Design, Implementation and Analysis of a Twitter-based Social IoT Network

Danielle Sheridan*, Anderson Augusto Simiscuka† and Gabriel-Miro Muntean‡
School of Electronic Engineering, Dublin City University, Dublin
Email: *danielle.sheridan37@mail.dcu.ie, †anderson.simiscuka2@mail.dcu.ie, ‡gabriel.muntean@dcu.ie

***Abstract*—The Internet of Things (IoT) is a fast-growing phenomenon that interconnects machines to other machines to communicate and share data. There is a new found interest to connect every day objects to the internet, thus the emerging smart homes and smart cities era, in which household objects and devices throughout the city, such as traffic lights, are online. Similarly, social media connects people to people, while IoT mirrors these connections with devices. This paper introduces a solution for interconnecting users and devices through the use of Twitter, one of the most used social networks. Twitter provides an extensive endpoint API and acts a suitable platform for human-to-machine communication. The solution presented allows human-to-machine connection through IoT protocols such as REST and MQTT, and integrated IoT devices to Twitter and a cloud-based server. Results analysis indicate the ideal settings and protocols for this social IoT network.**

***Index Terms*—IoT, social IoT, social media, cloud.**

I. Introduction

The Internet of Things (IoT) enables a plethora of devices to communicate with each other and share data through communication protocols such as WiFi, Bluetooth, IEEE 802.15.4, Z-wave, Radio Frequency Identification (RFID), Near-field communication (NFC) and SigFox [1]. Machine-to-machine (M2M) features incorporated into IoT allow machines to make decisions based on data. Several solutions aim to improve service delivery and user experience in IoT, which can be challenging due to the heterogeneity of the network and the variety of devices [2], [3], [4], [5]. A new paradigm known as the Social Internet of Things (SIoT) has emerged and it incorporates social networking concepts into IoT. These concepts include a structure with guaranteed route navigation, which will ensure scalability and effective discovery of connected objects and trust between connected objects and users that are 'friends' within the SIoT platform [6].

Based on these concepts, this paper introduces a novel social IoT network which uses the Twitter platform to allow users to control IoT devices remotely. A detailed presentation of the principle and testbed (illustrated in Fig. 1), is included, as well as the associated algorithms and protocols. Noteworthy is that a cloud-based server is integrated into the solution.

This paper is organised as follows. In section II, related works are presented and section III introduces the technical background of the solution. Section IV details the design and implementation of the testbed, and section V presents testing and results. Section VI finalises the paper indicating the conclusions and future work directions.

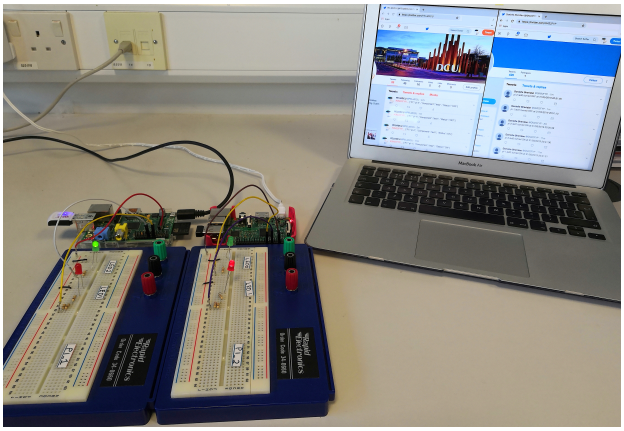

Fig. 1. Twitter-based social IoT network testbed

II. Related Works

A. IoT Protocols

IoT devices use M2M communications to interact with each other and it is estimated that by 2022, there will be 14.6 billion M2M connections [7]. Due to this increase in network traffic, it is critical for IoT protocols to meet requirements that will lessen the load for the network. Examples of protocols being employed in IoT are the Hypertext Transfer Protocol (HTTP) and the Message Queuing Telemetry Transport (MQTT).

