ANALYSIS OF THE PERFORMANCE OF MQTT WITH SHARED DICTIONARY COMPRESSION (SDC) IN IOT NETWORKS.

Sathiyakumar Vinyagamany

(148467L)

Master of Science/ Master of Engineering

Department of Electronics and Telecommunication Engineering

University of Moratuwa

Sri Lanka

January 2019

ANALYSIS OF THE PERFORMANCE OF MQTT WITH SHARED DICTIONARY COMPRESSION (SDC) IN IOT NETWORKS.

Sathiyakumar Vinayagamany

(148467L)

Thesis/Dissertation submitted in partial fulfillment of the requirements for the degree Master of Science/ Master of Electronics and Automation Engineering

Department of Electronics and Telecommunication Engineering

University of Moratuwa

Sri Lanka

January 2019

Declaration page of the candidate & supervisor

"I declare that this is my own work and this thesis/dissertation^{2} does not incorporate without acknowledgement any material previously submitted for a Degree or Diploma in any other University or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to University of Moratuwa the non-exclusive right to reproduce and distribute my thesis/dissertation, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books).

Signature: Date:

Sathiyakumar Vinayagamany

The above candidate has carried out research for the Masters/MPhil/PhD thesis/ Dissertation under my supervision.

Name of the supervisor: Dr. Jayathu G. Samarawickrama

Signature of the supervisor: Date :

Acknowledgement

I would first like to thank my thesis advisor Dr. Jayathu G. Samarawickrama of the Department of Electronics and Telecommunication at University of Moratuwa, Moratuwa, Sri lanka. The door to Prof. Jayathu G. Samarawickrama office was always open whenever I ran into a trouble spot or had a question about my research or writing. He consistently allowed this paper to be my own work, but steered me in the right the direction whenever he thought I needed it.

Finally, I must express my very profound gratitude to my parents and to colleges for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them.

Thank you.

Signature: Date:

Sathiyakumar Vinayagamany

Abstract

The Internet of Things or IOT is a set of organized computing devices that are provided with exclusive identifiers and the ability to transfer data over a network without requiring human-to-human or human-to-computer interaction. The Internet of Things spreads internet connectivity beyond traditional devices like desktop and laptop computers, smartphones and tablets to a range of devices and everyday things that use embedded technology to connect and interact with the outside environment, all via the Internet. The architecture of IOT will greatly grow in the next few years and there will a big demand in the field of IOT devices performances. IHS forecasts that the IoT market will grow from an installed base of 15.4 billion devices in 2015 to 30.7 billion devices in 2020 and 75.4 billion in 2025 as shown in the Figure 1 - [2].

In order to cope up with the impending needs, we have to improve the current application protocols used in the internet of things. One of the most popular application protocols for IOT would be MQTT - **Message Queue Telemetry Transport**). As the Internet of Things ' growth explodes, the underlying fundamental protocols are changing. In particular, MQTT, or Message Queue Telemetry Transport, is now the dominant protocol for IoT globally.

MQTT is a machine-to-machine (M2M)/"Internet of Things" connectivity protocol. It has been designed as an extremely lightweight messaging protocol packaged as publish / subscribe.

MQTT is alright equipped with compression technologies like deflate. However, our goal is to further enhance compression by introducing Shared Dictionary Compression. Shared Dictionary Compression is tool which uses the redundancies in the messages to form dictionaries for frequently occurring key strings. These dictionaries are distributed among the

devices and for compression and decommission. This could greatly in terms of the compression and hence the bandwidth.

However, it must be noted that enabling Shared Dictionary Analysis would require detecting frequency of repeating keywords among the messages. This could induce additional computation on top of MQTT. Moreover, the distribution of dictionary might have adverse effects on the bandwidth.

So, we will need to find the right balance between the achievable bandwidth reduction and computation complexity. In order to find the right tradeoff between computational costs and bandwidth reduction. We will need to implement an algorithm to assess the performance and determine the right settings for the SDC to function. This could be called as the adaptive algorithm, as the settings would greatly depend on the dataset used.

In terms of our research, we will initially evaluate the Compression Potential of SDC + MQTT. After confirming the potential of SDC, we will evaluate the same with adaptive algorithm. Consequently, using the adaptive algorithm, we will find the right balance between the performance and compression, by evaluating the compression potential and computational costs of SDC-MQTT.

Finally, to further optimize, we will need to analysis the ideal data format for SDC-MQTT for fully optimized performance of SDC-MQTT.

Table of Contents

Declaration of the candidate & Supervisor i

Acknowledgements ii

Abstract iii

Table of content iv

List of Figures v

List of Tables vii

List of abbreviations viii

List of Appendices ix

1. Introduction 1
    - 1.1 Internet of Things 1
    - 1.2 Machine to Machine Communication 3
    - 1.3 MQTT 4
        - 1.3.1 MQTT- IoT 5
        - 1.3.2 Bandwidth Optimization 6
2. Research Problem 7
    - 2.1 Objective 8
3. Literature Review 9
4. System Design 10
    - 4.1 Incorporate SDC into MQTT 10
    - 4.2 Sampling Brokers 11
5. System Implementation 12
    - 5.1 "Katha" – JavaFX Application 12
    - 5.2 The Controlled Environment 14
    - 5.3 Shared Dictionary Compression over MQTT 16
    - 5.4 Performance of SDC – First Trail 18
    - 5.5 Design of Adaptive Algorithm 20
6. Results and Discussion 22
    - 6.1 Parameters Selections 22
    - 6.2 Performance of SDC – Adaptive Algorithm 26
    - 6.3 Data formats Performance Analysis 27
    - 6.4 Power Consumption 31
7. Conclusion 32
8. Future Work 34

Reference List 35

Bibliography

Appendix A: Title 37

Appendix B: Title 37

LIST OF FIGURES

		Page
Figure 1.2.1	Types of Architecture	3
Figure 1.3.1	Basics of MQTT Protocol	4
Figure 1.3.2	The Sequence of Communication of MQTT Protocol	5
Figure 4.1.1	Overlay of Shared SDC over MQTT	10
Figure 4.2.2	Functional Block Diagram	11
Figure 5.1.1	Home Screen of Application	12
Figure 5.1.2	User Interface for Device Management	13
Figure 5.1.3	Dashboard and Analytics	13
Figure 5.1.4	Bandwidth Reduction Compared to Uncompressed	18
Figure 5.1.5	Processing times Compared to deflate	19
Figure 6.1.1	Time to Compress vs Dictionary Size	22
Figure 6.1.2	Dictionary Usage vs Dictionary Size	23
Figure 6.1.3	Dictionary Usage	23
Figure 6.1.4	Bandwidth Reduction vs Dictionary Size	24
Figure 6.2.1	Processing times Compared to deflate	26
Figure 6.3.1	Compression Ratios vs Various Data Formats	28
Figure 6.3.2	The Computational complexities for various data formats.	29
Figure 6.3.3	Computational Complexity Analysis on JSON format	30
Figure 6.4.1	Power Consumption	31

LIST OF ABBREVIATIONS

Abbreviation	Description
IoT	Internet of Things
IEEE	Institute of Electrical and Electronics Engineers
MQTT	Message Queue Telemetry Transport

CHAPTER 1

INTRODUCTION

1.1 Internet of things or IOT

The Internet of Things (IoT) is the system of physical items that contain embedded technology to connect and exchange information.
This field has huge potential and is predicted there will be nearly 70 billion IOT devices in existence by the end of 2025 [9]. We are still at the initial phases or changes that IoT will bring to our world. The application for internet connected devices are widespread. The ability to network embedded devices with limited CPU, memory and power properties means that in almost every field, IoT finds applications.

An increasing portion of IoT devices are molded for domestic use. Instances of consumer applications include connected automobiles, arts, home-automation, HVAC systems, health systems, and applications such as washing/drying machines, automatic vacuums, microwaves, or fridges that use Wi-Fi technology for remote dashboards. Domestic IoT delivers new openings for user experiences and connectivity.

Some Applications which was focused include:

- **Buildings:** Industrial carriers imitate the integrated home model in large facilities using intelligent building management systems that communicate parameters such as energy usage tracked by connected electrical asset performance meters, thermostats and HVAC systems.
  Another important aspect of the monitoring of building structures is to detect physical damage caused by leaky pipes, moisture dropouts, fire damage and smoke alarms.
- **Fleet management:** Tracking of fleet usage, predictive maintenance and redefinition of routes are carried out through live information, correspondingly dependent on traffic information and weather situations.
- **Supply chain:** Instant statistics joining facts from various sources (e.g. co2 level, humidity level, temperature, check when a box is opened, package arrival time, driver fatigue) makes many opportunities for the insurance companies.

Example, FedEx offers an IoT - tracking device enabling them to track all the package's location in real time. Moreover, it can also share key parameters of a shipment in real time.

- **Manufacturing Technologies:** Allocation of Investments for researches and platforms to evaluate how engineers can use smart interconnected tools permit various mechanisms to exchange.
- **Drones:** Ariel drones are tested by guarantors for entitlements and hazard study of high-risk plans, risk calculation and assets damage.
  Even some companies use drones to study the impact caused by forest fire over a big geographical area and hasten the entitlements procedures.
- **Smart Grid**: It's a special smart-grid that assures to automatedly use data on the behavior of electricity providers and users to improve electricity efficiency, reliability and economy.
- **Smart Farming:** Intelligent agriculture is a frequently overlooked business case for the Things Internet because it does not really fit into the well-known categories of health, mobility or industry. However, the Internet of Things could revolutionize the way farmers work because of the cloud based of farming operations and the huge amount of livestock that would be monitored at the same time.
- **Smart Cities:** Smart-city covers a extensive range of applications, ranging from traffic flow management to water supply, waste management, city-security and monitoring.
- **Pharmaceutical:** A drug temperature monitoring app uses sensors to detect if the temperature of the drug exceeds the acceptable range and ensures that medical supplies still comply with quality standards when delivered. IOT-based smart applications can be used not to monitor that drugs are kept within the correct temperature range of handling, but also to remind patients when it is time to take their drugs.
- **Aeronautical :** An app for equipment tracking offers an airline engineer a live view of the locations of each maintenance device. This IoT application not only generates significant cost savings and process improvements by increasing the efficiency of engineers, but also ultimately has a more reliable impact on customer experience.

IoT is ever growing sector and has huge potential in so many fields. As the number of IoT devices and IoT platform increasing year by year, we are pushed to optimize the usage of network and computation power further and further.

1.2 Machine to Machine (M2M)

Machine to Machine sets the machine connectivity foundations on which IoT has been improved, IoT development takes place and is practically based. IoT is the bigger connectivity vision driven by advances in Machine to Machine applications [10].
Machine to Machine **‘s** main goal is to connect a device to the cloud so that business can manage and collect data remotely from the device.

Machine to Machine Communication are two types of Architecture.

