Log Analysis of 360-degree Video Users via MQTT

Antti Luoto
Tampere University of Technology
Tampere, Finland
antti.l.luoto@tut.fi

ABSTRACT

Analysing 360-degree video users is beneficial for 360-degree video application development. The analysis can be done with logged user data. In this paper, we argue that MQTT is a conventional technology for distributed logging of mobile 360-degree video users. MQTT not only saves resources but allows communication from the logging server to mobile clients in various networking conditions relatively easy. We constructed a proof of concept to show the feasibility of the approach. As log analysis examples, the proof of concept visualizes results of the most popular region of interest analysis and k-means clustering. The used research method is design science.

CCS CONCEPTS

• **Information systems** → *Clustering*; • **Networks** → *Network architectures*; • **Human-centered computing** → User studies;

KEYWORDS

360-degree video, MQTT, Log Analysis

ACM Reference Format:
Antti Luoto. 2018. Log Analysis of 360-degree Video Users via MQTT. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

360-degree videos are getting more and more popular. They can be watched with head-mounted displays (HMD) but they appear in other contexts, such as websites, nowadays. We primarily discuss 360-degree videos in hand-held mobile device applications though the techniques are applicable in HMD or web usage as well.

In our earlier work, we made a user logging and visualization framework for 360-degree videos [21]. The framework works on Android and uses the video player offered by Google VR SDK. We presented a technique to place simple graphics over the video and discussed HTTP based user logging. We made the work with lightweightness and resource usage in mind. In this paper we propose improvements to the framework.

Low resource usage can be seen as one of the essential aspects of 360-degree video development [20], and HTTP is not an optimal solution for logging device orientation. 360-degree video user logging applications need to send small log entries often, for example 23 times a second, which causes overhead with HTTP.

We believe that the resource usage of the framework could be improved by using a lightweight communication protocol MQTT. Besides being lightweight, publish-subscribe pattern of MQTT brings network architectural benefits in multi-user environment and it supports applications located in various networking conditions suffering from bad connections or NATs.

One application for user log data is data analysis. In our earlier work, we did not discuss data analysis of the logged data. Since that, we have reported visual heat map analysis of the logged data in a web user interface [10]. This time, we present data analysis for the logged data that also benefits from MQTT. We implemented SQL-based solutions for calculating the most popular region of interest in the video and k-means clustering algorithm. MQTT helps sending the results for visualization on the mobile devices without a need for requesting them.

Thus, the challenges we aim to solve with MQTT are:

- Supporting near real-time log analysis.
- Saving resources on a battery powered mobile device.

The used research method is design science. It is a research method used in software engineering [31] that includes six steps: *problem identification and motivation*, *definition of the objectives for a solution*, *design and development*, *demonstration*, *evaluation and communication* [25]. We include all the steps in this paper.

2 BACKGROUND

This section covers the step *problem identification and motivation* for design science method.

2.1 360-degree Videos and User Logging

360-degree videos are omnidirectional spherical videos where the user can choose the direction of the view port. 360-degree videos have applications in security, robotics, education etc.

User logging in 360-degree video domain is an important feature for multiple parties and applications [5, 18, 19, 33]. For example, logs can be used to analyze not only the videos but also the video user interface such as graphical elements or interaction points on top of the video. While we are not, strictly speaking, in virtual reality domain, the discussion of Ritchie et al. [28] about the benefits of user logging in virtual reality applies to our work: it is an almost non-intrusive method of capturing a rich data source for analysis, it minimizes user interactions during the data capture, it has potential to reduce time overhead of the capturing process, and the captured data can also be reused.

Popular data for logging 360-degree video users includes the direction of the view (for example, in Euler angles) and timestamp.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

This is valuable information since many applications are immersive and only show the chosen view port that adapts to the device orientation. One challenge of 360-degree video user logging comes from the frequently updating view. A video with 23 frames per second requires a log packet to be sent every 43th millisecond if every frame triggers a logging event.