REST (Representational State Transfer) is an architectural style over HTTP protocol that uses a request/response model. It is stateless, which means that each request from a client to the server must contain all data necessary for the server to process the request, as the connection is always closed after the request. One of the advantages of REST is the fact it uses methods from the HTTP library such as GET, POST and DELETE requests and can be implemented in most programming languages and embedded devices as most operating systems include this library [8]. However, constrained devices with limited memory and power do not need all of these methods, and other HTTP features prove useless in an IoT environment, including design orientation. In [9], the author demonstrated how RESTful interfaces can be used to create a complex IoT structure, with several homogeneous devices. However, due to the resources required to perform HTTP requests, receive HTTP responses and, as pointed out in [10], the high usage of network resources to establish a connection

each time there is any transfer of data, REST may not be the most ideal protocol for lightweight M2M communications.

MQTT is an extremely lightweight messaging protocol that uses TCP as the transport layer in the TCP/IP model [11]. MQTT uses a publish/subscribe messaging model. In this model, there are subscribers, publishers and a broker. The subscriber receives the data that is sent by the publisher if it is subscribed to the data/topic being published, and the broker remains centred within this message exchange to manage the events. A subscriber can be subscribed to many topics at the one time [12]. MQTT was created to achieve three main goals: reliable degree of delivery assurance of messages, lightweight design to minimise network bandwidth and easy to implement on embedded devices with low resource requirements [13]. MQTT data is transferred as byte arrays unlike REST, which needs to define content type. There are few messages types included with MQTT such as, CONNECT, PUBLISH, SUBSCRIBE, UNSUBSCRIBE and DISCONNECT. Regarding message size, MQTT uses short headers, two bytes is the minimum packet size for a message, one byte for control field and one byte for packet length field, which is all that is needed in a DISCONNECT message.

B. IoT and Social Media

In [14], authors stated that the majority of young people receive information on world events through the use of social media platforms such as Twitter, Instagram, YouTube, Snapchat and WhatsApp, to name a few. Another application of social media analytics was presented in [15], aiming to use information concerning the emotional states of users through content they shared. The real time sharing of these updates could create faster emergency responses or improve crime control. Social media allows users to update a status which can be viewed by the user's friends. Similarly, with the concept SIoT, an object can update its status and location, and be seen by the friends of this device, which may be people or other objects. An implementation of an SIoT network in a smart home environment using Facebook was introduced in [16]. This study presented the benefits of using social media features in IoT to monitor and control connected devices in a home environment, taking advantage of the Facebook security system to keep data private and visible to a chosen audience. Authors, however, did not perform any network analysis.

Twitter is one of the world's biggest social media platforms, with around 500 million tweets sent each day by its about 326 million users [17]. It has an accessible API that can only be accessed with an approved developers account. Twitter has a character limit when creating tweets. This ensures that packets being received through the API will not be too large and slow down the transmission time. Twitter also prevents duplicated posts, which display an error message on the Twitter UI if attempted, in order to prevent spamming, also useful for the solution presented in this paper.

III. Technical Background

The solution design consists of mechanisms that allow users to tweet actions to be performed by IoT devices and receive tweets with notifications of status changes. This section presents the technical details of the components that are part of the Twitter-IoT solution.

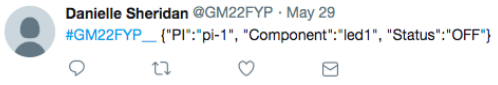

Fig. 2. Tweet format

Fig. 3. Tweet response from device

A. Communication Structure

Users can interact with devices by writing tweets from any twitter account, as long as not blacklisted in the IoT gateway. Tweets need to follow a certain structure that can be translated into device actions containing relevant information such as, the device in which the user wants to access a component, the target component on that device and the action desired (e.g. turn on/off).

The use of the hash sign (#) at the start of a particular word or phrase is an essential part of 'Tweeting' known as 'hashtagging'. Hashtags are used to identify tweets that belong to a certain subject or category. The hashtag used in the testbed was "#GM22FYP__", and it allows identification of messages sent to the smart IoT gateway of the platform.

Tweets with the correct hashtag and user action are converted into a JSON map, and device, component and status are the keys of the JSON map. The value for each key would relate to the chosen device, component or status update. For example, a user that wishes to turn on an LED on a device would tweet the following: #GM22FYP__ {"PI": "pi_1", "Component": "LED_1", "Status": "ON"}. This tweet information is then converted into a JSON map. An example of this structure being tweeted can be seen in Fig. 2, with a device response presented in Fig. 3.

