PROJECT -4

SUJAY PREMKUMAR

Experimental Evaluation of the CoAP, HTTP Transport Services for Internet of Things

ABSTRACT

Internet of Things has revolutionized the world with its new offerings to the technology. It's going to impact every business models and the way of life in the future. The main enablers of the technology are the core modules of the IoT, such as the communication and Transport services involved in sending the huge number of data from the sensors to the decision-making service may in in cloud. In the traditional network and web HTTP plays an important role in handling the application layer tasks of interacting with the users and having setup the communication medium between them. In the IoT world HTTP has huge drawbacks considering the heavy weight protocol, and runs over TCP. Since the CoAP, is light weight an runs and UDP and header length of the CoAP is very less, it fits perfectly for the IoT communication. In this project, I tried to give the experimental explanation and evaluation of the COAP and HTTP.

Keywords: CoAP, HTTP, throughput,

1. Description of the Experimental setup:

CoAP Framework:

I used the libcoap C implementation defined in the RFC 7252 and https://libcoap.net/

Compile Setup of the CoAP Test bed:

1. Latest development releases are available at, https://github.com/obgm/libcoap
2. If you have cloned the sources via git, you need to run **./autogen.sh** to generate the build scripts. This step is optional when a source archive is used. Next, say **./configure** followed by **make** and **make install**.
3. Besides common GNU options, the configure script provides the following switches to control the build process:

4. --enable-tests Will build the provided unit tests. This option requires **cunit** and its header files.
5. --enable-examples Will build the included POSIX examples.
6. --enable-documentation Will generate documentation for libcoap provided that **doxygen** or **a2x** are available.
7. Run the Coap-server as shown below,

//

```
coap-server(5)
==============
:doctype: manpage
:man source:   coap-server
:man version:  @PACKAGE_VERSION@
:man manual:   coap-server Manual

NAME
-----
coap-server - CoAP Server based on libcoap

SYNOPSIS
--------
*coap-server* [*-A* addr] [*-g* group] [*-p* port] [*-v* num]

DESCRIPTION
-----------
*coap-server* is a CoAP server which simulate 6LoWPAN devices which can be
addressed via the CoAP protocol.

OPTIONS
-------
*-A* addr::
  The local address of the interface which the server has to listen.

*-g* group::
  Join specified multicast 'group' on startup.

*-p* port::
  The 'port' on the given address the server will be waitung for connections.
  The default port is 5683 if not given any other value.

*-v* num::
  The verbosity level to use (default: 3, maximum is 9).

EXAMPLES
--------
* Example
----
```

coap-server -A ::1

Let the server listen on localhost (port 5683).

* Example
---
coap-server -A ::1 -p 13011

Quite the same, except listening port is '13011' (and not the default port 5683).

* Example
---
coap-server -A 2001:db8:81a8:0:6ef0:dead:feed:beef -v 5

The listening address is set to '2001:db8:81a8:0:6ef0:dead:feed:beef' and the verbosity level is set to '5'.

* Example
---
coap-server -A 2001:db8:81a8:0:6ef0:dead:feed:beef -g FF02:FD

Set listening address to '2001:db8:81a8:0:6ef0:dead:feed:beef' and join the All CoAP Nodes multicast group 'FF02:FD'.

FILES

There are no configuration files.

EXIT STATUS

0::
  Success

1::
  Failure (syntax or usage error; configuration error; document processing failure; unexpected error)

//

I have a linux machine with the IP: 10.0.0.73. I am executing the coap-server as

./coap-server -A 10.0.0.73 -p 5683

And, change the coap-client code to send the NON- conformable GET request to the server. Server and Client code is shown in the end.

//

coap-client(5)
==============
:doctype: manpage
:man source: coap-client
:man version: @PACKAGE_VERSION@
:man manual: coap-client Manual

NAME
-----
coap-client - CoAP Client based on libcoap

SYNOPSIS
----
coap-client [*-A* type1, _type2_ ,...] [*-t* type] [*-b* [num,]size]
[*-B* seconds] [*-e* text] [*-f* file] [*-m* method] [*-N*]
[*-o* file] [*-P* addr[:port]] [*-p* port] [*-s* duration]
[*-O* num,text] [*-T* token] [*-v* num] [*-a* addr] [*-U*] URI

DESCRIPTION
----
coap-client is a CoAP client to communicate with 6LoWPAN devices via the protocol CoAP (RFC 7252) using the URI given as argument on the command line. The URI must have the scheme 'coap' (or 'coaps' when coap-client was built with support for secure communication). The URI's host part may be a DNS name or a literal IP address. Note that, for IPv6 address references, angle brackets are required (c.f. EXAMPLES).