2.2 Log Analysis

360-degree video user data can be analyzed visually, for example, with heat maps. Applications for heat maps include analysis to find the areas of interest in the video [5]. Heat map shows an aggregation of users viewing a part of the video by coloring the area. The brighter the color, the more often the part was included in the users' field of view. Continuously updating frame video adds challenge to the heat map generation when compared to static images. Heat map generation is also a resource consuming task [7].

Another visual analysis method is scanpath. Scanpath is a linearly connected dot sequence following the eye movement coordinates in time but it is not good for aggregate data since markers and labels are easily confusing when analyzing multiple users [7]. Löwe et al. [19], for example, generate heat maps and scanpaths.

Visual analysis, however, requires manual work. Data mining algorithms could be an alternative. With data mining methods, we can programmatically find the regions of interest from the video. Visualizing such data mining results can be also less resource consuming than generating a heat map. Of course, producing a very simplified heat map could be an option but we were interested to explore other methods as well. K-means clustering, for example, is a popular clustering algorithm used in data mining that is generally easy to implement, simple, efficient and empirically successful [13].

2.3 Why MQTT

Capturing the device orientation for a single moment does not require a lot of data but the difficulty comes from the frequently updating video frame. Thus, we need to transfer a big number of tiny packets. However, protocol overhead of HTTP causes serious problems when transferring a big number of tiny packets [34].

MQTT is a lightweight messaging protocol that follows the publish-subscribe paradigm, which makes it suitable for resource constrained devices and non-ideal network connectivity conditions. Because of its simplicity, and a very small message header compared with other messaging protocols, it is often recommended for IoT. [6]

MQTT is often proposed for communication from sensors to (edge) gateways in IoT domain but we believe it has other uses as well. We consider mobile phone sensors similar to IoT sensors. In addition, MQTT allows the mobile phone to receive data by subscribing to MQTT topics. Thus it is easy to send data to multiple subscribers, for example, when multiple users watch the same video. An example of a mobile use case of MQTT is Facebook chat [36] that is often used with mobile devices in non-ideal network conditions.

IoT development survey 2016 [29] shows that MQTT and HTTP are the most popular IoT protocols which supports choosing MQTT as an implementation technique. Dizdarevic et al. [6] comment on the survey that "The reason for this is that MQTT and HTTP REST are currently comparably more mature and more stable IoT standards than other protocols."

Still, there are other IoT protocols, such as CoAP, available. It can be argued that MQTT is more suitable for IoT than CoAP because of the publish-subscribe architecture without responses [14]. Moreover, CoAP has more likely packet loss [14]. Furthermore, communication with the users can be restricted by NATs. Several authors mention the benefits of MQTT when communicating beyond NAT [3, 8, 22, 30]. Other suggested techniques for NAT traversal include Websockets, destination network translation (DNAT), virtual private networks (VPN), Universal Plug and Play (UPnP), and TCP hole punching. We consider them to be either more complex, less reliable, difficult to configure or not ideal for saving resources.

3 RELATED WORK

Related work shows that MQTT logging has been used in various domains such as home and industrial automation. Those domains are quite different from mobile 360-degree videos but they show that MQTT has been used for logging in the scientific literature.

The presented logging research from the 360-video perspective includes a few publications about publishing HMD tracking data sets and analyzing them. In contrast to their work, we concentrate more on hand-held usage with aim in lightweightness, supporting multi-user logging and near real-time data analysis.

Resource consumption comparisons between MQTT and HTTP are related work in a sense that we aim to save resources using MQTT. We did not made a resource consumption comparison since the existing studies support the assumption that MQTT saves resources also in our case.

3.1 Logging with MQTT

Avizheh et al. [2] propose a secure event logging system for smart homes using MQTT. They emphasize security by using Bitcoin blockchain for ensuring the integrity of log files. While we did not concentrate on security yet, authentication, authorization and symmetric payload encryption could be the first steps [12, 22].

In another home automation study, Agerwal et al. [1], use MQTT to send home automation data from sensors to Raspberry pi 3 which works as a central unit with data-logging. They use Node-RED and Mosquitto MQTT broker to create a cost-effective open source solution. We also use open source components such as Google VR SDK and Mosquitto.