Twitter processes around eight thousand tweets sent every second. In order to obtain the desired tweets, they must be constantly searched, something possible thanks to the specified hashtag previously mentioned. Once the tweet hashtag matches the criteria the tweet data can be processed in order to complete the user operation. Tweet content is analysed so the expected map keys can be extracted. Once this is performed, the data is transmitted by a communications protocol to a cloud-based server. If a map or the appropriate keys are not found, the data will not be sent. Twitter uses the REST protocol for communications.

B. Protocols, Platform and Devices

Two communications protocols were implemented in the solution for network analysis: MQTT and REST, which were described in section 2.

The solution uses Adafruit IO, which is an online cloud platform designed to store IoT data. Adafruit keeps data private by default and supports data protocols such as REST and MQTT. The stored data can be accessed by using a username and a security key [18]. There are feeds in Adafruit which hold the metadata sent to the server. Each feed represents a value of a component, which in the testbed represent values for each component on each device. The data can be accessed and updated by a device by supplying the feed name.

Testbed also includes Raspberry Pis [19], which are systems-on-a-chip widely used in IoT solutions. Several LEDs and breadboards are also part of the testbed.

IV. Design and Implementation

The solution design is illustrated in Fig. 4. Design and implementation details are presented in this section. The implementation and the testbed allow for testing of the proposed architecture and solution performance analysis in a real scenario.

A. Testbed Setup

One of the Raspberry Pis acts as a gateway to bridge Twitter to Adafruit, running specific scripts for Twitter and Adafruit connection. The remaining Raspberry Pis act as IoT devices with components (e.g. LEDs) that can be controlled through Twitter, running scripts to read Adafruit's feeds and act accordingly. The gateway Raspberry Pi can also act as an IoT device and have components attached to it.

The gateway Raspberry Pi connects to Twitter, search for relevant tweets, analyse them, connects to Adafruit and send the appropriate data. The gateway also continuously check for changes in the Adafruit feeds informing users about new status updates generated in the IoT devices.

Each Raspberry Pi acting as an IoT device contains two LEDs, red and green, which can be 'ON' and 'OFF'. These Raspberry Pis constantly check their corresponding feed on Adafruit and change act appropriately responding to the latest user action.

In order to determine resistor values used for each LED, voltage supplied is considered, which in the testbed is 5V. Voltage-drop across the LED is taken away from the supplied voltage, and for the red LED this value is 3.2V. The desired current that will flow through the LEDs must be about 25mA. Using Ohm's Law which states Voltage=Current x Resistance, the resistance value needed for the red LED is about 128 Ohms, which will be rounded to 150 Ohms. The green LED which has a voltage drop of about 3.3V so it needs a resistance value of around 68 Ohms, therefore, a 100 Ohm resistor will be used with the green LED.

Raspberry Pis' GPIO pins supply the power to the LEDs through resistors and a breadboard. The required pins are set as outputs. Values considered in the Raspberry Pis in order to update LEDs are 1 for on and 0 for off.

B. Twitter Streams Filtering

Twitter's API can be accessed with a developer account, in which the user is given OAuth authorisation to send requests

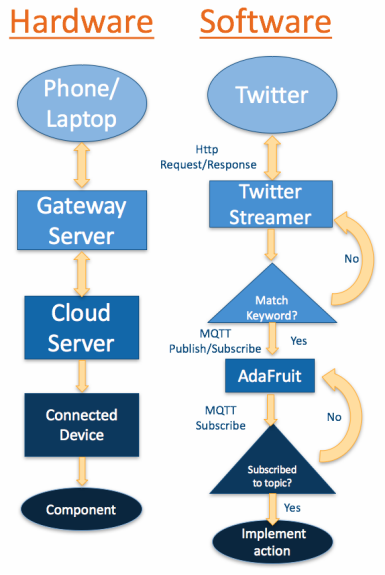

Fig. 4. Solution design

to the API. Requests are sent over a TCP connection using a RESTful connection. The Twitter API contains several relevant endpoints that can be used for tasks such as tweet filtering in real time. The parameters for this endpoint are optional, and an example of its use is to track a specific keyword in a specific language. By selecting English in this parameter, it avoids misinterpretation of other languages. The programming language used in the scripts running in the gateway Raspberry Pi is Python, and a Python library called 'Twython' [20] is employed with the tasks of connecting to Twitter and setting up a stream (TwitterStreamer()). In order to avoid delays in searching for relevant tweets, the TwitterStreamer() is threaded and the tweets are put into a Queue() to be handled appropriately, as presented in Algorithm 1.