1. Vertical Type
2. Horizon Type

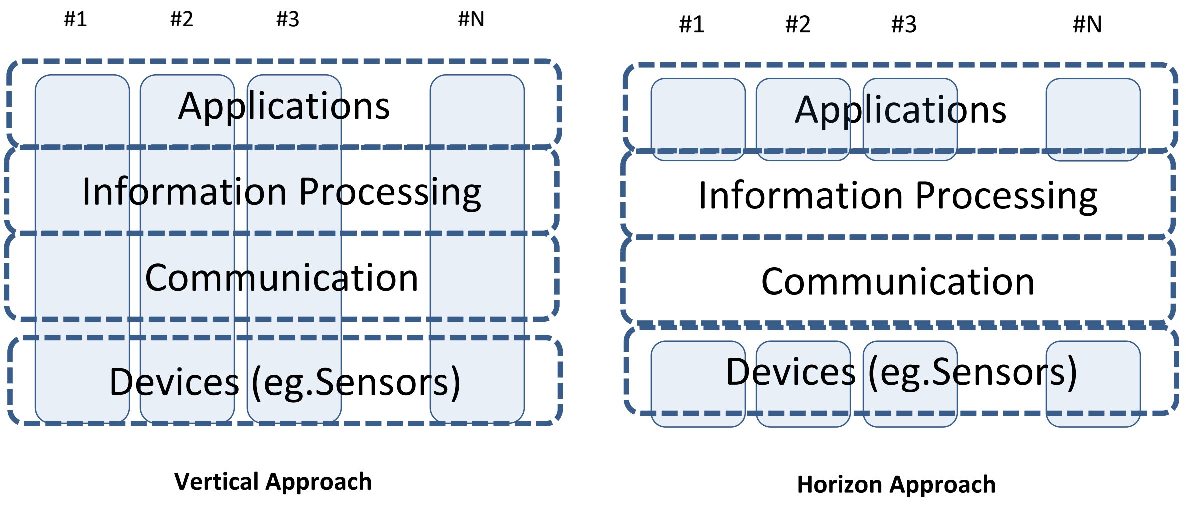


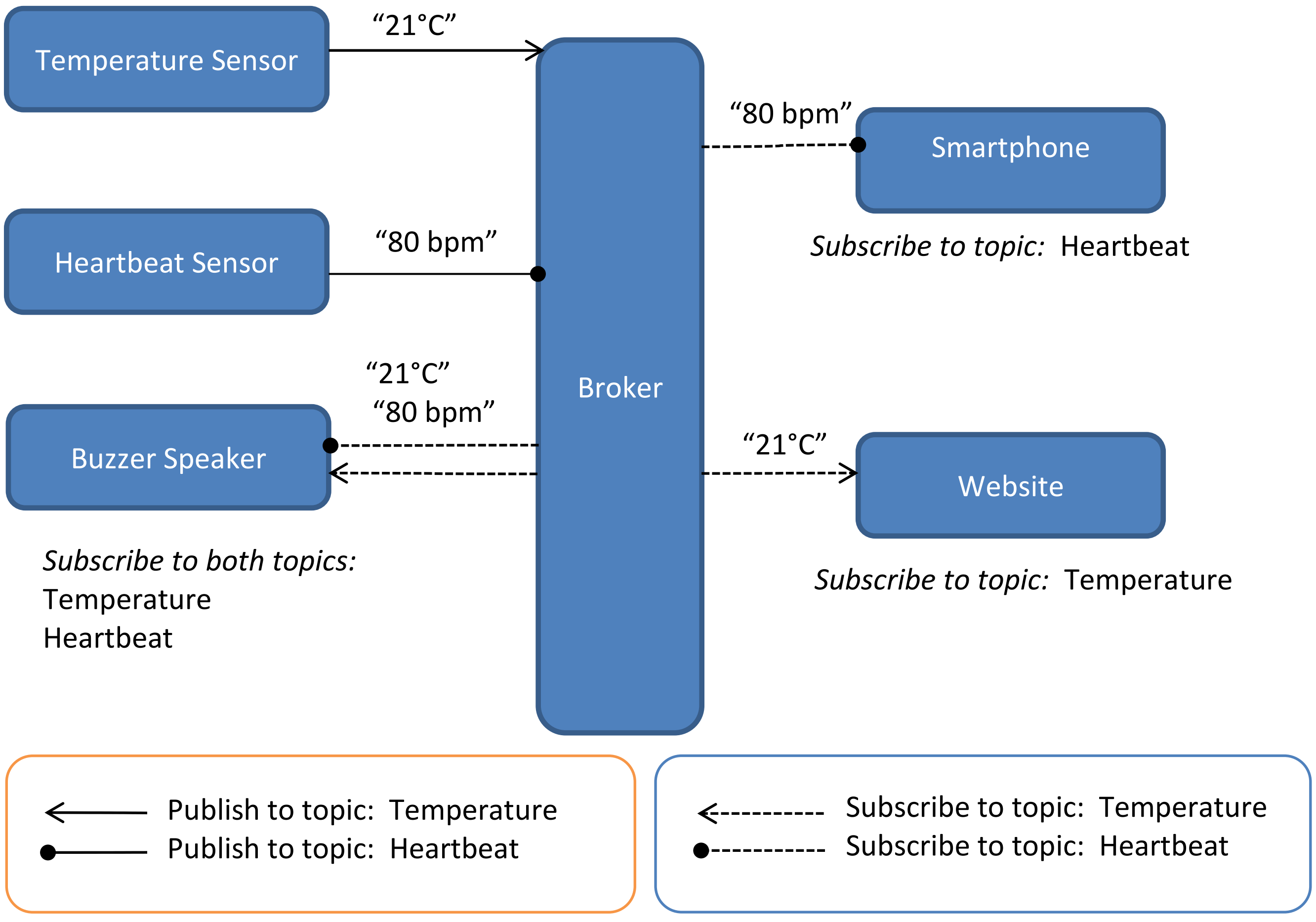
Figure 1.2.1 : Types of IOT Architecture

As shown above in Figure 1.2.1, In a vertical approach, communication systems and data processing can be improved for each application with legacy communication facilities. The use of the Horizon Approach, however, requires optimizing communication protocols in terms of scalability. If the system is a single backbone for the processing and communication of information, the system should be involved in large - scale IOT deployments.

1.3 MQTT (Message Queue Telemetry Transport)

MQTT is acronym for Message Queue Telemetry Transport. It is a publish/subscribe, lightweight and very simple messaging protocol, intended for data/ power restricted devices and low-bandwidth with networks whose latency is very high.

It scales on commodity hardware horizontally and vertically to accommodate a large amount of simultaneous consumers and publishers while maintaining low latency and fault tolerance.


Temperature Sensor
"21°C"
Heartbeat Sensor
"80 bpm"
"21°C"
"80 bpm"
Buzzer Speaker
Subscribe to both topics: Temperature Heartbeat
Broker
"80 bpm"
Smartphone
Subscribe to topic: Heartbeat
"21°C"
Website
Subscribe to topic: Temperature
Publish to topic: Temperature
Publish to topic: Heartbeat
Subscribe to topic: Temperature
Subscribe to topic: Heartbeat

s

Figure 1.3.1: Basics of MQTT protocols

The main point of communication is the MQTT broker, who sends all interconnects the messages between the right recipient and the sender. Each client that issues a memo to the broker contains a topic in the message. The subject is the directing information of the broker.

Every customer who wishes to receive messages, has to subscribe to the exact subject and the broker forwards all the messages to the corresponding subject to the customer. The customers do not therefore need to distinguish each other, they only communicate about the topic. Ideal for messaging applications like, Facebook messagers [3].

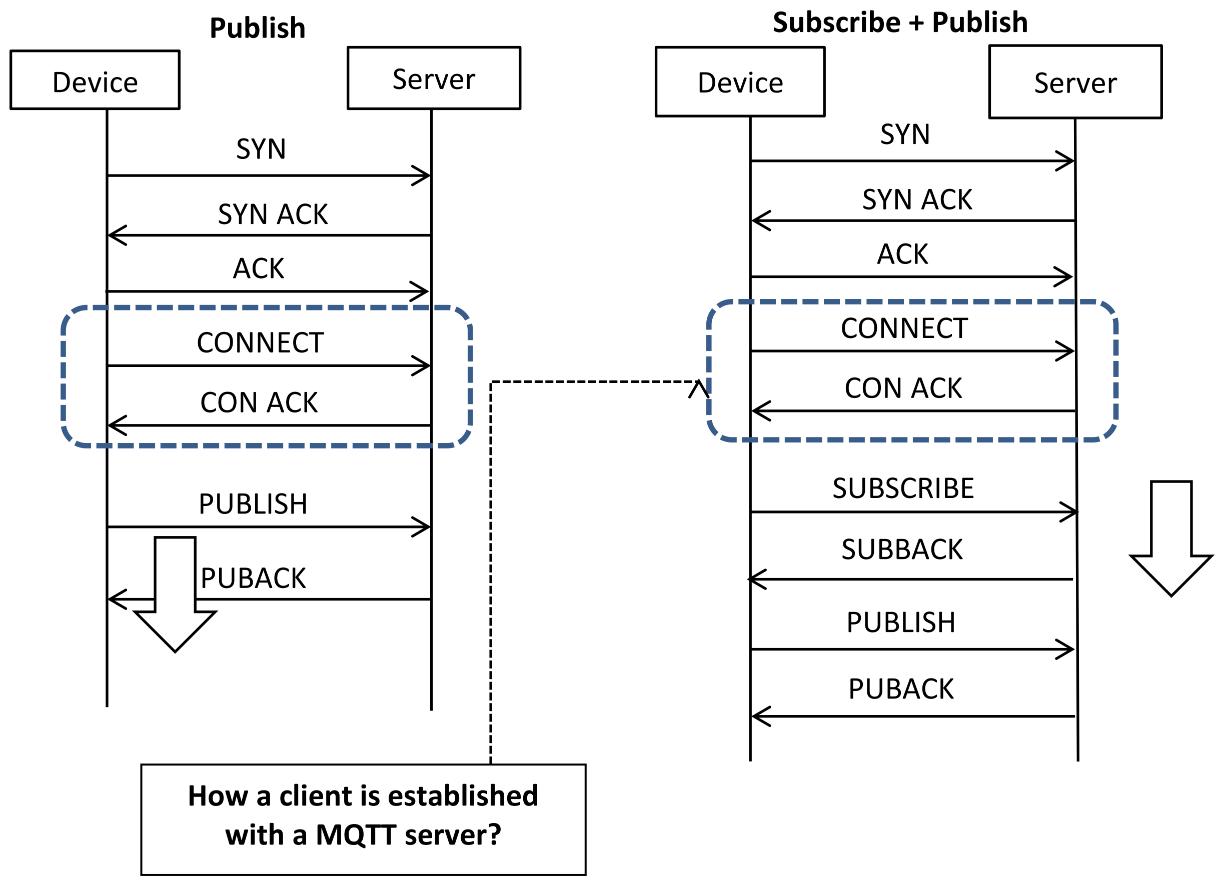


Figure 1.3.2: The sequences of communication of the MQTT protocol [11]

1.3.1 MQTT - IoT

As stated in the introduction MQTT is considered for small bandwidth, networks with low latency for which MQTT is already optimized. Its design principles are well-thought-out to minimize the need for network bandwidth and device resources, and hence it is the best protocol for building analysis. As in any building, there could be a situation where network signals are not available in the various areas of the building. While MQTT ensures reliability and some degree of delivery assurance, it is unquestionably the best choice. However, MQTT requires data networks that are frequently metered in addition to the bandwidth available in rural buildings inclines to be lesser than in urban center. Consequently, dipping the bandwidth consumption of MQTT interprets into cost reductions and allows their use even in zones where bandwidth is restricted.