Wenger et al. [32] discuss lightweight logging in industrial automation environment. In their solution, control devices connect to a cloud MQTT broker publishing logging data so that monitoring service subscribes to logging topics. We are not explicitly discussing cloud environment but, for example, a web based monitoring service can be implemented with the database [10].

Zabasta et al. [35] have an MQTT enabled event handling and historian systems (used for data logging) for utility networks. They note that the responsibilities for those components should be investigated more carefully. Similarly to their work, we use relational database to store the sensor data and use JSON when sending data with MQTT.

Herle et al. [11] propose a REST bridge for using MQTT via standard HTTP methods. They use the bridge as a message logger

which logs all the received messages to a MongoDB database. With their solution users are able to get a whole history of events instead of just the latest event. In contrast, our solution does not provide a way get the whole history of events via MQTT. This is a clear disadvantage when compared to our earlier REST solution.

Kodali and Gorantla [15] made a Python application to receive weather tracking sensor data (JSON) via MQTT broker and store it to SQLite database. The broker, the application and the database are located on the same Raspberry Pi. Kodali and Sarjerao [16] also hint about a similar system where air quality data is continuously stored in a MQTT server. Likewise, we use JSON and relational database, but we have the logger application and the database in one server and MQTT broker on other. The broker is the only component available in the open Internet in our solution.

Harris and Curry [9] note that MQTT does not provide inherent logging capabilities. Therefore they made a tracker for logging MQTT traffic by LoRaWAN devices. Because of the inherent logging capabilities, we also had to implement a logging components.

3.2 Analysing 360-degree Video Users

Lo et al. [18] offer public HMD user logging data sets. Their architecture is primarily for only local logging and they create saliency maps and motion maps with the logs. In contrast, our logging server architecture enables a way to collect data sets simultaneously with multiple users and analyze the logs in real-time.

Corbillon et al. [5] also released a public 360-degree video head movement data set. They used the logs to create example statistics for analyzing users' navigation patterns.

Wu et al. [33] also published a data set for exploring user behavior in spherical videos. They present preliminary analysis of their data set by presenting visualizations and statistics. However, they do not discuss producing statistics in real-time. Examples of their statistics include head movement angular speeds, where as we calculate the most popular region of interest and k-means clustering.

Qian et al. [27] computed weighted linear regression on head orientation data with results that the head orientation can be predicted with over 90% accuracy. They predict single user traces, where as we concentrate on multi-user analysis. However, they propose (not implemented) using multi-user statistics to help with the inaccuracy of single user prediction.

3.3 Resource Consumption Between MQTT and HTTP

Luoto and Systä [22] made a memory and CPU usage comparison between MQTT and HTTP. MQTT used less resources even with the suggested request-response pattern and symmetrically encrypted message payload. Comparison with 1000 request-responses is applicable to 360-degree video domain because 23 seconds of 23 FPS video produces about 1000 log entries.

Chen and Lin [4] compared MQTT and HTTP proxies with the result that the MQTT proxy had a shorter latency and saved more power than the HTTP proxy. While proxy comparison is not directly related to our work, their results suggests that MQTT could use less power in our solution as well.

A comparison of power consumption between MQTT and HTTP [23] on Android shows that MQTT has lower power consumption for example on maintaining open connection and sending and receiving messages (when compared to HTTPS long polling). We also use Android and aim to reduce power consumption by choosing MQTT.

Yokotani et al. [34] made comparisons in different scenarios with 10, 100 and 1000 devices with MQTT and HTTP. They concluded their study so that MQTT performs better than HTTP. While we could not evaluate our work even with ten devices, such results encouraged us even more to choose MQTT, and evaluation with more devices could be future work.

4 SOLUTION

To define the *objectives for the solution* step of design science method, our objectives are: saving resources by choosing other network protocol than HTTP, and supporting near real-time data analysis by enabling an easy access to analysis results in a realistic Internet architecture with NATs. The section also covers the steps *design and development*, and *demonstration* (as far as it is possible in with text and images).