During the stream filtering phase, it is also possible to add security measures. The hashtag filtering is the first one, as only users who know the hashtag can control the objects. Additionally, filtering by username is also possible, as Twython receives the username related to all tweets. The gateway Raspberry Pi contains a list of 'whitelisted' users whose tweets must be processed, increasing network security. Twitter already performs a reliable authentication process during login.

C. Tweet Processing

Tweet content validation is performed before data is sent to Adafruit, so it receives data in the format it expects, as demonstrated in Algorithm 2. Twitter API sends responses in JSON format, encoding instructions in the 'text' field of the JSON response. An example of a JSON response when updating a status is presented in Fig. 5. The format used in the text field is a map and so the symbol '{' is searched first in the text and split using the function split(), in order to extract the contents of the map removing the unnecessary initial curly bracket. The string is split into two halves and

Fig. 5. Twitter JSON response

second half also needs to be split() in order to remove the closing curly bracket '}'. The first half of the new content generated by split() contains the string with the data from the tweet. If the text extracted from the tweet does not contain the curly brackets in the desired way, an exception thrown. If the split() function succeeds in removing both curly brackets, the function json.loads() converts the map into JSON format, generating an exception if the conversion is unsuccessful. The keys of the JSON map are then checked to ensure that all keys where inserted by the Twitter user (i.e. device, component and status). If the keys are present, then the data is sent to Adafruit.

D. Protocols Implementation

The gateway Raspberry Pi sends (i.e. maps of information from tweets into feeds) and receives (i.e. confirmation of insertion of data into feeds) data to and from Adafruit. Raspberry Pis used with IoT components read feeds on Adafruit. The communications of Raspberry Pis with Adafruit was implemented with the library Adafruit_IO_Python [21], and both REST and MQTT were employed for testing, as both protocols' implementations are available in the library.

The gateway Raspberry Pi sends the data received from Twitter to Adafruit, in the appropriate format. Once the client (i.e. gateway Raspberry Pi) is initialised and connected to Adafruit, there are functions available to be used according to the actions of the client, and shown in Algorithm 3. For instance, when the client connects, the function "connect()" is responsible for subscribing the client to the appropriate feed. There are other functions for disconnections and updates from Adafruit when topics are changed and the new status is tweeted to inform the relevant audience. The publishData() function sends data to Adafruit. The MQTT client package contains the function loop_background(), which creates a new thread that continuously listen for changes in feeds without disconnections.

The connected devices that are being controlled through Twitter, listen continuously for updates in the feed that they are subscribed to. For instance, in the testbed, the first Raspberry Pi will subscribe the feeds where "pi-1" is featured, as demonstrated in Fig. 6, which shows the feed "pi-1" and two

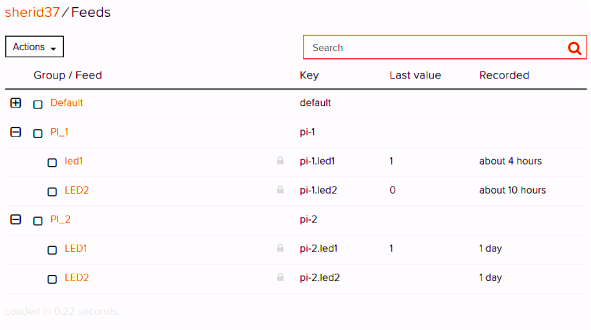

Fig. 6. Adafruit feeds page

Algorithm 1 Twitter Stream Filtering

```
Track_term = '#GM22FYP__'
Class TwitterStreamer(TwythonStreamer):
        def __init__(Oauth_tokens, Oath_secret, queue):
                super(TwitterStreamer).__init__(
                        ↪ Oauth_tokens, Oath_secret, queue)
        def on_success(data):
                if 'text' in data: put_data_in_queue()
        def on_error(data): print data
def receive_tweets(queue):
        TwitterStreamer(Oauth_tokens, Oath_secret, queue)
        TwitterStreamer.statuses.filter(track=Track_term,
            ↪ language=English)
def handle_tweets(tweet_queue):
        while true:
                get_from_queue()
                get_map_from_data()
                if (all_keys_present()): send_data()
def main:
        Thread(target=receive_tweets,args=[Queue()]).start()
```