OPTIONS

-a addr::
The local address of the interface that has to be used.

-b [num,]size::
The block size to be used in GET/PUT/POST requests (value must be a multiple of 16 not larger than 1024 as libcoap uses a fixed maximum PDU size of 1400 bytes). If 'num' is present, the request chain will start at block 'num'. When the server includes a Block2 option in its response to a GET request, coap-client will automatically retrieve the subsequent block from the server until there are no more outstanding blocks for the requested content.

-e text::
Include text as payload (use percent-encoding for non-ASCII characters).

-f file::
  File to send with PUT/POST (use '-' for STDIN).

-m method::
  The request method for action (get|put|post|delete), default is 'get'.
  (Note that the string passed to *-m* is compared case-insensitive.)

-o file::
  A filename to store data retrieved with GET.

-p port::
  The port to listen on.

-s duration::
  Subscribe to the resource specified by URI for the given 'duration' in
  seconds.

-t type::
  Content format for given resource for PUT/POST. 'type' must be either
  a numeric value reflecting a valid CoAP content format or a string
  describing a registered format. The following registered content format
  descriptors are supported, with alternative shortcuts given in
  parentheses:

    text/plain (plain)
    application/link-format (link, link-format)
    application/xml (xml)
    application/octet-stream (binary, octet-stream)
    application/exi (exi)
    application/json (json)
    application/cbor (cbor)

-v num::
  The verbosity level to use (default: 3, maximum is 9).

-A type::
  Accepted media types as comma-separated list of symbolic or numeric
  values, there are multiple arguments as comma separated list
  possible. 'type' must be either a numeric value reflecting a valid
  CoAP content format or a string that specifies a registered format as
  described for option *-t*.

-B seconds::
  Break operation after waiting given seconds (default is 90).

-N ::
  Send NON-confirmable message. If option *-N* is not specified, a
  confirmable message will be sent.

-O num,text::
  Add option 'num' with contents of 'text' to the request.

-P addr[:port]::
  Address (and port) for proxy to use (automatically adds Proxy-Uri option
  to request).

-T token::
  Include the 'token' to the request.

-U ::
  Never include Uri-Host or Uri-Port options.

EXAMPLES
----
* Example
----
coap-client coap://coap.me

Query resource '/' from server 'coap.me' (via the GET method).

* Example
----
coap-client -m get coap://[::1]/

Query on localhost via the 'GET' method.

* Example
----
coap-client -m get coap://[::1]/.well-known/core

Quite the same, except on the resource '.well-known/core' on localhost.

* Example
----
echo -n "mode=on" | coap-client -m put \
coap://[2001:db8:c001:f00d:221:2eff:ff00:2704]:5683/actuators/leds?color=r -f-
----
Send text 'mode=on' to resource 'actuators/leds?color=r' on the endpoint with
address '2001:db8:c001:f00d:221:2eff:ff00:2704' and port '5683'. Note that the
port '5683' is the default port and isn't really needed to append.

* Example
----
coap-client -m put coap://[fec0::3]/ck -T 3a -t binary -f to_upload
----
Put the contents of file 'to_upload' with content type 'binary' (i.e.
application/octet-stream) into resource 'ck' on 'fec0::3' by usage of a token

'3a' via the 'PUT' method.

FILES

There are no configuration files.