1.3.2 Bandwidth Optimization

We can use methods such as **Shared Dictionary Compression.**
[6] It reduces the bandwidth required by using a dictionary that is shared between the client and server. It is best for small messages that do not match with traditional tools like gzip. The idea behind the method is to create a dictionary of long strings that appear in the same domain (or popular search results) throughout many notifications (MQTT messages). The compression searches for the appearance of the long strings in a dictionary and substitutes them with references to the said dictionary. The output is then further compressed with DEFLATE.

DEFLATE [4] is a patent free data-compression algorithm. Many open source algorithm implementations are available in the internet. The typical library for implementation most people use zlib. The zlib provides DEFLATE/INFLATE compression and decompression functions. The zlib library also has a zlib data format that accommodates DEFLATE compressed data with a header and checksum.

GZIP is alternative compression library that uses DEFLATE to compress data. Actually, most GZIP implementations use the internal zlib to perform INFLATE/DEFLATE compression procedures. GZIP creates its individual GZIP data format, which accomodates compressed DEFLATE data with a header and a checksum.

CHAPTER 2

RESEARCH PROBLEM

In this paper, in order to decrease the bandwidth further to accommodate the huge demand of IoT in the future, it is projected that a server compresses messages/topic listed by subscribers during the initial stage using a Shared Dictionary Compression tool along with traditional tools such as GZIP/DEFLATE. Then the subscribers and publishers are supplied with the produced compressed dictionaries, when new messages are published, new compressed messages can be produced using these compressed dictionaries. A common compression of the dictionary could also be used not only to compress the topics or messages, but also to compress events. SDC uses resemblances between events to improve compression ratios. In the case of IOT events, the messages could be recurring more frequently. As most events and notifications include metadata like place, units, timestamp, ids then numerous values which are understandings from the site (e.g., temperature, humidity levels, etc.)

In numerous successive notifications this metadata will be repetitive and could be endorsed to the dictionary. SDC can therefore pay off for the inefficiencies of sensory information redundancies.

Therefore, the use of SDC along with traditional compressions methods will reduce the bandwidth consumption significantly.

Nevertheless, SDC needs a vocabulary to be produced and discarded before the compression, which introduces extra bandwidth and computational-power, hence our approach to the problem will be as follows.

- Incorporate the SDC into MQTT
- Evaluate the compression potential and computational costs of the SDC – Deflate – MQTT.
- An adaptive algorithm will be developed for determining the expiry of SDC and to initiate the renewal of these dictionaries.
- Evaluation of adaptive algorithm
- Analysis the tradeoff between computational costs and bandwidth reduction.

2.1 Objective

The assessment is applied on top of a MQTT broker by means of a java application developed application for Building Analytics. The setup is implemented and studied in a controlled environment – Windows 10 – JavaFX based application.
We develop custom made dashboards and analytics tools to study and judge the feasibility of the bandwidth reduction with the cost of computation power.

In our Example we will passing information such as Zone Air Temperature, Zone Air Humidity, Traffic Information, and Locations Etc.
We will use the communicated datasets to formulate the dictionaries which will help us evaluate the compression potential and computational costs of direct MQTT and SDC synchronized MQTT, then we present an assessment of the adaptive dictionary maintenance algorithm and pin-point the tradeoff between the bandwidth reduction and computational cost in the view of the applied limits measured in the Building Analytics – IoT Research.
Our Aim:

1. **To Implement the SDC over MQTT and Adaptive algorithm in a JAVA based application.**
2. **The developed application could be used to find the right key point of saturation for bandwidth reduction, beyond which the bandwidth reduction is only insignificant and the excess computational costs becomes waste.**

Further we can evaluate the scenarios of different data format and to determine the best use case format, by

3. **Analyzing the computation times and of various data formats, by using the found trade off point, by comparison with traditional compression techniques like deflate only and then with deflate plus Shared Dictionary Compression.**

CHAPTER 3

LITERATURE REVIEW

In the ever-growing field of IoT, MQTT is major player when it comes to communication protocol. As it is extremely lightweight and simple, designed for constrained devices and for lesser bandwidths. This protocol is even suitable for high-latency or unreliable networks. Hence MQTT's versatility and lightweight design makes it the finest IoT protocol. However, the field of IoT is growing rapidly. We need to fine tune the MQTT further for more bandwidth savings to accommodate more devices in the future.

Our plan is to include a compression methodology – Shared Dictionary Compression on top of MQTT. Shared Dictionary Compression needs a dictionary to be produced and discarded prior to compression, which introduces extra bandwidth and computational-power. Bandwidth reduction is the primary goal of our project and in an IoT environment computation should be also minimal due to hardware constraints.

Nevertheless, including a new compression technology into MQTT will make it bulkier and might increase the computation complexity. Hence our role will also be extended to find the sweet spot of optimization of the integration of Shared Dictionary Compression. This is carried out by an algorithm called Adaptive Algorithm – To evaluate the compression potential and computational costs of SDC-MQTT, and to select the right cutoff between the achievable bandwidth and the computation power. We are required to develop an application with such algorithms for analyzing the bandwidth reduction vs computation complexity for practical operation. The application should be more analytic and should be based in a controlled environment. So, we choose java FX application based to be run on a window 10 environment., where we can integrate comprehensive analytical tools and all-inclusive dashboards for the study. Such analysis of the actual use of the dictionary should be comprehensive in the field of IoT. As most events and notifications contain information like place, time, dates, units, ids and numerous values which are understandings from the surroundings (e.g., temperature, humidity levels, etc.) In numerous successive messages this information will be repetitive and could be endorsed to the dictionary. SDC can consequently pay off for the redundancies of sensory data.

CHAPTER 4

SYSTEM DESIGN

4.1 Incorporate the SDC into MQTT

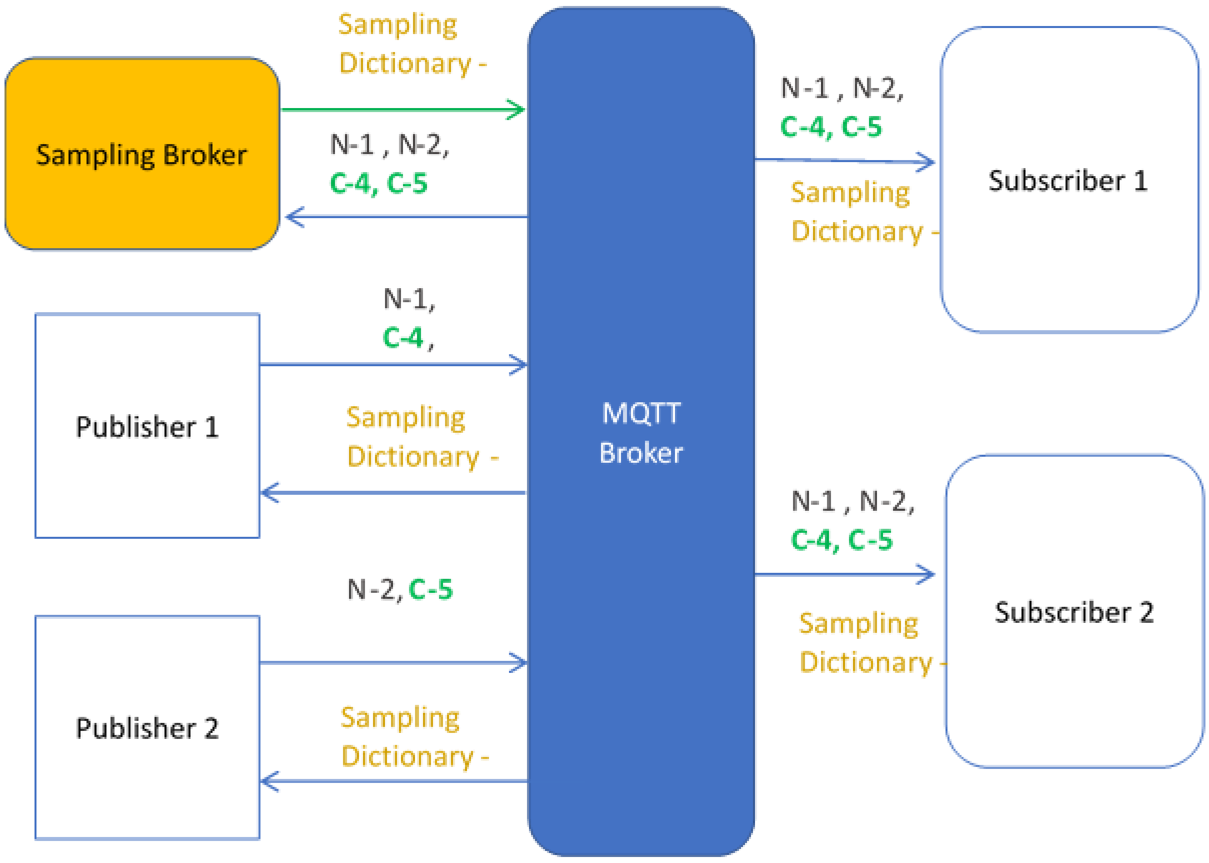


Figure 4.1.1 – Overlay of the Shared Dictionary Compression over MQTT

Firstly Publisher 1 and Publisher 2 sends messages to Subscriber 1 and Subscriber 2 – Notification 1 and Notification 2 (N1 and N2). The Sampling broker for the particular topic, samples from all Publisher 1 and Publisher 2 ‘s messages (N1 and N2) and evaluates the bandwidth reduction in case of the expressed Dictionary with N1 and N2 – [6].

If there is a significant bandwidth reduction, the formulated Dictionary is published to Publisher 1 and Publisher 2. This Sampled dictionary is used to compression new messages, ie C-4 and C5.

A method of evaluating the bandwidth reduction and determining whether or not sampled dictionary should be published or not has to be developed. Such methods could be addressed as adaptive algorithm as it will change its behavior at the time of implementation, based on formulated dictionary and its bandwidth savings. This algorithm is very important to calculate and fine tune the computation costs and the bandwidth saved.

- Publisher 1 and Publisher 2 sends messages to Subscriber 1 and Subscriber 2.
- Therefore, notification N-1 and N-2 is published.
- Adaptive algorithm detects bandwidth reduction; hence **Sampling Dictionary SD** is produced.
- Sampling Dictionary SD3 is used by the publishers to compress notification C-4 and C-5.

4.2 Sampling Brokers

An agent should sample notifications, create dictionaries, keep the dictionary over a period and spread the dictionary over the available network. We name this agent as sampling broker.
The sample broker operates an adaptive algorithm that screens the compression ratio and generates new vocabularies/dictionaries. The adaptive algorithm makes the current dictionary obsolete if any more savings could be made with a new dictionary. In our case of state of affairs, the adaptive algorithm asses the effect of a sample of the most recent notifications on bandwidth reduction. If the present bandwidth used can be reduced more, a new dictionary is then distributed over the available broker. The dictionary-size affects the compression potential and overhead bandwidth and the extent of historical messages affects the time a dictionary is sampled. The adaptive algorithm is a heuristic that attempts to balance the two parameters.