subgroups, "led1", "led2". If a value of 1 or 0 is presented in the feed "pi-1.led1", the LED will reflect this, 1 represents on and 0 represents off.

V. Results and Discussion

For testing purposes, the time between sending of the tweets and the acknowledgment were recorded to determine the average, maximum and minimum time in which the user would perceive instructions being implemented.

Initial tests consisted of several tweets being sent at a low frequency to ensure they were processed with no packet loss. The timestamps were computed to determine maximum, minimum and mean time of transactions. The frequency of tweets being sent was then increased in order to determine latency and packet loss at different rates. Timestamps were computed just as soon as a tweet was received from Twitter, before data being sent to Adafruit, and then another timestamp was recorded when data was received from Adafruit. Frequencies analysed were every 5 seconds, 2 seconds, 500 milliseconds and 100 milliseconds.

Algorithm 2 Tweet Processing

```
def analyse_tweet(tweet):
        try:
                get_map(tweet)
        catch IndexError:
                print(error)
        try:
                convert_to_json(map)
        catch ValueError:
                print(error)
        if(keys "PI", "Component", "Status" are present)
                send_data()
def get_map(tweet)
        return(data.split('{')[1].split('}')[0])
def convert_to_json(map):
        return json.loads(map)
```

Algorithm 3 Cloud Communications

```
def connected():
        subscribe(feeds)
def disconnected():
        system.exit()
def message(feed, payload): //only for Gateway
        tweet_change_in_component(feed, payload)
def message(feed, payload): //only for Connected Device
        change_status(payload)
def send_message_to_adafruit(tweet): //only for Gateway
        if(Component==led1)
                if(status=="ON")
                        send_to_Adafruit(component, {
                            ↪ value: 1}, feed)
                elif(status=="OFF")
                        send_toAdafruit(component, {
                            ↪ value: 0}, feed)
        if(Component==led2)
            if(status=="ON")
                        send_to_Adafruit(component, {
                            ↪ value: 1}, feed)
                elif(status=="OFF")
                        send_toAdafruit(component, {
                            ↪ value: 0}, feed)
def main:
        createClient(Adafruit_username, Adafruit_key)
if (connected_to_client)
                loop_background()
else
                print(error)
```

The graphs presented in Figs. 7, 8 and 9 show average, minimum, and maximum latency, respectively, considering the time tweets were received by the Twitter streamer before publishing it to Adafruit, plus the time needed for the data to be sent to Adafruit and be updated in the feed. These tests considered REST for Twitter and MQTT for Adafruit connections.

