[Group 11] Assignment #2: Protocol Analysis

1. Team Member Details:

1)	Priyam Garg	pgarg6@ncsu.edu
2)	Divyang Doshi	ddoshi2@ncsu.edu
3)	Brendan Driscoll	bhdrisco@ncsu.edu
4)	Jordan Boerger	jwboerge@ncsu.edu
5)	Vishal Veera Reddy	vveerar2@ncsu.edu

Percent Contribution	
Priyam Garg	20%
Divyang Doshi	20%
Brendan Driscoll	20%
Jordan Boerger	20%
Vishal Veera Reddy	20%

Tasks		**Members**				
Topic	**Sub Tasks**	**Priyam Garg**	**Divyang Doshi**	**Brendan Driscoll**	**Jordan Boerger**	**Vishal Veera Reddy**
HTTP	Code					100%
	Analysis					100%
	Report					100%
CoAP	Code			50%	50%	
	Analysis			50%	50%	
	Report			50%	50%	
MQTT QOS1	Code	50%	50%			
	Analysis	50%	50%			
	Report	50%	50%			
MQTT QOS2	Code	50%	50%			
	Analysis	50%	50%			
	Report	50%	50%			
Report		20%	20%	20%	20%	20%

We believed that every teammate did their part well and had the equal contribution in the project.

2. OBJECTIVE

In this assignment, we will be testing three communication protocols namely HTTP, CoAP and MQTT (QoS 1 and 2) used in the IoT area. The goal will be to determine which protocol suits the transfer of different amounts of data, the length of time to transfer said data, and the amount of overhead generated within the header for each subject.

3. DESCRIPTION

3.1 HTTP

To test this HTTP implementation, we used two separate laptops. A laptop was used as the HTTP server and another laptop was used as the client over a LAN connection connected by a 2 Gbps internet connection. The server laptop was a very old one and hence may have taken some extra time. Despite the challenges we have made some good observations with the protocol. The common observations from the table are:

- The time taken for data transfer increased as the file size increased even though we reduced the number of times each file was downloaded.
- The total data transferred increased as the file size increased.

The primary reason why the time taken has increased for larger files even though we transfer them a lesser number of times is because for each larger file the data would be broken down into smaller pieces to be sent over the network by the transport layer. So, a larger file would have to be broken into more chunks and hence would take longer. Similarly at the receiving end these chunks have to be integrated.

3.2 CoAP

3.2.1 Development

Our CoAP implementation uses the aio-coap library which is an already developed CoAP library for Python. We then create a server with our test files located on one device. The second device creates a client which sends a GET request for a specific file location to the server over our LAN. The server response payload consists of the contents of our requested file with the header for that message.

3.2.2 Analysis

For testing the CoAP implementation our group tried 2 different methods. First A laptop was used as the CoAP server and a raspberry pi model 2 was used as the client over a LAN connection. The raspberry pi may not have been the best device to use as a client, as it ran very slowly at points and as the experiments progressed the device seemed to slow down even more, especially across the transfer of the 1MB file compared to the 10MB file, where the throughput rate slowed down from 66.5 Kb per second to 27.6 Kb per second.

Seeing this slowdown, we decided to try again with two laptops instead of the raspberry pi. This yielded much faster results, however still notably slower than the other protocols. These were the results we included in our table. Based on the table we can see that interestingly CoAP has the highest throughput for the 100B file transfers, however it is the slowest for all the other transfers. This tells us that the CoAP protocol is very good at handling small files, however it struggles and starts to slow down significantly as file size increases. We can also see that the CoAP protocol has the least extra application layer data being transferred. From these observations we can see that while it may be slow, CoAP may work well with cheaper, low power devices that may not have much memory or storage space available, where only small packets of data would be transferred between client and server. This shows how CoAP is ideal for IoT applications where such situations are common.

While the raspberry pi’s poor performance make made it a bad candidate to use for our comparisons, seeing the diminishing returns with larger files does give us a good indication as to what types of challenges we may run into further down the line once we begin work on our class project, as we can see now the limitations we’ll face as far as data transfer rates are concerned when dealing with larger files. We’ll need to work to optimize our file sizes and package sizes if we want to run data transfer protocols on a raspberry Pi.

3.3 MQTT

3.3.1 Development

We have used three computers for the MQTT protocol implementation, one as a publisher, one as the subscriber, and one as the broker. We have also used the paho-MQTT library available in python for implementing the subscriber and the publisher and used the open-source MQTT broker Eclipse Mosquitto. Other than that, Wireshark is used for the calculation and the pack sniffing for getting the information regarding the overhead and the application-level data.