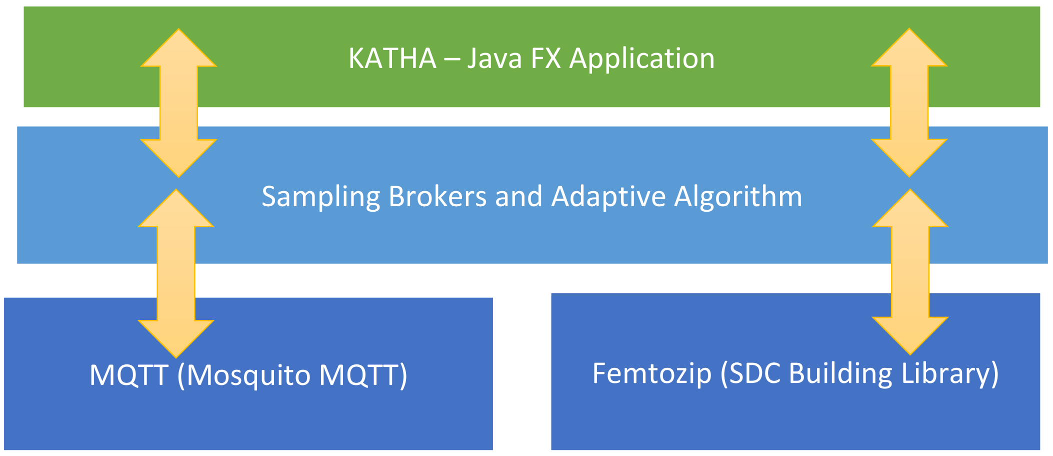

KATHA – Java FX Application
Sampling Brokers and Adaptive Algorithm
MQTT (Mosquito MQTT)
Femtozip (SDC Building Library)

Figure 4.2.2 – Functional Block Diagram

CHAPTER 5

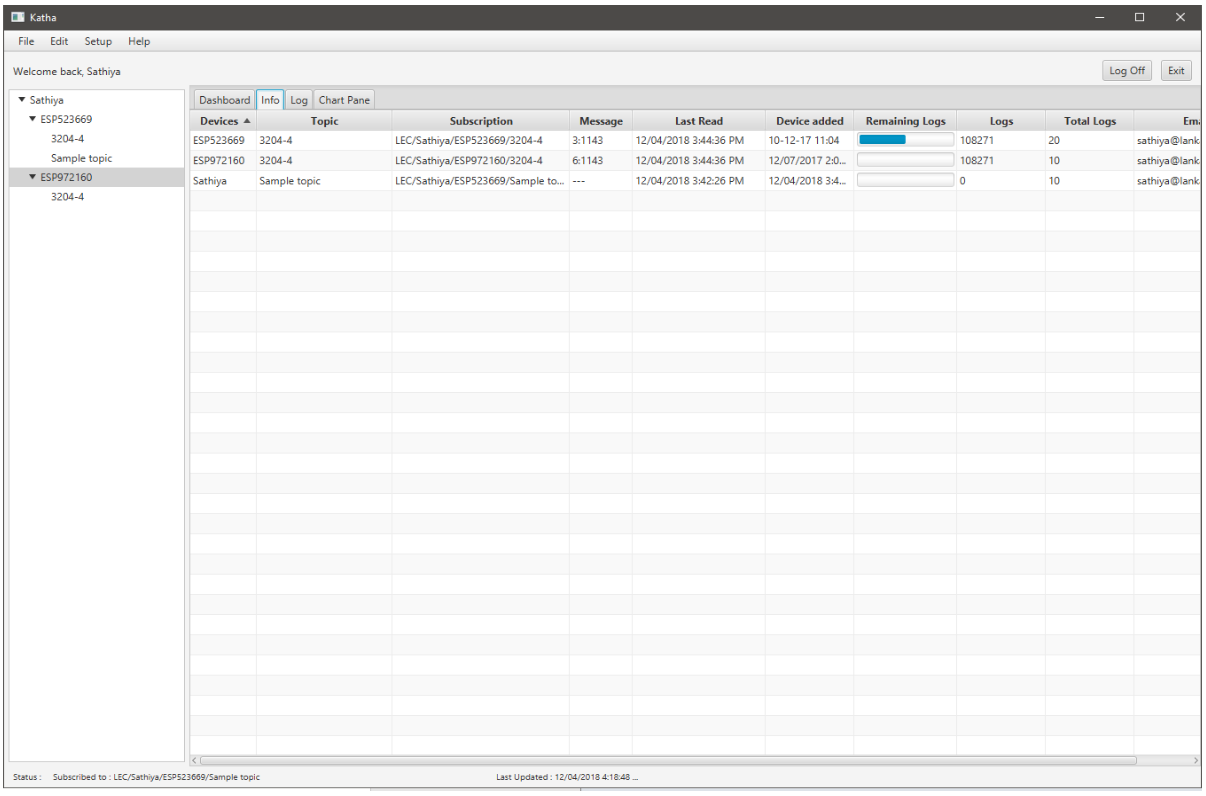
SYSTEM IMPLEMENTATION

5.1 Katha Java FX Application

Eclipse Mosquitto is an open source (EPL / EDL licensed) messaging broker implementing versions 3.1 and 3.1.1 of the MQTT protocol. Mosquitto is lightweight and can be used on all devices from single board microcontrollers to full servers with low power. The Mosquitto - MQTT protocol offers a lightweight method to perform messaging using a publishing / subscription model. This makes it suitable for messaging things like low-power sensors or mobile devices such as smart phones, embedded computer systems on the Internet.

We have used a library class known a Femtozip, for building SDC dictionary. Femtozip represents two natural extensions of Gzip and Deflate like compression schemes.

We developed a javaFX application to implement Femtozip along with a SDC library on top of the Mosquitto – MQTT.



Welcome back, Sathiya

- Sathiya
  - ESP523669
    - 3204-4
    - Sample topic
  - ESP972160
    - 3204-4

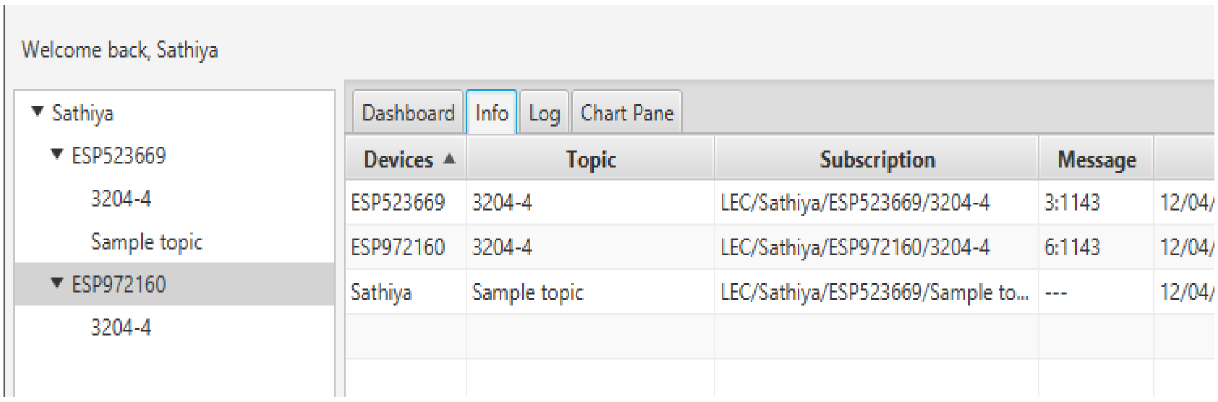
Devices	Topic	Subscription	Message	Last Read	Device added	Remaining Logs	Logs	Total Logs	Em
ESP523669	3204-4	LEC/Sathiya/ESP523669/3204-4	3:1143	12/04/2018 3:44:36 PM	10-12-17 11:04	🟦	108271	20	sathiya@lank
ESP972160	3204-4	LEC/Sathiya/ESP972160/3204-4	6:1143	12/04/2018 3:44:36 PM	12/07/2017 2:0...		108271	10	sathiya@lank
Sathiya	Sample topic	LEC/Sathiya/ESP523669/Sample to...	---	12/04/2018 3:42:26 PM	12/04/2018 3:4...		0	10	sathiya@lank

Status : Subscribed to : LEC/Sathiya/ESP523669/Sample topic

Last Updated : 12/04/2018 4:18:48 ...

Figure 5.1.1 – Home Screen of the Application

The software will be used to analyze the performance of our implementation and helps us to decide the tradeoff between the computational cost and bandwidth.



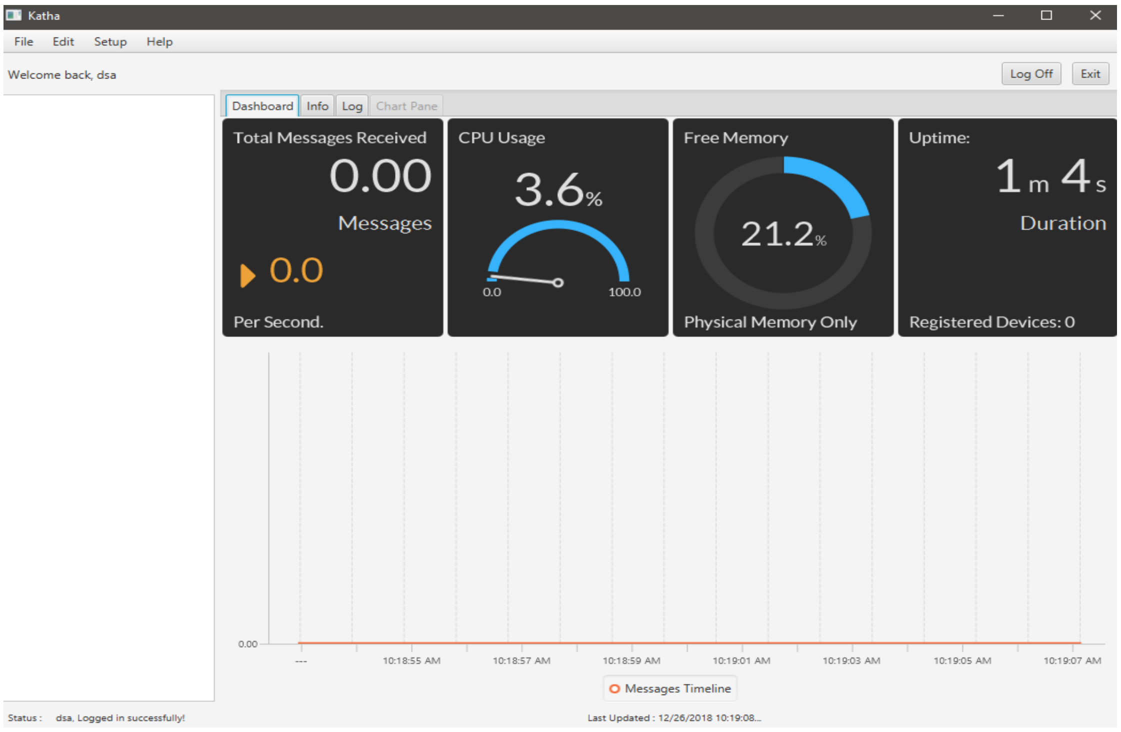
Welcome back, Sathiya

- ▼ Sathiya
  - ▼ ESP523669
    - 3204-4
    - Sample topic
  - ▼ ESP972160
    - 3204-4

Devices ▲	Topic	Subscription	Message	
ESP523669	3204-4	LEC/Sathiya/ESP523669/3204-4	3:1143	12/04/
ESP972160	3204-4	LEC/Sathiya/ESP972160/3204-4	6:1143	12/04/
Sathiya	Sample topic	LEC/Sathiya/ESP523669/Sample to...	---	12/04/

Figure 5.1.2 – User Interface for Device Management

The client library is divided into several sections. It condenses a MQTT client and FemtoZip library that is utilized for compression based on dictionaries. The SDC locates and removes current active dictionaries when expiry time is elapse. Figure 2 shows the interactions between the MQTT broker and the client. Each subject has a dictionary-topic compatibly.