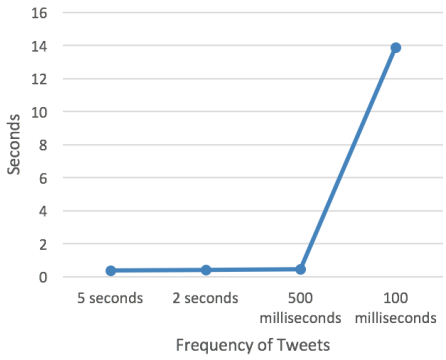

Fig. 7. Average total latency

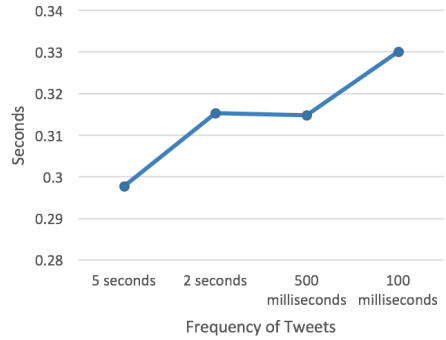

Fig. 8. Minimum total latency

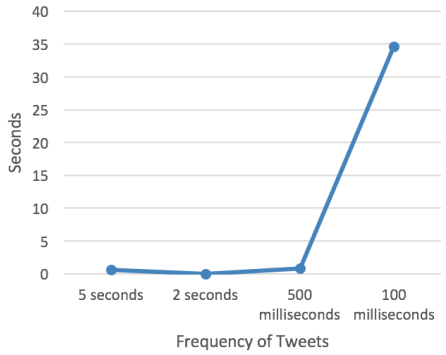

Fig. 9. Maximum total latency

Each tweet sent was processed by the scripts, demonstrating that there was no packet loss in these scenarios. A limitation of using Twitter's API for real time data only allows 1% of the tweets containing the search term being retrieved, which did not impact testing due to the hashtag used being very specific for the testbed and not having widespread use.

From the results from the graphs in Figs. 7, 8 and 9, it is possible to notice that tweets are processed with varying latency. For instance, when a few hundred of tweets were sent every 5 seconds, the minimum latency observed until the

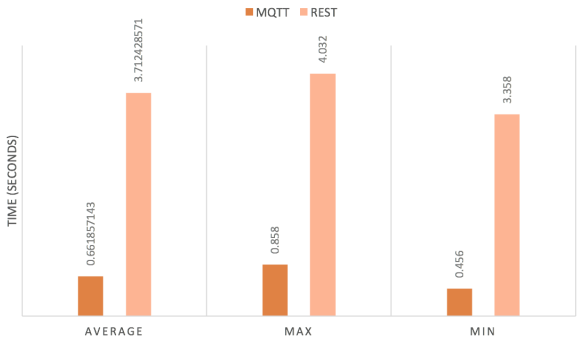

Fig. 10. Delay in connecting to four feeds in Adafruit (MQTT vs. REST)

feed in Adafruit was updated was less than 0.3s whereas the maximum latency was around 0.6s. If the tweet frequency is very high, for instance, one tweet every 100ms, the latency varied from 0.33s (min.) to 34.6s (max.). The results show that the latency is low if tweets are sent every 500ms, 2s and 5s, but can be very high when tweets are sent every 100ms, due to high processing in the gateway Raspberry Pi and delays in the connections with Twitter and Adafruit. Therefore, a scenario with tweet frequency of 100ms is not recommended, as the maximum latency observed was 43 times higher than the maximum latency observed in the 500ms frequency (34.6s versus 804ms).

For protocol comparisons, the gateway Raspberry Pi communicated with Adafruit using REST and MQTT. Delay was measured for both protocols, as seen in Fig. 10, for the connection with the four feeds available on Adafruit. MQTT performed much better than REST, as REST creates new connections every time it needs to transmit data. On average, in order to connect to the four feeds on Adafruit, the gateway Raspberry Pi experienced 82% less delay on communications when using MQTT instead of REST.

VI. Conclusions and Future Work

This paper presented the design, implementation and analysis of a Twitter-based IoT Network. MQTT and REST protocols were examined and tests demonstrated that MQTT performs with much less delay when connecting to the cloud server. Raspberry Pis were controlled by tweets turning on and off LEDs and that allowed further analysis on the best scenarios for the solution, such as tweet frequency which perform well up to 500ms. Future work includes support to different protocols, such as CoAP, and other types of sensors and IoT devices. Multiple developers accounts could also be tested, allowing multiple gateways with diverse IoT components. The feasibility of other social networks, such as Facebook, could also be investigated.

Acknowledgement

This work was supported by the Irish Research Council and Dublin City University, grant number EPSPG/2015/178, and in part by European Union's Horizon 2020 Research and Innovation programme under Grant Agreement no. 688503 for NEWTON project (http://newtonproject.eu).

References

[1] S. Al-Sarawi, M. Anbar, K. Alieyan, and M. Alzubaidi, "Internet of Things (IoT) communication protocols: Review," in *ICIT 2017 - 8th International Conference on Information Technology, Proceedings*, 2017, pp. 685–690.

[2] A. A. Simiscuka and G. M. Muntean, "Age of Information as a QoS Metric in a Relay-Based IoT Mobility Solution," in *Proc. of the 14th International Wireless Communications and Mobile Computing Conference (IWCMC)*, 2018, pp. 868–873.