EXIT STATUS

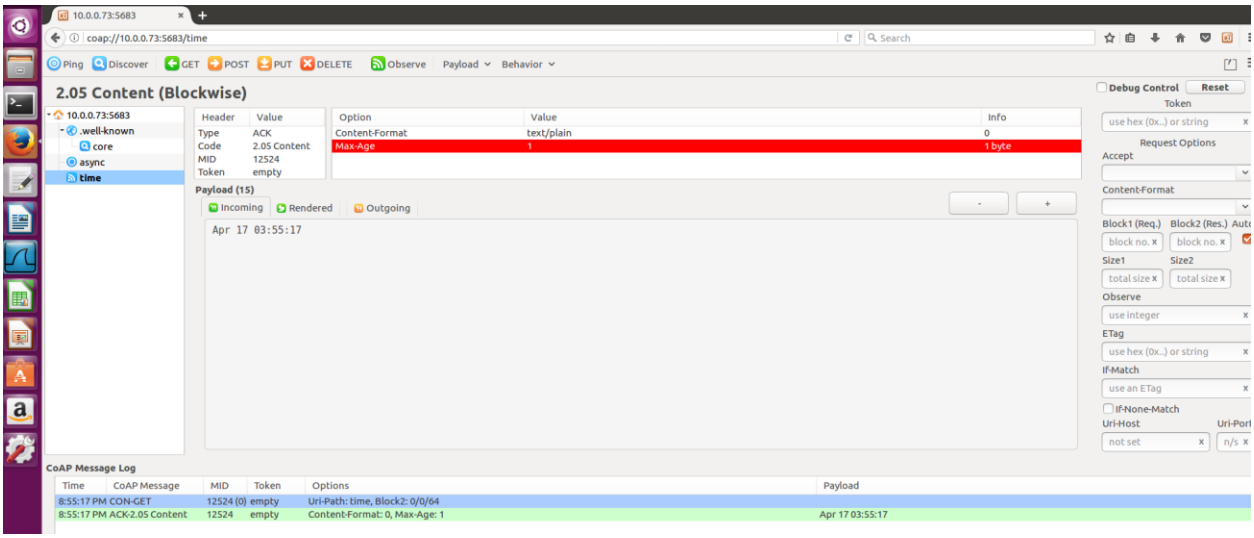
0::
  Success

1::
  Failure (syntax or usage error; configuration error; document
  processing failure; unexpected error)

//

./coap-client -m get coap://10.0.0.73:5683 -N

8. Install the Copper Cu plugin in the Mozilla which is the client plugin implementation.
   Below shows the cooper CoAP client able to discover the CoAP server and able to call GET on time function.



HTTP Framework Setup;

Install apache server in any of the linux machine. Python client script to call GET request is shown at the end.

2. Experimental Results:

CoAP Evaluation:

Run the CoAP server and client which uses the libcoap implementation and on Ubuntu 16.04 machine. Both the machines are connected to the local router via Wifi. Enable ping on both the machines.

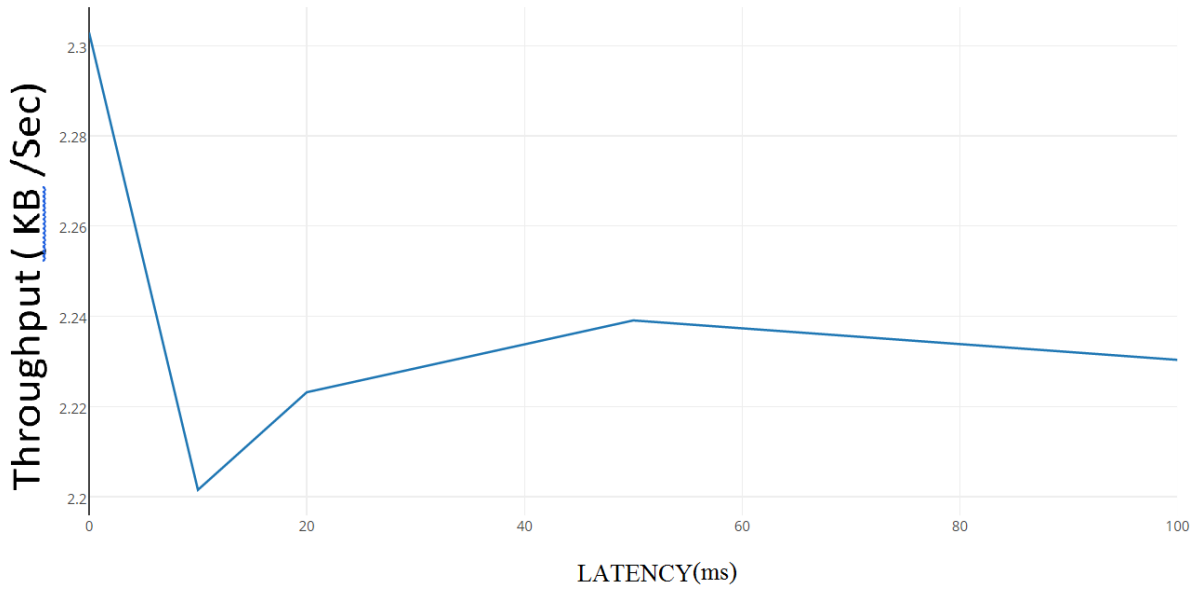
Start the server, client as shown above. Start the Wireshark to capture the packets flow. Throughput is calculated by taking the cumulative Bytes received over the period of 60sec and dividing the Cumulative bytes' value by 60sec.