Welcome back, dsa

Total Messages Received 0.00 Messages 0.0 Per Second.

CPU Usage 3.6%

Free Memory 21.2% Physical Memory Only

Uptime: 1 m 4 s Duration Registered Devices: 0

Status : dsa, Logged in successfully!

Last Updated : 12/26/2018 10:19:08...

Figure 5.1.3 - Dashboards and Analytics

5.2 The Controlled Environment

The Machine used to run the application:

1. Processor : Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz 1.80 GHz
2. Installed RAM: 8.00 GB (7.88 Stable)
3. System Type : 64-bit Operating System x64 based processor
4. Wifi : Intel(R) Dual Band Wireless – AC 8265
5. Operating System**:** Windows 10 Home Single Language

The library used to run the application

1. Mosquito MQTT – Messaging Protocol Sever.
2. Femtozip – Popular Compression Tool.
3. TilesFX – For Dashboard Preparation and Analytics.
4. Eclipse Paho MQTT – Messaging Client.
5. Medusa – For Further Dashboard Preparation and Analytics.
6. ControlsFX – For Finetuning the User Interface.

Software Used for Application Development.

1. IntelliJ IDEA 2018.3 – Community Edition
2. JavaFX Scene Builder – 2.0

Additional Software Used for Testing and Analyze.

1. MQTT Spy – 0.5.4
2. Microsoft Excel and Google Sheets

Datasets

The DEBS2015 dataset [8] contains all taxi trips within New York over a year. We changed the data set into JSON, XML and CSV to should observe the bandwidth reduction with different message formats. The DEBS2015 datasets in the various formats should deliver different bandwidth reduction with respect to the format. Generally it is expected, the XML format should introduce more overhead and hence a higher redundancy. Twitter dataset was obtained from the public records firehose twitter, is basically coordinates residing in US and it was taken in November 2016 during the presidential elections

5.3 Shared Dictionary Compression – Over MQTT

Shared Dictionary Compression is a compression protocol which identifies and uses the redundancies between the messages to create dictionaryies. These dictorayes enable the publishes and subscribers to compress and decompress the messages on transmitting and receiving the messages.

The identifying of repeated strings with in the message is done by multiple passes of Huffman coding. Huffman's coding leverages the similarities between the messages and help to create the dictionary.

A Sample Broker is responsible for generation and decimation of dictionaries within the SDC. Each topic is assigned a sampling broker. Hence dictionaries are generated per topic, in other words each topic will have their own dictionaries. These dictionaries will possess a parameter called the expiry time. Any Publishers of the same topic can use the dictionary to compress their messages before sending, and any subscribers of the same topic are allowed to use the dictionary to decompress their messages. However, after the expiry time of the dictionaries, no publisher is allowed to compress the messages or no subscribers are allowed to decompress the messages using the expired dictionary.

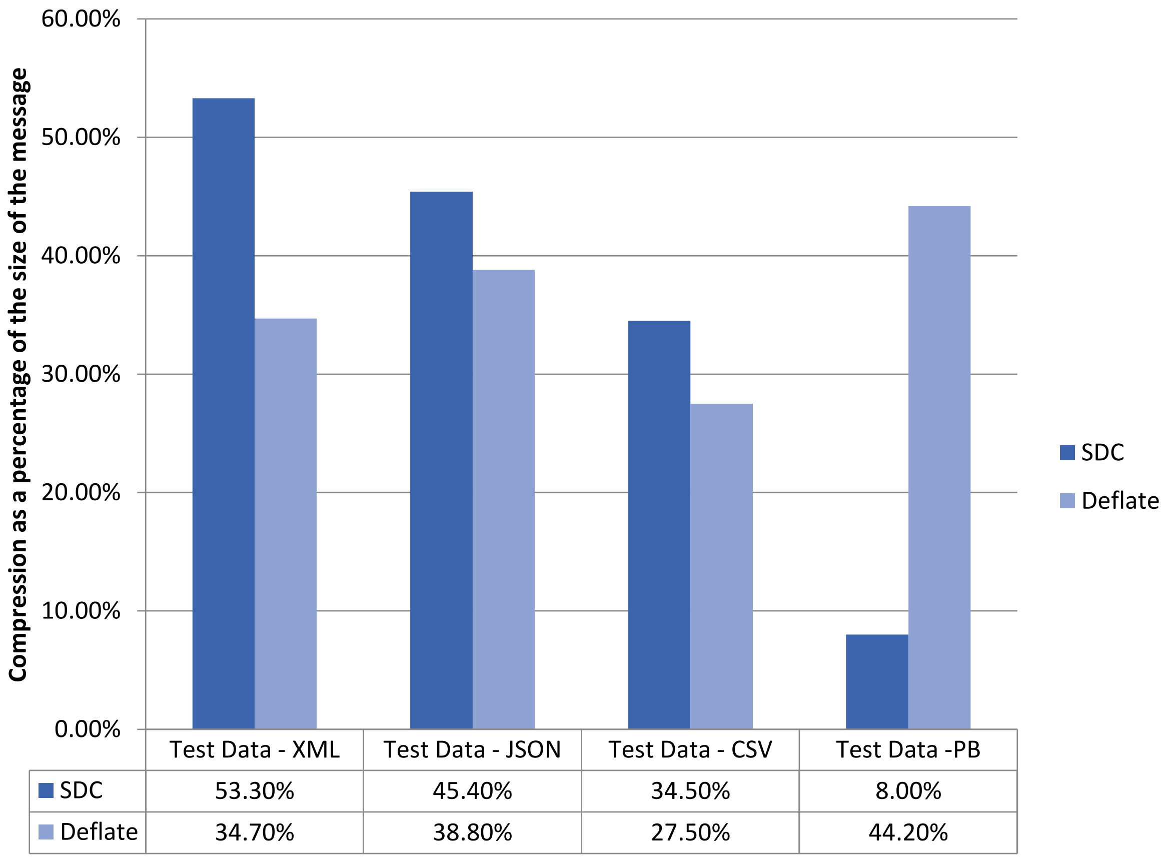
The calculation of the expiry time is determined by the adaptive algorithm. The adaptive algorithm is logic used to identify the potential of generating a new dictionary. If the compression is substantial, a new dictionary is proposed with a new expiry time and the old dictionary is discarded. However, if the savings are not up the mark, the existing dictionary's expiry time is extended. The adaptive algorithm requires messages to collected in a ring buffer at the sampling broker. The dictionary is generated is based on the stored notification.

The ring buffer size (B_{size}) is crucial element in computational complexity. If the size of the ring buffer is high, the comparison at the SDC is very high and leads to high computational requirement. Hence it one of the parameters that we need to setup through the adaptive algorithm. The second critical parameter is size of the dictionary ($SD_{Multiplier}$).

With out setting with these parameters initially, our runs with the proposed composure, we noticed, that the entries are disposed, if the maximum number of entries is reached; a 160 MB dictionary is used in the worst case. In pub / sub compression, the aim is to reduce the overall bandwidth between publishers and subscribers, including all related overheads. Our work concerns the tradeoff to enable dictionary-based compression in pub / sub and how the sweet spot can be determined and automated in terms of bandwidth reduction. However, we set our goals to find the computation power for a maximum compression ratio. This can be achieved by finding the point of saturation for bandwidth reduction, beyond which the bandwidth reduction is only insignificant. Yet 160 MB is way above the acceptable level. So, we planned to allocate 50 kB or smaller, depending on the content of the message. For testing the potential of SDC along with MQTT is implement without the adaptive algorithm, that is by fixing B_{size} and $SD_{Multiplier}$. Every Three Hundred notifications (**B_{size}**), a dictionary is made using the previous hundred notifications (denoted as **Rate**) and the size of dictionary is restricted to twice the notification size average. (**SD_2**).

5.4 Performance of SDC – First Trial

The results are as follows:



	Test Data - XML	Test Data - JSON	Test Data - CSV	Test Data -PB
■ SDC	53.30%	45.40%	34.50%	8.00%
■ Deflate	34.70%	38.80%	27.50%	44.20%

Figure 5.4.1 - Bandwidth reduction compared to deflate

The figure 5.4.1 is the performance analysis of the SDC without any adaptive algorithm or any moderator as such. Please also note that deflate is the default compression technology of MQTT. We are implementing SDC along with deflate. We will comparing the results with the traditional compression mythology deflate throughout.

Further different types of data have to be evaluated for the same datasets. In XML,JSON and CSV data types, the SDC provides better compression than deflate. However, in Google protobuf (PB), the compression is not much. – 8%. However overall there is a significant amount of compression that is deficiently achievable. 53.3% is achievable with SDC compression compared to just 34.7% of deflate when using XML.

In order to measure the commutation potential, we are measuring the time taken to compress and decompress the message. This parameter is a representation of how computation complexity.

Algorithm	**Compression time (millis)**	**Decompression time (millis)**	**Compression ratio**
Deflate	340	98	34.7%
Deflate + SDC	**2998**	**382**	**53.3%**

Table 5.4.2 – Processing Times compared to deflate

The table 5.4.2 is the results of the experiment. The processing times are indicated along with the compression ratios. The compression time is almost 3 seconds, if the initial parameters B_{size} or the $SD_{multiplier}$ is not fine-tuned. This is the reason for the need of adaptive algorithm. We need to fine tune these values to optimize the compression times and the reduce the size of the dictionaries.

SDC is faster on decompression where windowing complexity is eliminated ie, Huffman tree does not have to be computed on the fly. Minor notifications are less-compressible using deflate, as there is lesser recurrence within a message. In SDC, mutual tags are endorsed to the SD, hence the bandwidth savings are consequently developed. Overall thought in our evaluation, it makes sense to sample bigger dictionaries and exchange them less frequently.

5.5 Design of Adaptive Algorithm

The sampling broker produces the dictionaries and records the savings in bandwidth. The sampling broker performs an adaptive algorithm that either produces a new dictionary or extends the expiry time of the previous dictionary. The production of a new dictionary is because of the variations in the content of notifications. The adaptive algorithm selects the dictionary parameters, the size and the number of historical notifications to be sampled. The dictionary- size affects the overhead bandwidth and the compression performance. The length of historical messages affects the time a dictionary is sampled. An adaptive algorithm manages and finds the right balance between the performance and overhead bandwidth [5].

