. We have just run server and client codes in a raspberry pi.</li> <li>2. Try to run the <b>sever in raspberry pi</b> and <b>client in your laptop/PC</b></li> <li>3. Run Wireshark (<a href="http://www.wireshark.org">www.wireshark.org</a>) in the laptop to capture the CoAP communication packets between server and client.</li> <li>4. Analyse the contents of CoAP GET request and response for the 'counter' resource.</li> <li>5. <b>Save &amp; submit</b> the captured packets in pcap or pcapng format</li> <li>6. Submit a text file with the following information. <ul style="list-style-type: none"> <li>○ State the message ID of the CON message with GET method sent to the CoAP server.</li> <li>○ Stste the message ID and payload value of the response message containing the value of 'counter'</li> </ul> </li> </ol>

The End \*\*\* Have Fun!