Impairment type	Throughput (Bytes/ sec)
• Without any Impairments	138828 / 60.067477802 = **2302.876**
• With 10ms delay	130804 / 60.110259988 = **2176.067**
• with 20ms delay	133636 / 60.110900422 = **2223.1575**
• with 50ms delay	135052 / 60.315911331 = **2239.077**
• with 100ms delay	134108 / 60.128954162 = **2230.340**
• with loss 1%	135996 / 60.008569538 = **2266.276**
• with loss 2%	130804 / 60.111930149 = **2176.007**
• with 5% loss	118532 / 60.110009171 = **1971.917**
• with 10% loss	109948 / 60.037060877 = **1831.335**

COAP-UDP THROUGHPUT IS NOT IMPACTED BY LATENCY

UDP is a protocol used to carry data over IP networks. One of the principles of UDP is that we assume that all packets sent are received by the other party (or such kind of controls is executed at a different layer, for example by the application itself).

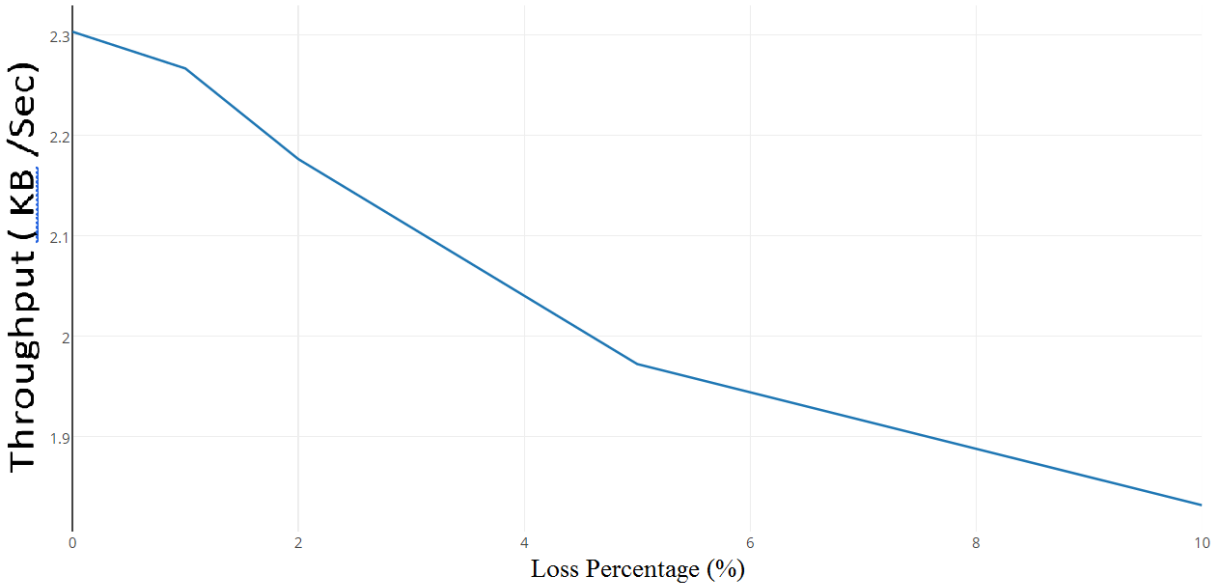
In theory or for some specific protocols (where no control is undertaken at a different layer – e.g. one-way transmissions), the rate at which packets can be sent by the sender is not impacted by the time required to deliver the packets to the other party (= latency). Whatever that time is, the sender will send a given number of packets per second, which depends on other factors (application, operating system, resources, …).



COAP-UDP THROUGHPUT IS IMPACTED BY LOSS

In UDP, throughput is affected by the loss since there is no retransmission, loss causes the packets to be dropped and not retransmitted resulting in less throughput.