The basic requirement of our adaptive algorithm is to evaluate the amount of bandwidth that could be saved if a new dictionary is published. If the bandwidth savings are better than a certain brink, the existing dictionary is deleted and the new dictionary could be published, else the existing dictionary's expiry is extended. I.e., we will be using our adaptive algorithm to determine the $SD_{multiplier}$, the B_{size} , and the T_{exp}

The Bandwidth reduction could be calculated as

$$\boxed{\text{BR} = 1- \frac{\text{Size Compressed Notification}}{\text{Size of Uncompressed Notification}}} \qquad - \quad (1)$$

The Rate – R is resulting by dividing the total notification size between the first and last message in the buffer by the time span.

$$\boxed{R = \frac{\sum_{i=0}^{buffersize} |ni|}{t_{buffersize} - t_0}} \qquad - \quad (2)$$

The amortization time is the preset time span for the renewal of a dictionary.

$$\boxed{T_{amortize} = \frac{|\text{SD}|}{R * BR}} \qquad - \quad (3)$$

Where |SD| is the size of the dictionary, and R is (2)

Two parameters are taken into account when the dictionary is sampled: $SD_{multiplier}$ and its B_{size}. When prolonging the use of a dictionary, the $SD_{multiplier}$ and B_{size} values are tuned up and taken into consideration for the next evaluation. An increase in the size of dictionary or the sampling window size could lead to a greater reduction in bandwidth.

Every time the dictionary configuration is changed, the counter *adaptations* are increased. After a number of attempts with growing differences, B_{size} is not extended beyond a point because the computational costs are too high. This limitation should be selected based on the typical size of the notification. T_{exp} is the fixed expiry timestamp of a generated dictionary. An ideal dictionary should at least remunerate 10 times, as decided as practically.

$$\boxed{T_{exp} = T_{amortize} * adaptions^3 * 10} \qquad - \quad (4)$$

CHAPTER 6

RESULTS AND DISCUSSION

6.1 Parameter Selections

To design the adaptive algorithm, we have to recognize the connection between computational costs and bandwidth reduction. I.e., the two dictionary parameters from which the algorithm can select from: **the history length for dictionary sampling** and **the dictionary size.**

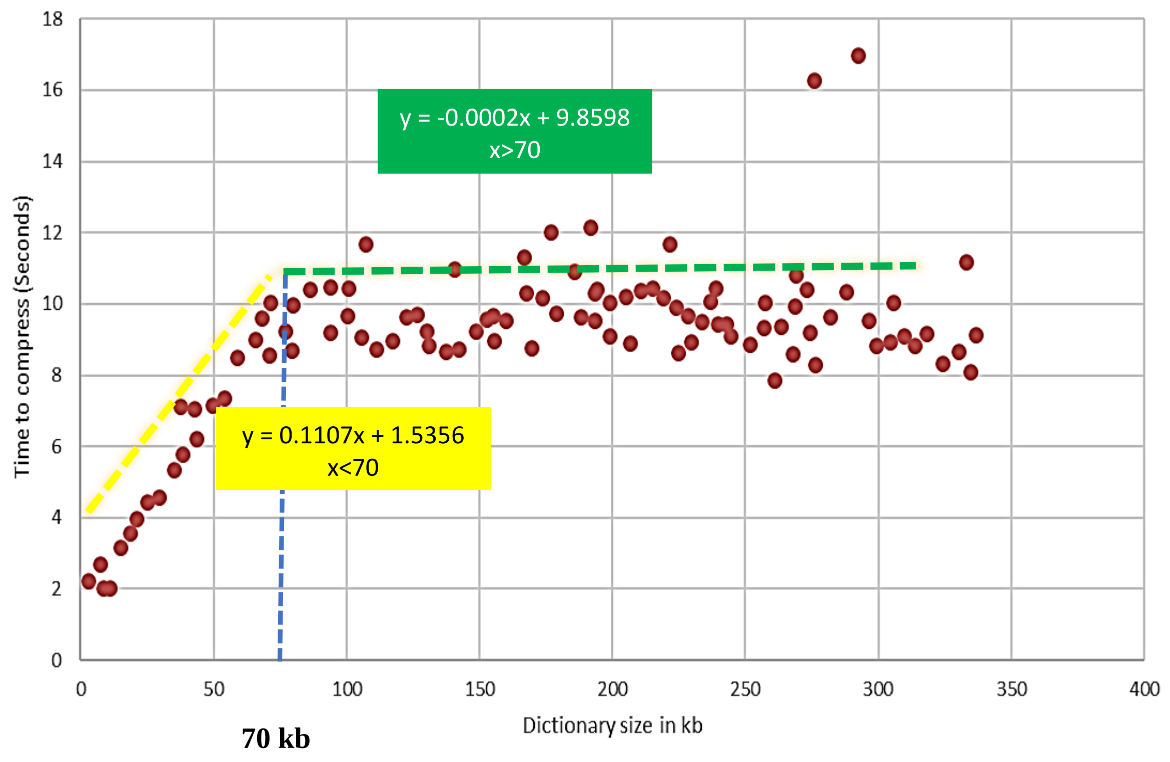


Figure 6.1.1 - Time to compress vs. dictionary size

Figure 6.1.1 shows the trend of the time needed to compress thousand messages vs the dictionary size for the twitter-us.json dataset. The relation increases linearly until to a dictionary size of 70 kilobytes, which matches to a size of twenty-one times the typical message size.

After this, the time to compress remains approximately constant. The time to compress continues the same because the extra entries in the dictionary are not used much, as shown in Figure 7.2 keeping 70 kb as the boundary point, the least-squares regression lines were plotted on both the graphs. The trend is mostly similar, but the thresholds depends of the content.

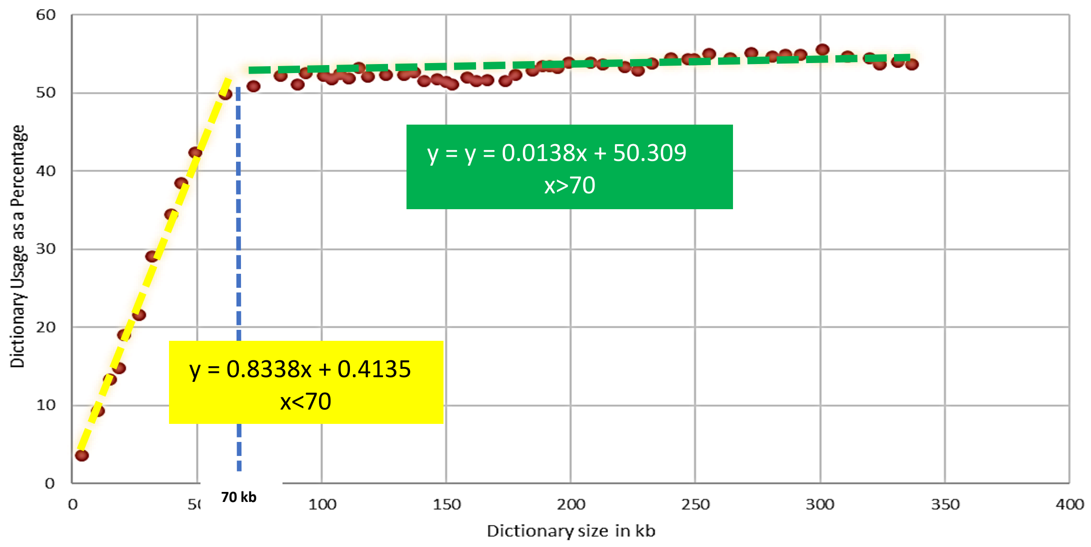


Figure 6.1.2 - Dictionary usage vs. Dictionary size

An ideal dictionary only contains entries that are actually used. In this trial, by keeping the dictionary size as 70 Kilobytes, we first make a dictionary of 5500 messages and then use it to compress the next 10,000 messages and record which parts of the dictionary are used and which parts of the dictionary actually reused.

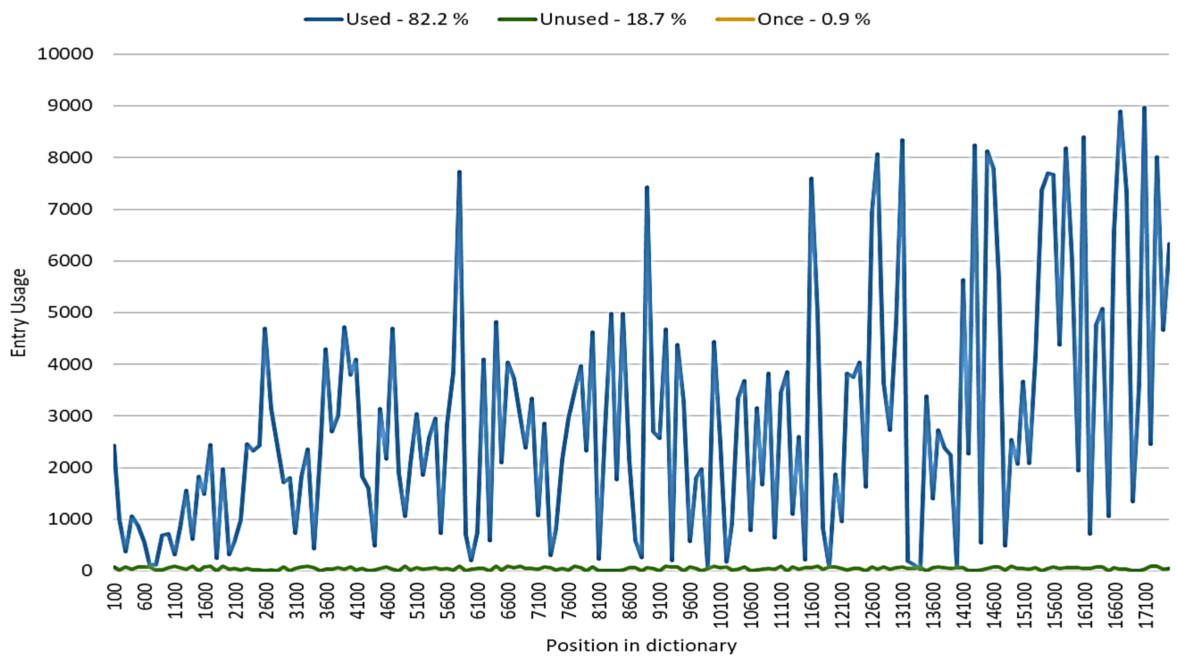


Figure 6.1.3 - Dictionary Usage

Figure 6.1.3 shows how often each part is used in the dictionary. Many dictionary parts are used more than 5000 times, which resembles to almost all other messages. Small parts are even used more than 9000 times, denoting that these parts are referenced in a single message many times. The substrings are arranged before they are fused and the most common substrings are towards the end of the dictionary.

Unused entries are the result of changes in the stream. Higher dictionary use can be achieved if the content is known in the first place, which is not practical in a stream.