For calculating the time of the MQTT transfer, we have noted the time at two ends: one at the publisher end and the other at the subscriber end. The transfer time is calculated by finding the difference between the timer started at the publisher end and the subscriber end. We achieved this by storing the time at the publisher end and then storing the time at the subscriber end; This end time is again sent to the publisher via the MQTT protocol and strong and performing the necessary calculations at the publisher end. Other than that, we have also started capturing the packets at the publisher's end whenever the publisher starts transferring packets. This makes sure that we are capturing all the packets. Then we implemented a Wireshark packet transfer in python to read the packets data and get the total application layer data transferred from sender to receiver.

3.3.2 Analysis

The analysis and the data transfer rate are performed by running the three elements publisher, broker, and the subscriber, each at three different devices connected to the same local network.

The significant observations after conducting the results were that as the file size increased, the throughput increased because of the multiple sending iterations of the same sized file from the sender to the receiver. As the underlying protocol is TCP only, there may exist a delay in each iteration of the file transfer as the larger sized file is transferred fewer times, and also in a single message, the throughput increase with the increase in the file size. Also, the overhead decreased with the increase in the file size because as the complete data is being sent in a single MQTT protocol, the message overhead would be approximately the same for each message, and thus as the overhead is identical for every message and the file data has been sent in a single message the overhead decreases with a tremendous rate as the file size increases.

Other than that on observing the overhead, the overhead of the QoS 2 is more than that of the QOS 1, mainly because of the Publish Release message that has been sent from the sender to the receiver, and this is also calculated as the overhead, and this explains the difference between the overhead of QoS 1 and QOS2.

4. COMPARISON

4.1. 100B

- For 100B the average throughput was maximum for CoAP. The probable reasoning behind this is as CoAP uses UDP which is much light weight protocol as compared to TCP which is used by MQTT and HTTP. One more reason is that UDP is much faster when the system has high load and in this case we are transferring 100B file 10,000 times because of which there is high load on the network.

- We can also observe that highest standard deviation is observed in CoAP which is because of the UDP used underhood and we're using confirmable method for CoAP

- The total application layer data transferred from sender to receiver per file divided by the file size is minimum for CoAP. From this we can infer that overhead is minimum in case of CoAP. We can also observe that overhead is maximum in case of HTTP. This is reasonable because CoAP is made for less network overhead that can support IOT devices.

- The overhead for MQTT QoS1 is less than that of MQTT QOS2 because MQTT also send PUBREL message in case of QOS2

4.2 10KB

- In case of 10KB file, the highest throughput observed is for MQTT. Both MQTT QOS1 and QOS2 have similar throughput measurements for this case. The reasoning behind this is that MQTT works better for large file size as compared to CoAP because it uses TCP underneath it which ensures reliable delivery by itself, whereas in case of CoAP files have to be divided in segments which might get lost and needs to be re-send for a confirmable delivery.

- The reasoning behind why HTTP is slower is as it uses large packet size and it opens and close connection for each request where as MQTT can utilize a single connection to send multiple messages

- We can also observe that for MQTT QOS1 the standard deviation is high, which suggests that there might be some packets which took longer to deliver, as they might have dropped and needed to be re-sent which increased their time and hence served as outliers in the observation.

- Same as 100B file transfer, CoAP has the least overhead in this case. But, we can observe that as file size increased, the difference in overhead for CoAP and MQTT is reducing. This is because the header and overhead size remains same even when file size increases so it's impact is reduced

4.3 1MB

- We can observe that throughput is highest for MQTT, followed by HTTP which is followed by CoAP. The reasoning is similar as the file size increases CoAP's performance in terms of throughput decreases as it uses UDP and having confirmable transfer will incur more overhead in ensuring that file is successfully delivered i.e every chunk is received by the client.

- Same as 100B file transfer, CoAP has the least overhead in this case. But the difference in overhead for MQTT and CoAP is reduced significantly. HTTP has the highest overhead again as expected.

4.4 10MB

- MQTT again has the highest throughput. We can observe that throughput is considerably low in case of CoAP as CoAP struggles with big file transfers.

- In this case also, the overhead for CoAP is the least, followed by MQTT QOS1 followed by MQTT QOS2 and HTTP. The difference in overhead for CoAP and MQTT is very negligible in this case.

4.5 Overall

- Overall, the throughput for small file transfer (100B) is fastest in CoAP because of it using UDP which is faster compared to TCP. As the file size increases the performance of CoAP drops in terms of throughput and MQTT performance becomes better as TCP ensures reliable delivery on it's end. HTTP although faster than CoAP, is slower compared to MQTT because it's not able to transfer multiple messages in a single connection.

- Overhead for CoAP is least in all cases, but the overhead difference becomes negligible as the file size increases between CoAP and MQTT

- We also observe that standard deviation for CoAP is small which tells us that outliers are low as compared to MQTT (some files take longer to transfer)