4.1 Original HTTP Logging

The original logging and visualization framework runs on Android mobile phone [21]. It uses the class VrVideoView of Google VR SDK to create a video player in Android application. OnNewFrame function of the class is executed when the video frame updates. This also triggers our logging events. So, we get log only from drawn frames and sometimes the player skips frames. The most important logged information contains yaw, pitch, accelerometer values (x, y, z), and video time. Yaw and pitch present rotation in a spherical space. Yaw is the vertical angle between -180 and 180 degrees, and pitch is the horizontal angle between -90 and 90 degrees.

The log entry was sent to RESTful logging server which used PostgreSQL as database. The used data format was JSON. The logged data was used, for example, to create heat maps in a web application [10]. In addition, we could visualize traces of the previous view sessions in the Android application. Creating a HTTP request for every frame in 23 FPS video already makes a lot of traffic, and the FPS can be even higher. The log entries could be sent as batches but it does not support the near real-time aim of the study.

4.2 MQTT Logging

To test the feasibility of MQTT on our objectives, we implemented a proof of concept. The architecture of the proof of concept can be divided to three main components: mobile phones with video players, MQTT broker delivering messages, and back end with Logger and database. The used MQTT broker is Mosquitto. Figure 1 shows the components on a high abstraction level.

4.2.1 Logger. We implemented the logger application with Node.js and MQTT.js library. The logger subscribes to topics for registering a viewing session and sending logging data. After receiving the data it stores the data to the database. It also publishes data analysis results.

4.2.2 Mobile Client. We used the class MqttAndroidClient of the Eclipse Paho project to implement MQTT on Android. The mobile application presented in our earlier work was not changed other

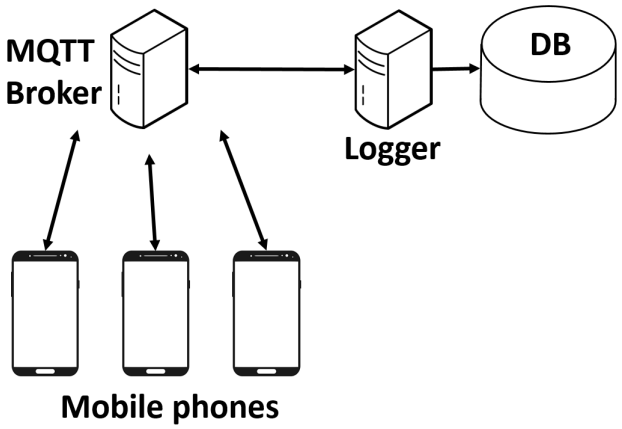

Figure 1: Components of the proof of concept. Bidirectional arrows illustrate that both the logger and the phones publish and subscribe.

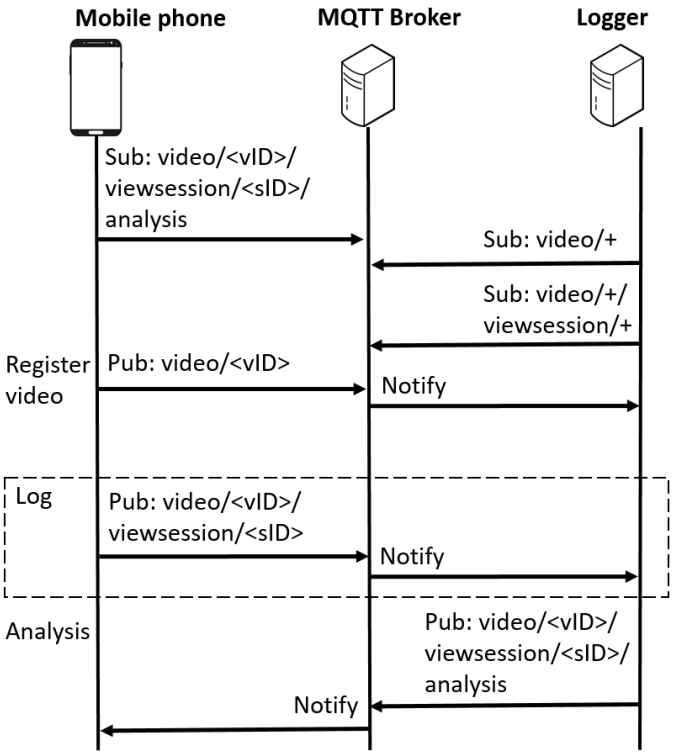

Figure 2: MQTT communication. The dashed box illustrates an operation that happens the most often. Subscriptions and video registration happen once in a view session. Frequency of analysis can be adjusted. Plus character is MQTT wildcard that matches a single topic level. vID refers to video URL hash and sID to view session hash.