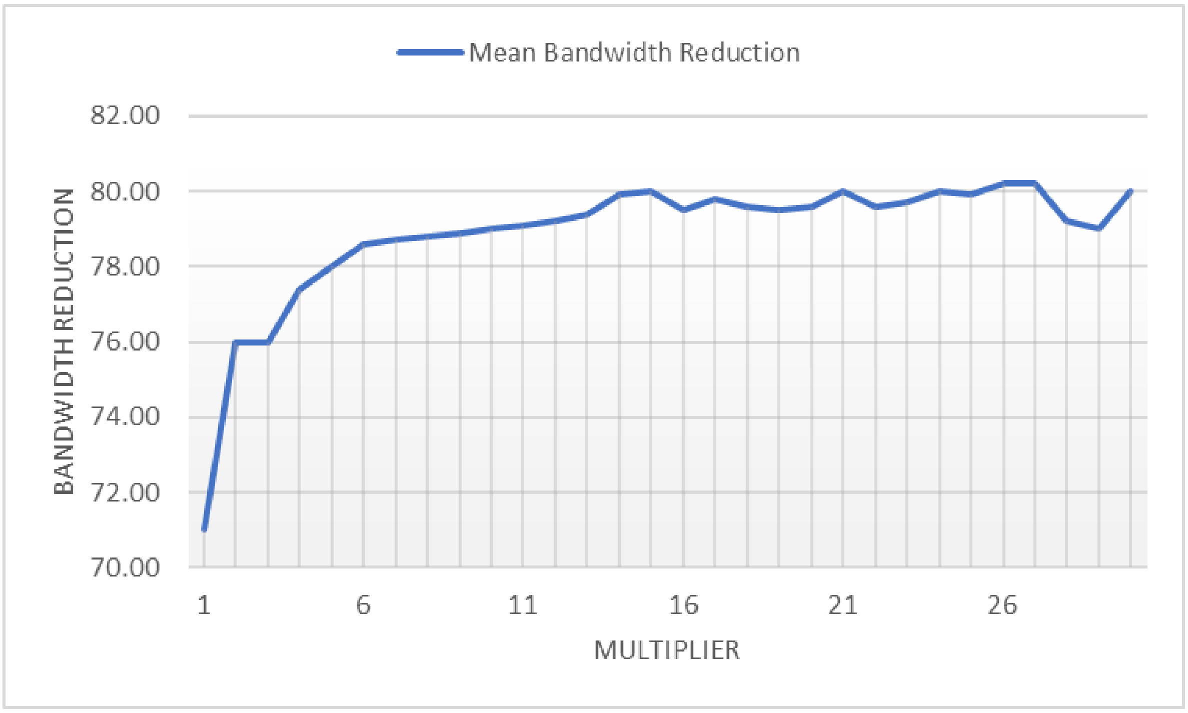


Figure 6.1.4 – Bandwidth reduction vs the dictionary size.

Figure 6.1.4 shows the average bandwidth reduction and dictionary size as an average message size multiplier. The general trend is that larger dictionaries lead to higher ratios of compression. This trend becomes steady because of the additional dictionary entries that are not often used. It is also important to sample these dictionaries over a large number of messages. After we have sampled the dictionary, we can evaluate different dictionary sizes with low additional computational costs.

In order for us to find the right balance, we take the dictionary, shorten it to a precise scale, create a compression model that comprises the SDC and assess the bandwidth reduction. As shown in Fig.7.4, the bandwidth reduction can be showed using a polynomial. Through this method, we can determine an ideal parameter for the dictionary size by using only several points and then matching a polynomial over several observations. **We chose 300 messages approximately as a number of messages to produce a dictionary.** The mean of all dataset's saturates at this threshold. These parameters will be used in the future experiments.

6.2 Performance of SDC + Adaptive Algorithm – Second Trail

When a dictionary is extended, the $SD_{multiplier}$ and B_{size} values are increased. The increase in the size of dictionary or the length of the sampling could result in a greater bandwidth drop. Every time the arrangement of the dictionary is altered, the counter adaptation is also increased. There is point where B_{size} could no longer be increased since the computational costs will be high. After we have sampled the dictionary, we can evaluate different dictionary sizes with low additional computational costs. For this purpose, we take the dictionary, truncate it to a specific size, create a compression model that includes the Huffman tables and evaluate the bandwidth reduction. The band width reduction can be modeled using a polynomial - to determine a good parameter for the dictionary size by observing only several points and then fitting a polynomial over several observations.

The ideal values for the parameters may depend on the content of messages. In our case it was

- SD multiplier - Size of Dictionary = 70 kb
- Bsize - Number of Messages sampled = 300

Algorithm	**Compression time (millis)**	**Decompression time (millis)**	**Compression ratio**
Deflate	340	98	34.7%
Deflate + SDC	**2998**	**382**	**53.08%**
Deflate + Adaptive Algorithm + SDC	**~912**	**~197**	**48.08%**

Table 6.2.1 – Processing Times compared to deflate

The table 6.2.1 compares the results of the implementation of SDC along with the adaptive algorithm with the traditional compression deflate. The highlight was that the compression time is brought from 2998 milliseconds to 912 milliseconds, i.e. ~3 seconds to less than 1 seconds. However, with the implementation we are able to reach 48% compression only whereas without any moderator such as adaptive algorithm we are able to reach 53.08%. However, the achievable compression is justifiable with relative to the compression and decompression times.

6.3 Data formats performance analysis

Figure 7.5 denotes the best outcome for each data-format with respect to the attained bandwidth reduction. The presented computational costs correspond to the best result in terms of bandwidth reduction. The unit of the computational cost is the total compression time in minutes taken by the processor. The outcomes for bandwidth reduction demonstrate the technique with the best bandwidth reductions of all variations in the case with one subscriber and one publisher to a range of topologies that have several more publishers than subscribers.

The computational costs are found in terms of processor usage time. For the computational cost of the publisher, we take the time required to compress each message in nanoseconds and equivalent of sum these times for all messages. Likewise, we obtain the same for subscribers.

Frequently from our experimentation, we discovered that point when the size of the dictionary is equal to the size of the message. Keeping the dictionary size same, the following processing times were observed as shown in Figure 6.3.1. Here the time taken to compress the messages are done through the publishes and decompression is done through subscribers. The compression time is generally larger and the decompression is shorter. This is perfect in practical terms. As usually the subscribers are less critical systems so less potential for processing and publishes on the other hand, are already equipped with significant processing power.

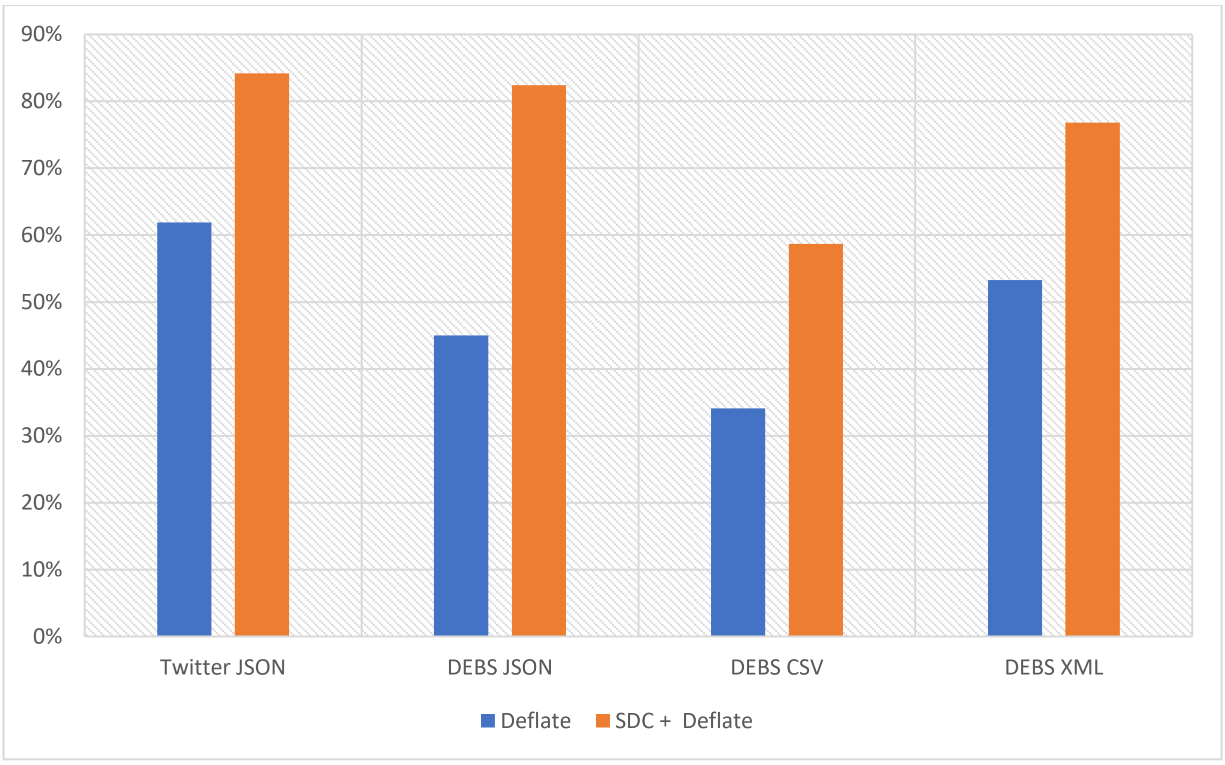


Figure 6.3.1 –Compression Ratios vs Various Data Formats

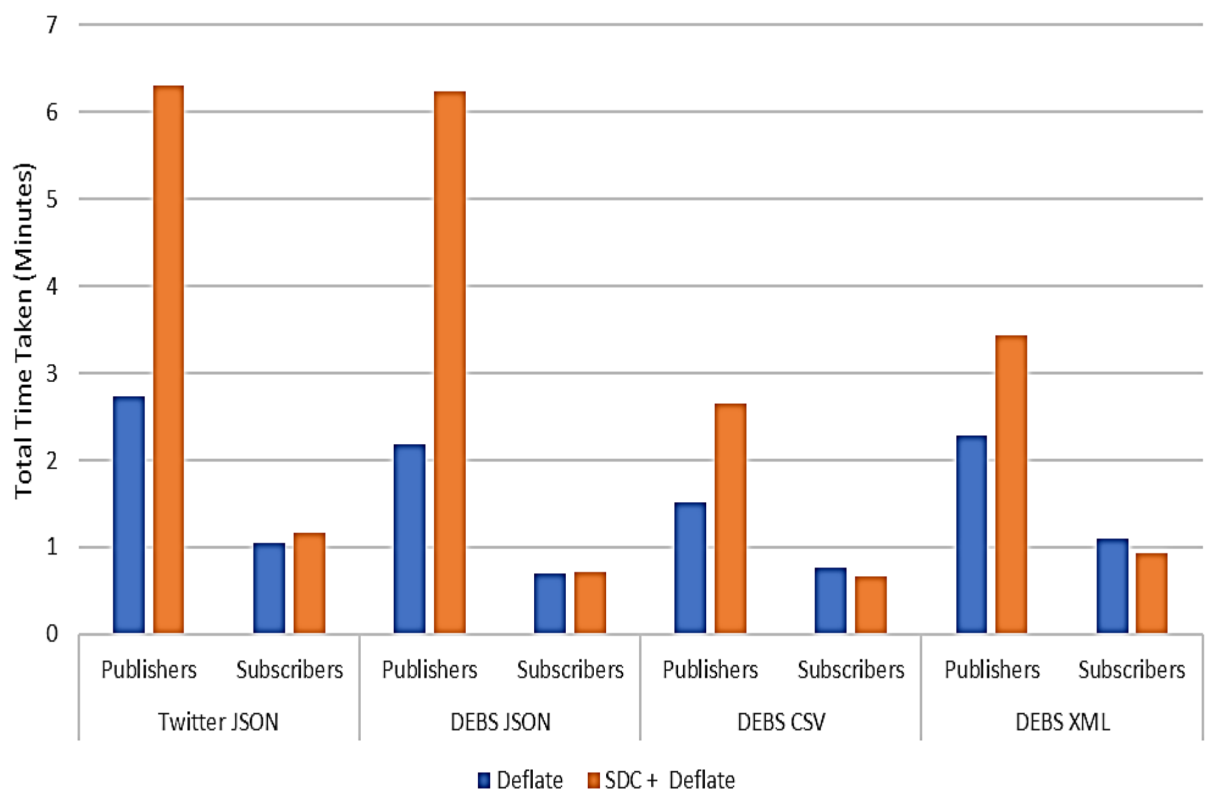


Figure 6.3.2 -The computational complexities for various data formats

Figure 6.3.2 compares the processing time for the compression methodology for various data formats. The lowest computational overhead for publishers is achieved with Deflate, as there is no additional processing occuring other than the defalte itself. However SDC + Defalte on the other hand plays a bigger role on the processing times or compression performance, for understandable reasons. It is also noted, that if the initial parameters are set incorrectly, the size of the dictionary continuously increases, when no further bandwidth is achieved and eventually ends up with a big dictionary, which harmfully affects the processing times overhead. The main reason for this behavior is that Adaptive ends up with dictionaries that are too large, which do not have a high impact on the BR. Furthermore, if the adaptive algorithm ends up with dictionaries that are too large, then the overhead is amplified in topologies that have many more publishers than subscribers.