[3] A. A. Simiscuka and G.-M. Muntean, "A Relay and Mobility Scheme for QoS Improvement in IoT Communications," in *Proc. of the IEEE International Conference on Communications Workshops (ICC Workshops)*, 2018, pp. 1–6.

[4] A. A. Simiscuka, T. M. Markande, and G.-M. Muntean, "Real-Virtual World Device Synchronisation in a Cloud-enabled Social Virtual Reality IoT Network," *IEEE Access*, vol. 7, pp. 1–12, 2019.

[5] A. A. Simiscuka and G.-M. Muntean, "Synchronisation between Real and Virtual-World Devices in a VR-IoT Environment," in *Proc. of the IEEE International Symposium on Broadband Multimedia Systems*, 2018, pp. 1–6.

[6] L. Atzori, A. Iera, G. Morabito, and M. Nitti, "The social internet of things (SIoT) - When social networks meet the internet of things: Concept, architecture and network characterization," *Computer Networks*, vol. 56, no. 16, pp. 3594–3608, 2012.

[7] Cisco, "Cisco Visual Networking Index: Forecast and Trends, 2017-2022 White Paper," 2019. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html

[8] V. Lampkin, W. T. Leong, L. Olivera, S. Rawat, N. Subrahmanyam, and R. Xiang, "Building Smarter Planet Solutions with MQTT and IBM WebSphere MQ Telemetry," 2012. [Online]. Available: https://www.redbooks.ibm.com/redbooks/pdfs/sg248054.pdf

[9] C. Prehofer, "Models at REST or modelling RESTful interfaces for the Internet of Things," in *IEEE World Forum on Internet of Things*, 2015, pp. 251–255.

[10] T. Yokotani and Y. Sasaki, "Transfer protocols of tiny data blocks in IoT and their performance evaluation," in *IEEE 3rd World Forum on Internet of Things*, 2017, pp. 54–57.

[11] S. Sreeraj, N. Suresh Kumar, and G. Santhosh Kumar, "A framework for predicting the performance of IoT protocols, a use case based approach," in *International Conference On Smart Technology for Smart Nation, (SmartTechCon)*, 2017, pp. 577–580.

[12] H. W. Chen and F. J. Lin, "Converging MQTT resources in ETSI standards based M2M platform," in *IEEE International Conference on Internet of Things, iThings, IEEE International Conference on Green Computing and Communications, and IEEE International Conference on Cyber-Physical-Social Computing*, 2014, pp. 292–295.

[13] M. H. Asghar and N. Mohammadzadeh, "Design and simulation of energy efficiency in node based on MQTT protocol in Internet of Things," in *International Conference on Green Computing and Internet of Things, ICGCIoT*, 2016, pp. 1413–1417.

[14] M. A. Alharbe, "Awarenessability and influences on raising of traffic accidents through the content of social media in the internet of things: A practical empirical study by the internet of things and multimedia on university students in western Saudi Arabia," in *International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud), I-SMAC 2018*, 2018, pp. 48–51.

[15] P. Yenkar and S. D. Sawarkar, "A survey on social media analytics for smart city," in *International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud), I-SMAC 2018*, 2019, pp. 87–93.

[16] B. Jadhav and S. C. Patil, "Wireless Home monitoring using Social Internet of Things (SIoT)," in *International Conference on Automatic Control and Dynamic Optimization Techniques, ICACDOT 2016*, 2016, pp. 925–929.

[17] P. Cooper, "28 Twitter Statistics All Marketers Should Know in 2019," 2019. [Online]. Available: https://blog.hootsuite.com/twitter-statistics/

[18] "Adafruit IO," 2019. [Online]. Available: https://io.adafruit.com/

[19] "Raspberry Pi," 2019. [Online]. Available: https://www.raspberrypi.org/

[20] "Twython - Twython 3.6.0 documentation," 2019. [Online]. Available: https://twython.readthedocs.io/en/latest/

[21] "Adafruit_IO_Python on GitHub," 2019. [Online]. Available: https://github.com/adafruit/Adafruit_IO_Python/tree/master/Adafruit_IO