than by refactoring the HTTP related code to use MQTT, adding subscribing to the data analysis topic, and drawing the analysis

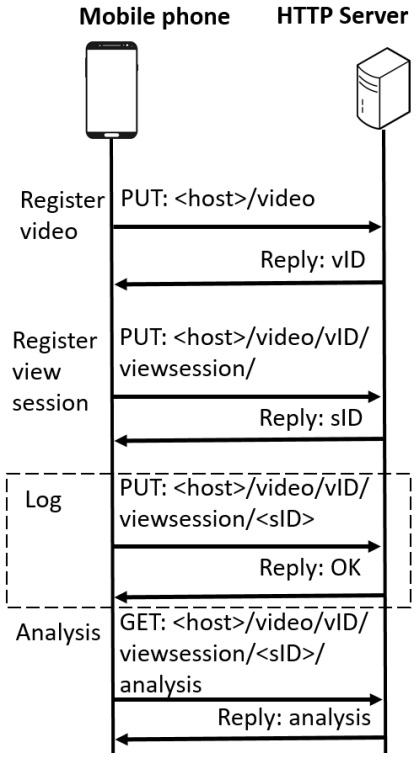

Figure 3: HTTP communication for comparison with MQTT version. The same steps can be found from both the sequences (except that registering a view session is included in 'log' step in MQTT version). The last request is hypothetical since it was not implemented with HTTP.

results on video (using the visualization technique presented in our earlier work). Log message payloads are still in JSON format.

4.2.3 MQTT Topic Design. MQTT topics are strings that the broker uses for delivering messages for interested clients. Clients can subscribe to a topic to receive messages published to that topic. MQTT topics have levels that are separated by slash characters.

The topics needed to be designed with one-way communication in mind. For example, in request-response pattern, it is possible to ask for a unique client id from the back end, and then use that for identifying view sessions. Since we can not do request-response with a publish-subscribe protocol, we had to generate the needed ids in the video player client. For example, when a user starts to watch a video, the video application generates an id for the view session using a hash function. Then, when the application sends the first log entry, it also registers a view session using an MQTT topic which contains the generated hash. For example, publishing for the first time to topic *video/(video URL hash)/viewsession/(view session hash)* registers a new view session and adds the log data (from payload) for the video identified by *view session hash*. Later publishing to the same topic, only adds the log data to database since the view session is already generated.

Figure 2 presents a how the MQTT communication works between the mobile clients and the logger. For comparison, Figure 3 shows an original HTTP version of the same sequence.

The MQTT topic structure is based on three base topics:

- *video/(video URL hash)* – For registering a new video to the system. The recipient can parse the hash from the topic string of the incoming message while the message payload contains only the original URL.
- *video/(video URL hash)/viewsession/(view session hash)* – For registering a new view session, and publishing log entries related to certain video. Again, the hashes can be parsed from the incoming topic string. The payload contains the logged data (orientation, video time etc.).
- *video/(video URL hash)/viewsession/(view session hash)/analysis* – For the back end to send analysis results about a certain view session to a device. *Analysis* could be replaced with a descriptive name of the analysis such as *popularRegionOfInterest*.

4.2.4 Database. The log database consists of three tables. Table Video contains URL of the video and a shortened hash of the URL used to form MQTT topics. We used shortened hashes (10 characters) because long topic names cause overhead in MQTT [34]. Table Viewsession contains an id for the current view session. These ids are used to identify certain view sessions in MQTT topics and to send back analysis information to devices. The last table is for the actual log entries. The database is implemented with PostgreSQL.

4.3 Log Analysis

4.3.1 Algorithms. We believe that the most convenient place to perform data mining would be the database in our architecture. Thus, we would not need to first retrieve the data, and then calculate it with the Logger or even with mobile phone. Retrieving lots of data could be heavy especially with multiple simultaneous users.

