CSC 591/791 ECE 592/792: IoT Architecture, Application, and Implementation
Spring 2022
Assignment 2

G10 Members:
Chaitanya Pawar
Pavel Koprov
Rachana Kondabala
Rishi Patel
Sajal Kaushik
Thomas Batchelder

Table 1: Team Contribution

Subtasks	Chaitanya Pawar	Pasha Koprov	Rachana Kondabala	Rishi Patel	Sajal Kaushik	Thomas Batchelder
Team Meetings	16.66	16.66	16.66	16.66	16.66	16.66
Maintaining GitHub Repository	16.66	16.66	16.66	16.66	16.66	16.66
CoAP Code Development	0	0	0	100	0	0
CoAP Code Testing	0	50	0	0	50	0
CoAP Conducting Experiment	0	0	0	100	0	0
HTTP Code Development	0	0	0	0	50	50
HTTP Code Testing	0	0	0	0	50	50
HTTP Conducting Experiment	0	0	0	0	100	0
MQTT Code Development	0	50	0	0	0	50
MQTT Code Testing	0	35	20	0	10	35
MQTT Conducting Experiment	0	50	0	0	0	50
Leading the Team	100	0	0	0	0	0
Assignment Report	16.66	16.66	16.66	16.66	16.66	16.66
Preparing Final Submittal	16.66	16.66	16.66	16.66	16.66	16.66
Raw score:	166.64	251.64	86.24	266.64	326.64	301.64
% Contribution	12%	18%	7%	19.0%	22%	22%

1. INTRODUCTION

In this experiment, various application-layer protocols were tested to identify the benefits and drawbacks of each communication protocol. The protocols that were tested include HTTP, CoAP, MQTT (QoS1 and QoS2). The performance of each protocol was measured based on how long it took to transfer different sizes of data. For the experiment, four files with different sizes were used. The files were 1 MB, 10 KB, 10 MB, 100 B. The amount of overhead associated with the header of each transfer was also evaluated for the performance of all the protocols.

2. MQTT Experiment

2.1. Development & Testing

Paho MQTT library was used to implement the MQTT publisher and subscriber. The broker was implemented using Oracle Mosquitto on one of Ubuntu's PCs. MQTT works using the pub/sub model with a broker/server acting as the middleman. The publisher will publish its starting time, QoS, and file count to the subscriber to calculate the metrics accurately. Once the subscriber has received the correct amount of files, it will print the total time, average message time, the standard deviation for the message time, average kilobits per second, the standard deviation of the kilobits per second. The maximum number of messages that could be inflight by the publisher was set to 1 from the default 20. A value of 10000 was also tested, resulting in a high throughput but a significant drop in the number of messages received by the broker. An important point of discussion was determining the time measurements for the file transfer. The time was calculated from when the publisher first published the message to when the subscriber received the final message.

2.2. Experimental Analysis

MQTT has a better throughput when compared to CoAP. Total application layer data transferred from sender to receiver (including header content) per file divided by the file size is pretty much the same for QoS1 and QoS2. MQTT produced consistent results, except for QoS 1 with 100 B. This seemed to be an issue with the broker or possibly modifying the in-flight messages.

3. CoAP Experiment

3.1. Development & Testing

CoAP implementation was done using the aiocoap python library. CoAP is like HTTP as it is a client-server protocol. It also uses REST architecture. Two files were created, one as a client and a server. The client made a get request to the server, which served the data on different URI's respectively. The file transfer time started when the request was made and ended when the client received the data.

3.2. Experimental Analysis

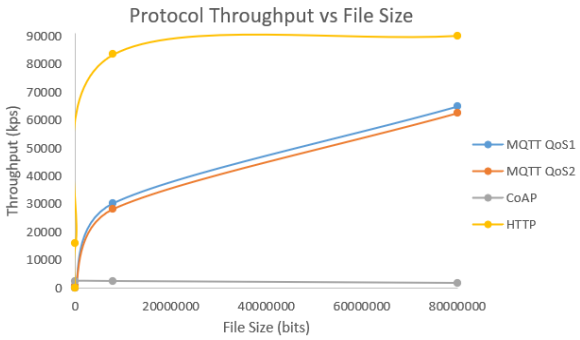
CoAP has the lowest throughput compared to MQTT and HTTP. Since CoAP utilizes UDP to transfer data, high packet error rates increase the file transfer time. CoAP also had the most consistent throughput and lowest standard deviation compared to the other protocols. CoAP divides packets into small segments and uses UDP to send packets, so the file transfer time is slow. Due to this, the throughput is also very consistent since the data is always sent in small chunks, so file size is not a factor. CoAP is also the fastest with 100 bytes since the header data is small and acknowledgment of packets is not as strict.

4. HTTP Experiment

HTTP implementation was done by using the *requests* library in python. One computer was the server, and another computer was the client. The server is python's inbuilt HTTP server, and the client makes a get request to get the data. Experimental results for HTTP suggest that the amount of time needed to transfer the file increases as the file size increases, regardless of the decrease in the number of file transfers. As the file size increases, the total data which is transferred also increases. Larger size data like 10 MB takes longer time to transfer, therefore it's broken down into packets by the transport layer and sent over the network. Integration of these packets takes time on the receiver side. HTTP protocol shows the largest total

application layer data transferred from sender to receiver per file size as its headers are in the text format, but not bytes.

5. Protocol Comparison



5.1. 10 MB

Based on the graph, it can observe that HTTP has the highest throughput. MQTT is also fast as it uses a single connection for sending multiple messages. The lowest is in the CoAP as it suffers from big file transfers. Overhead of CoAP is the least followed by MQTT. HTTP has the highest overhead.

5.2. 1 MB

For 1 MB, the highest throughput is in HTTP protocol followed by MQTT QoS 1 and then MQTT QoS 2 and then CoAP. This is due to the fact CoAP is built for the transmission of smaller files, so it must divide the large files into smaller packets and then add headers to each packet. As CoAP is using UDP, some packets are dropped and CoAP must resend them. HTTP has the highest overhead in this experiment.

5.3. 10 KB

In this experiment, the HTTP still has the highest throughput, but now CoAP takes the second place. Overhead for the CoAP is the least, followed by MQTT (QoS 1 and QoS 2) and then highest for HTTP.

5.4. 100 B

CoAP shows the best performance in sending the smallest files with the throughput more than 5 times larger than MQTT. HTTP is the second fastest in this experiment. The low MQTT performance can be explained by the technology of how we measured the time: we had to reduce the inflight message count to 100.

6. Conclusion

As a corollary of this experiment, we can say that the MQTT shows very good results in sending medium to large files with small overhead, meaning the bandwidth of the network won't be clogged with useless data. CoAP shows the best performance while sending small files in situations with a very reliable network. In addition, it has the lowest overhead, resulting in a better energy and memory efficiency. HTTP shows the best throughputs in almost all scenarios but has the largest overhead which is not the best case for populated networks.

As a disadvantage for server/client models, they all require reliable network connection because absence of the response from the server results in the data not being transmitted until the next request is sent.