Hence the initial parameters setting is extremely important and is the pillar stone of the SDC's performance. It is found that there is less variance in the bandwidth reduction between JSON or XML when SDC is used along deflate. However, we can't observe any similar patterns in the case of Deflate alone. The common strings blend is endorsed in the shared dictionary, which is comparable in all system discrepancies. The processing time for JSON format is analyzed in Figure 6.3.3.

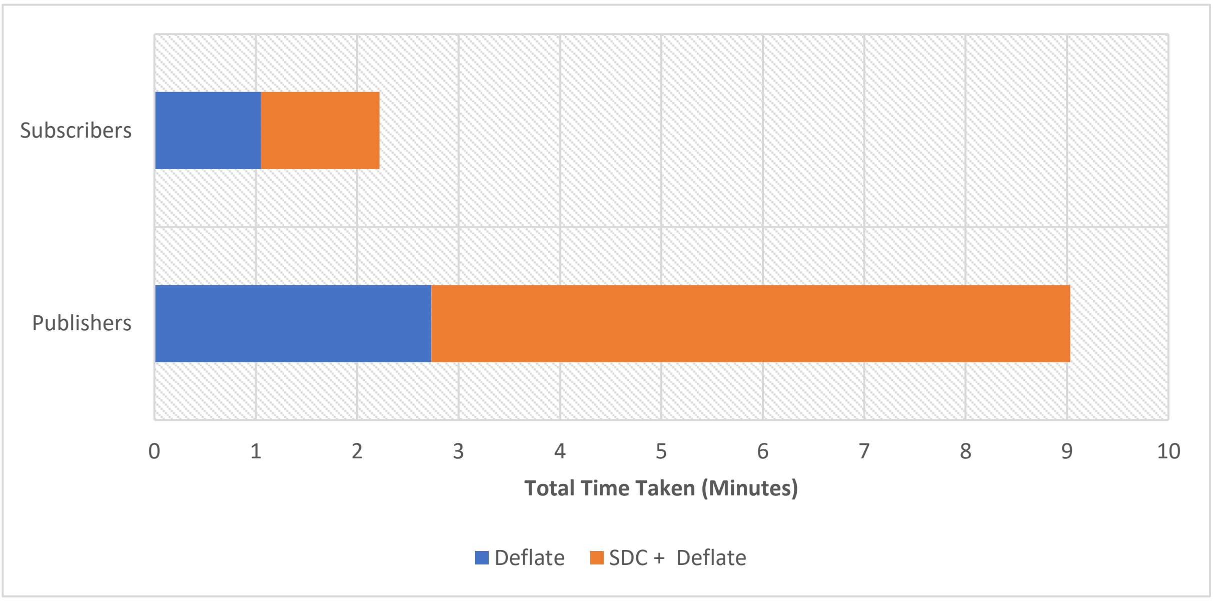


Figure 6.3.3 -Computational Complexity Analysis on JSON format.

However only if technologies like machine learning or artificial intelligence could be implemented in the place of adaptive algorithm, which could not only predict the bandwidhth reduction, it should work smoothly by giving the best bandwidth reduction for the used computational power.

6.4 Power Consumption

In order to compare the power consumption to transmit/reserve the data between the uncompressed data and the compressed data, we considered the characteristic power consumption for an ESP8266 Microcontroller. The particular microcontroller uses around 0.165 to 0.174 watts per transmission of 1024 bytes (1 Kilobytes) corresponding to the case scenario, where signal strength is from -65 dBm to -80 dBm.

Hence

Power Consumption per 1 Kilobytes could be considered as 0.174 watts on the worst case and the total power consumed comparison for a data transfer of 10240 bytes (10 Kilobytes) would be as represented by the below the table.

Algorithm	Compression ratio	Before Compression (Bytes)	After Compression (Bytes)	Power (watts)
No Compression	0%	10240	10240	1.749
Deflate	34.70%	10240	6686.72	1.1421
Deflate + SDC	53.08%	10240	4804.61	0.82063
Deflate + Adaptive Algorithm + SDC	48.08%	10240	5316.61	0.90808

Figure 6.4.1 -The computational complexities for various data formats

The power consumed by deflate, that is the traditional compression tools used by MQTT is 1.1421 watts, however with SDC integrated, we can bring it down to 0.82063 watts which is 0.3215 watts less than the default Deflate. However even though the power consumption is lower, the compression and decompression times can be burden, hence with the integrated adaptive algorithm, the power used is 0.90808 watts, which is 0.23402 watts lesser than the Deflate. Hence a **20.49 %** power reduction is achievable with the proposed MQTT with the SDC integrated.

CHAPTER 7

CONCLUSION

The analysis of the actual use of the dictionary was comprehensive. However, it is best advised to have SDC as an option as, there are some applicaiotns where computatinal factors are a lower consern compared to the bandwifht reduction, and some applications its vise versa. So best the SDC is layed in as an option with in MQTT.

Example, a single IoT device with multiple core processor and is part of a mesh network of similar devices might need to use the available bandwidth more efficiently, compared to a single device of lower processing power in a consolidated network.

In our research we tried to implement SDC as a third party intergration. However if its part of MQTT and manged by MQTT, it would be more convient and efficent. Since SDC is implemetned per topic, it would make sense, if it is inbuilt within the topic manager of MQTT. SDC is particularly useful in the field of Internet of Things. The adaptive algorithm greatly depend on the dataset. If the dataset has less freuquent words, the compression is not much. However in an IoT enviroment, the frequency of repeating keywords is significantly high. As most events and notifications include metadata like place, units, timestamp, ids then numerous values which are understandings from the site (e.g., temperature, humidity levels, etc.)

In numerous successive notifications this metadata will be repetitive and could be endorsed to the dictionary. SDC can therefore pay off for the inefficiencies of sensory information redundancies. The most surprising finding was that the use of the CPU for compression with large dictionaries depends on the contents. This is because future entries tend to be used less. Greatly helped us fine-tune and find the balance between the bandwidth and compression.

We believe that the main hurdle for adopting SDC in pub/sub is the manual parameterization of dictionary compression. Our approach solves this problem using a combination of fitting polynomials to determine the parameters for the dictionary automatically. Therefore, we denote a methodology to find the sweet-spot between computational resources vs. bandwidth. In our practice, once the polynomial is determined by the adaptive algorithm and the initial parameters are set, the system could deliver the maximum compression with the minimum delay in the processing load.

CHAPTER 8

FUTURE WORK

Currently, we designed and implemented cloud- based SDC-MQTT using a simplistic system model. A more comprehensive system prototype including sizes and bandwidth boundaries on all relations within the system including the clients would be able to improve for many more situations. Even though the adaptive algorithm provides a best compression with minimal computation times, the setup requires lots of manual settings to fixed initially. If the initial values were not fixed the dictionary size goes beyond acceptable size for distribution. Hence a higher limit is required. The calculation of this higher limit could be very difficult. Since through trail and error would cause a heavy burden on the processing unit in the initial runs. We will need to formulate the higher limit and thresholds for the creations of dictionaries by the adaptive algorithm.

Further, the concept could be now be implemented in the MQTT to the protocol level,since MQTT has topic mangemanet and user/device subscritntins mangement builtin. And hence the payload or perforamce could be brought to a fully optimised state. Futher libraries for the IoT devices could have the options for SDC builtin along with MQTT. Also the MQTT should have built in option to automaticaly enable the SDC option, if it depends frequncy of keywords to higher.

AI or Artifical intelligence could play a very interesting role in setting the appropriate setting for the SDC to preform efficently. This could greatly eleminate the eroors in the judgemnt of the dictionary generation and transmission. Construcuing the ploynomial by the adaptive algorythm is highly computationally complex. AI on the other had could be understand and predict the message patterns and be able to properly manage and disemble the dictionaries.

REFERENCES

[1] Subrahmanyam, Rong Xiang, Gerald Kallas, Neeraj Krishna, Stefan Fassmann, Martin Keen, Dave Locke, "*Building Smarter Planet Solutions with MQTT and IBM WebSphere MQ Telemetry*", pp 4-45, An IBM Redbooks publication, 2012.

[2] Louis Columbus, "*Roundup Of Internet Of Things Forecasts And Market Estimates*", 2016. [Online] Available: https://www.forbes.com/sites/louiscolumbus/2016/11/27/roundup-of-internet-of-things-forecasts-and-market-estimates-2016/#1ea80d44292d. [Accessed: Nov. 12, 2018].

[3] Lucy Zhang , "Building Facebook Messenger*", facebook.com*, para 5, 2011. [Online] Available: https://www.facebook.com/notes/10150259350998920. [Accessed May 1, 2018.]

[4] L. Peter Deutsch, *"DEFLATE Compressed Data Format Specification version 1.3"* ,1996. pp 6-9.

[5] Christoph Doblander, Tanuj Ghinaiya, Kaiwen Zhang, Hans-Arno Jacobsen, *"Shared Dictionary Compression in Publish/Subscribe Systems",* 2016, Presented at the 10th ACM International Conference

[6] H. White. "*Printed English compression by dictionary encoding*". Proceedings of the IEEE, 55(3):390–396, March 1967

[7] Christoph Doblander, Kaiwen Zhang, Hans-Arno Jacobsen, “Publish/Subscribe for Mobile Applications using Shared Dictionary Compression”, presented at the IEEE 36th International Conference on Distributed Computing Systems (ICDCS), 2016.

[8] Z. Jerzak and H. Ziekow. *“The DEBS 2015 Grand Challenge”* Ian Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems, DEBS ’15, pp 266–268, New York, NY, USA,2015.

[9] Sam Lucero, “IHS Technology, IoT platforms: enabling the Internet of Things”, *technology.ihs.com* , Page 5, March 2016. [Online] Available: https://cdn.ihs.com/www/pdf/enabling-IOT.pdf [Accessed May 2 2018]

[10] T. Fujita, Y. Goto, A. Koike, “M2M architecture trends and technical issues”, The Journal of IEICE, Vol.96, pp.305 ー 312, 2013.

[11] Tetsuya Yokotani, Yuya SasakiP “Comparison with HTTP and MQTT on required network resources for IoT”, published at International Conference on Control, Electronics, Renewable Energy and Communications (ICCEREC), 2016.