Our first algorithm calculated the most popular region of interest. The algorithm divides the 360-degree sphere to a graticule of 30 times 30 degree spherical rectangles. This results in 72 spherical rectangles if we count the topmost and bottommost spherical triangles as rectangles for the sake of simplicity. From now on, we call these rectangles as tiles. A sphere can be tiled in different 30 degrees was chosen because the minimum and maximum for both the yaw (from -180 to 180 degrees) and pitch (from -90 to 90 degrees) are divisible by 30 and it felt conventional enough, but it could be any other value as well. Figure 4 visualizes the division. After that, every coordinate within a second is positioned in one of the tiles. For example, a coordinate in 10 degrees of yaw 10 degrees of pitch is positioned in a tile defined by area of 0–30 degrees of yaw and 0–30 degrees of pitch. Then the algorithm counts the tiles with most positioned coordinates and returns an ordered list. With an ordered list it is easy to get, for example, the three most popular tiles and visualize them in the application.

PostgreSQL evaluates comparisons in SELECT statement as true or false and returns accordingly either 't' or 'f' character. We use a combination of those characters to identify a certain tile on the sphere surface. Each tile is identified by a combination of 16 't' and 'f' characters. The first 11 characters identify the yaw position and five latter characters identify the pitch position of the tile. The

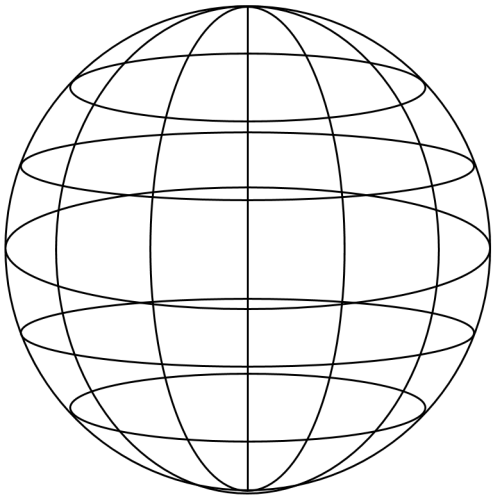

Figure 4: A sphere divided to graticule of 72 tiles each with 30 times 30 degrees area.

string has from zero to two 't' characters and the rest are 'f'. So, for example, 'ftfffffffffftf' refers to a tile with left bottom corner located in -120 degrees in yaw and -30 degrees in pitch and tile identified by 16 'f' characters has left bottom corner located in 150 degrees in yaw and 60 degrees in pitch. The combination resembles a binary number and could be converted to an integer easily. However, on the application side it is straightforward to iterate through the result table and multiply 30 degrees with index of 't' to find the correct position in degrees. Figure 6 shows measured execution times on a modern laptop (Thinkpad W541, Windows 10, PostgreSQL 10.5) where execution times seem to be near linear in relation to amount of rows with our logged data. The following listing shows the SQL source code of the algorithm:

```
SELECT *, COUNT(tile) FROM
  (SELECT yaw < -150,
    yaw BETWEEN -150 AND -120,
    yaw BETWEEN -120 AND -90,
    -- ... (going through all the
       30-degree yaw sections)
    yaw BETWEEN 120 AND 150,
    pitch < -60,
    pitch BETWEEN -60 AND -30,
    -- ... (going through all the
       30-degree pitch sections)
    pitch BETWEEN 30 AND 60
    FROM record
    WHERE time BETWEEN 0 AND 1000) AS tile
GROUP BY 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
    12, 13, 14, 15, 16
ORDER BY count DESC
```

To elaborate the query, the derived table named 'tile' contains truth values. The query that produces that derived table uses table 'record' that contains rows with yaw, pitch and time. In the example,