As seen from the table above the throughput is almost same for the latency introduced communication. And, throughput keeps decreasing as the %loss level is increasing.



HTTP Evaluation:

HTTP-TCP DIRECTLY IMPACTED BY LATENCY

TCP is a more complex protocol as it integrates a mechanism which checks that all packets are correctly delivered. This mechanism is called **acknowledgment**: it consists in having the receiver sending a specific packet or flag to the sender to confirm the proper reception of a packet.

TCP CONGESTION WINDOW

For efficiency purposes, not all packets will be acknowledged one by one: the sender does not wait for each acknowledgment before sending new packets. Indeed, the number of packets which may be sent before receiving the corresponding acknowledgement packet is managed by a value called TCP congestion window.

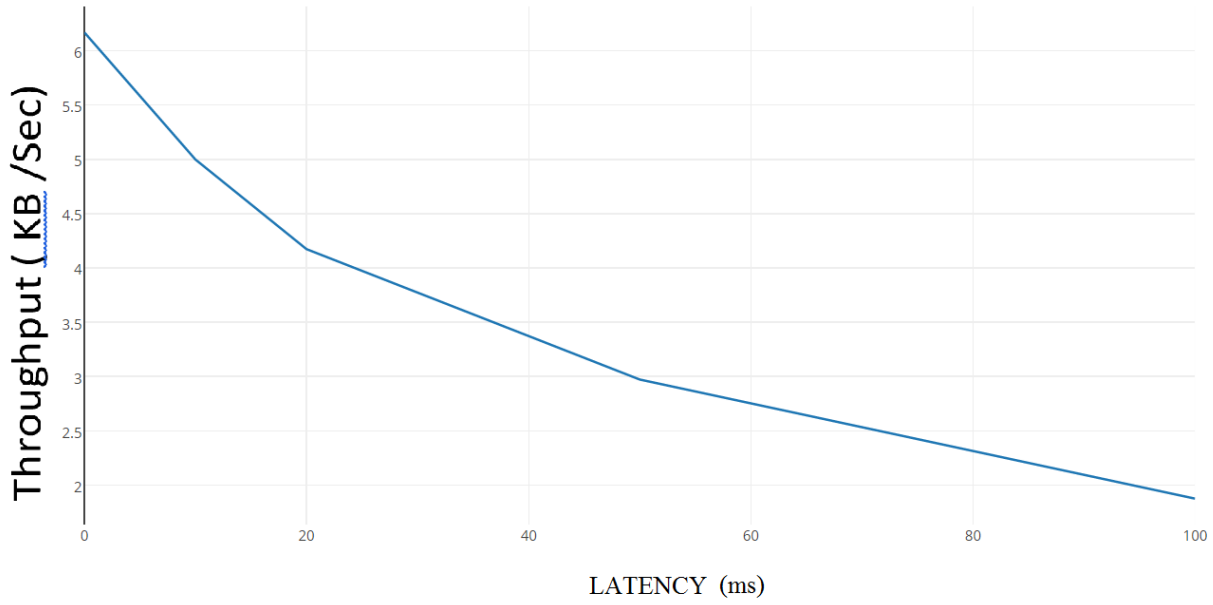
HOW THE TCP CONGESTION WINDOW IMPACTS THE THROUGHPUT

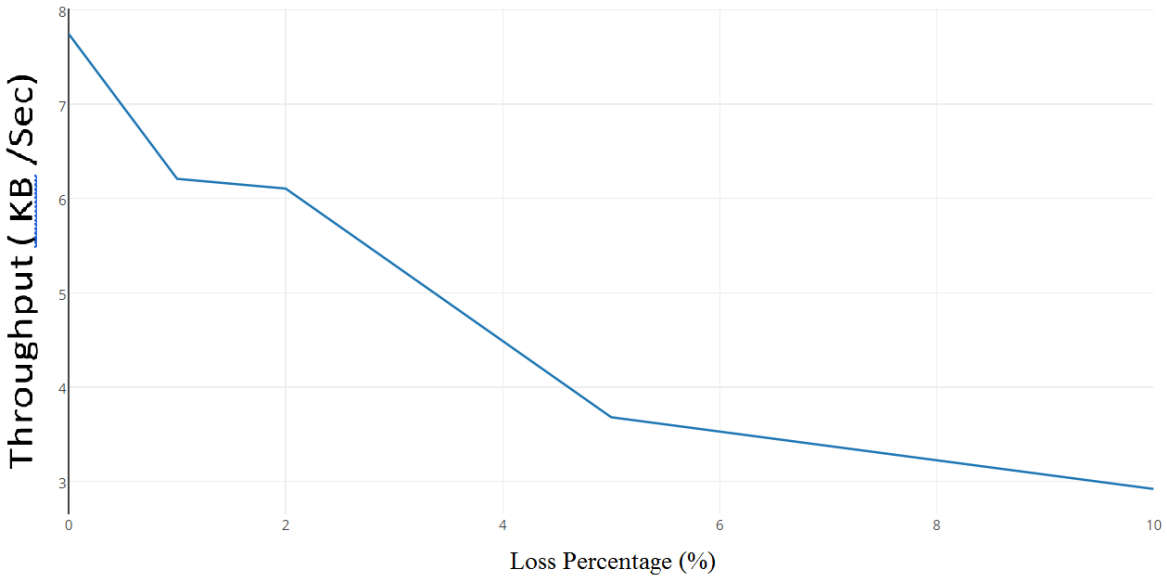
If we make the hypothesis that no packet gets lost; the sender will send a first quota of packets (corresponding to the TCP congestion window) and when it will receive the acknowledgment packet, it will increase the TCP congestion window; progressively the number of packets that can be sent in each period will increase (throughput). The delay before acknowledgment packets are received (= latency) will have an impact on how fast the TCP congestion window increases (hence the throughput).

When latency is high, it means that the sender spends more time idle (not sending any new packets), which reduces how fast throughput grows.

Impairment type	Throughput (Kilo Bytes/ sec)
• Without any Impairments	370645 / 60.057818108 = **6.171**
• With 10ms delay	300852 / 60.182116673 = **4.999**
• with 20ms delay	250600 / 60.045578301 = **4.173**
• with 50ms delay	178564 / 60.083286746 = **2.971**
• with 100ms delay	115710 / 61.753968561 = **1.873**
• Without any Impairments	464927 / 60.015799819 = **7.746**
• with loss 1%	378892 / 61.028558678 = **6.208**
• th loss 2%	375454 / 61.497204546 = **6.105**