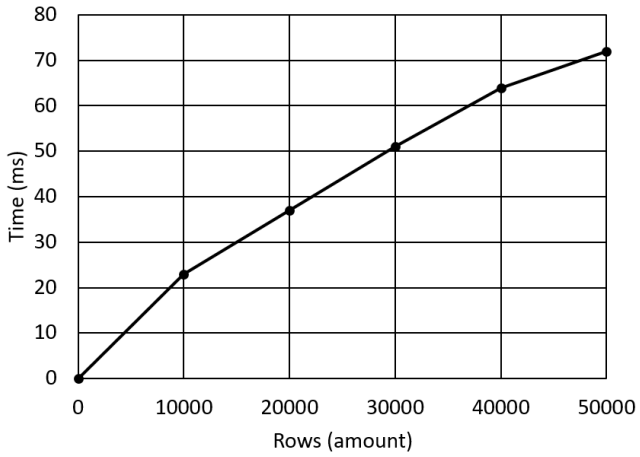

Figure 5: Execution times of the SQL query calculating the most popular tile.

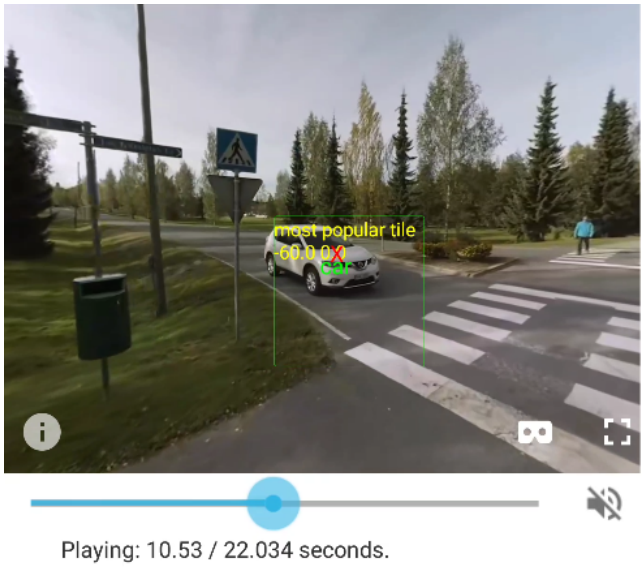

Figure 6: Example of most popular tile visualization. -60.0 tells the yaw angle and 0.0 pitch angle of the tile. Label 'car' comes from the metadata visualization presented in our earlier work. Red 'X' marks the center of the view port.

we choose time between 0 and 1000 ms but it could other timeframe as well. BETWEEN operator is inclusive meaning that begin and end values are included but we do not consider that a disruptive problem with the proof of concept. The main query counts the amount of different rows in the derived table 'tile'. The numbers (1–16) in GROUP BY refer to the unnamed columns of the derived table. Then the final result is ordered in descending order according to the amount of different truth value combinations. One row of the main query result contains 16 truth values and how many times such combination was included in the derived table 'tile'.

The second algorithm we implemented was k-means clustering algorithm. With the algorithm, we strive to find clusters from the orientation data inside the timeframe of one second. While implementing data mining algorithms with SQL is not always considered efficient or feasible, k-means clustering can be implemented in SQL efficiently and flexibly [24]. We used a recursive PostgreSQL solution presented in Periscope Data blog [26] with minor modifications (which is not standard SQL). An example of the flexibility of the SQL implementation was that limiting the chosen timeframe (one second) is easy using SQL BETWEEN operator. Experimenting the query with our test data, using two clusters and ten iterations, takes 850 ms with 10000 log records, 2200 ms with 30000 log records and 3700 ms with 50000 log records. By halving the iterations to five, the query execution times halve but the accuracy declines. Finally, we send the cluster information to the devices for visualization.

4.3.2 Receiving Analysis Results. With HTTP the common way to get the analysis results would be to request them from the back end. However, the back end can be protected by NAT not allowing incoming requests. MQTT can be used for NAT traversal which does not require special work other than setting up the MQTT broker and making it accessible by the parties involved in MQTT messaging. This works also the other way around if a mobile client is behind NAT and the back end needs to push data to it. Such functionality makes the architecture flexible.