• with 5% loss	226029 / 61.420321063 = **3.680**
• with 10% loss	180130 / 61.698690804 = **2.919**

Since the HTTP uses the TCP with the latency and loss there will be retransmissions, which causes the throughput to go down, as depicted in the graph below.


Throughput (KB /Sec)
6
5.5
5
4.5
4
3.5
3
2.5
2
0
20
40
60
80
100
LATENCY (ms)


Throughput (KB /Sec)
8
7
6
5
4
3
0
2
4
6
8
10
Loss Percentage (%)

3. Conclusion and Future Work:

- HTTP is primarily designed for Internet devices, while CoAP is designed for resource constrained devices (e.g. used in IoT, WSN, and M2M comm.).
- HTTP uses TCP, CoAP uses UDP.
- HTTP uses IP (at N/w layer), while CoAP uses IPv6 with 6LoWPAN.
- HTTP does not support multicast, while CoAP does (bcoz of UDP).
- HTTP work on client/server model, while CoAP can use both Client/server and Publish/Subscribe Model.
- HTTP needs synchronous communication, while CoAP does not.
- HTTP is complex and has large overhead compared to CoAP.

So, in summary: CoAP is optimized to be small, light and fast for M2M/IoT applications. Like HTTP, it has very little logic/semantics embedded in it, and instead uses a limited command set to communicate with RESTful server processes. Unlike HTTP, it's designed to work on the UDP (or a UDP-like) protocol, instead of TCP/IP. As expected, our experiments show that CoAP has the lowest overhead and affected by the impairments and the least number of bytes transferred compared to that of HTTP.

The transport services are compared based on object the total number of transferred bytes, and the introduced overhead. The experiments are performed in an emulated setup using Linux tc network emulator. Future work involves the changes required to perform these experiments in the real traffic and real time scenarios to check the performance. May integrate the sensors with CoAP and HTTP client-server and perform communication in the real time.

Source Code:

Source code for Coap server client and HTTp client can be found in the git hub repository

https://github.com/Sujaydas/IoT/tree/master/CoAP-HTTP_Performance