5 EVALUATION

In this section we evaluate the solution and the design decisions as required by the step *evaluation and communication* of design science. The criticism is directed at significance of the resource savings, claimed real-time functionality, heaviness of k-means clustering in SQL, id clashing, what we lost with HTTP, our MQTT topic design, and generally the usefulness of k-means clustering in our use case.

Is it significant to save resources by optimizing logging if a 360-degree video application already consumes much computational resources? If the logging is frequent (for example, multiple times a second) then logging users in long videos can take a considerable amount of resources. Let us assume a scenario where application makes a log entry ten times a second. In two minutes we will end up with 1200 log entries. 1000 messages with either HTTP or MQTT already makes a clear difference in resource usage [22]. In addition, the presented ideas are applicable to software without 360-degree video players as well, such as, mobile applications logging device orientation or other often updating sensors. Resource savings with other applications could be even more clear if the basic functionality is not as heavy as, for example, playing 360-degree video.

We aim to support near real-time analysis. In our case real-time means that user gets the analysis results visualized on the video so that the lag is not distracting. Measuring a tolerable lag is out of the scope of this paper. However, in our database the maximum time used for the most popular region of interest analysis was about 70 ms with 50000 log records added to the time of transferring the data. We did not consider this lag distracting. One reason is also that the most popular location changes so slowly that updating it around once a second provided good enough user experience. Even less sparse updating pace could be enough which would reduce the workload of the back end. Bigger tiles also shorten SQL query

time, for example diving the sphere to 60 times 30 degrees tiles reduces the query time around 15 ms with 50000 log records. K-means clustering starts to be too slow with our test data but it could be optimized further.

While MQTT is considered relatively mature technology [6], combining Android (Java) and MQTT was not especially straightforward and could be a dead-end for an inexpericed developer. In contrast, our experience suggests that starting MQTT development with Node.js and MQTT.js is easier.

Since we generate ids used for logging on Android devices, and try to keep them short to prevent MQTT topic over head, id clashing is possible. The possibility increases when the amount of users grows. We did not consider this an important issue with the proof of concept but it needs to be solved for real usage scenarios. One solution could be to use centralized id generation technique and use MQTT request-response to receive unique ids for clients [22].

Long topic names cause overhead with MQTT [34]. Our topics could be shorter but we wanted to keep the structure readable and hierarchical. It also allows the flexible usage of MQTT wildcards, for example, if a party is interested in all the messages related to a certain video. The topic for analysis results is less frequently used than the others, so lengthy topic overhead is not as harmful with that particular topic.

With the original HTTP solution, it was easy to create a Node.js server for browsing the logging results via WWW and REST. Web browsers do not directly support MQTT so requesting and browsing the logging history for a certain view session is not as straightforward. Generally using MQTT makes it more difficult to implement a browser based user interface.

Our division of spherical space in spherical rectangles results so that the topmost and bottommost rectangles are actually spherical triangles. This can add a bit of complexity for visualizing the tiles. On the other hand, we assume that most of the videos have most of the interesting content relatively near the equator of the sphere and thus concentrating on those areas has higher priority.

Using PostgreSQL for k-means clustering was relatively simple. However, if k-means clustering is too heavy it can be lightened, for example, by reducing the amount of iterations, shortening the timeframe, or limiting the amount of data (SQL LIMIT operator). Ordonez [24] discusses multiple optimization ideas. It is also good to note that PostgreSQL allows running Python (PL/Python) and R (PL/R) for more advanced data mining but using those might bring overhead when compared to raw SQL.

One challenge of k-means clustering is that the correct amount of clusters is not always clear beforehand. For experimentation, we used two clusters with the proof of concept but one way to determine a proper suggestion for a good amount of clusters could be elbow method [17].

It is good to note that our analysis does not tell the actual gaze point because our devices can not track users' gaze. But it is a problem with all the techniques and devices that do not track the gaze. However, the ideas presented in this paper could be applicable for gaze tracking as well.

6 CONCLUSIONS

In this paper, we showed how MQTT can be used for logging 360-degree video users' device orientation. The work is based on earlier work about logging users with HTTP. By choosing MQTT we aim to save computational resources, and furthermore battery usage of mobile devices. Using MQTT also enables relatively easy NAT traversal which makes publishing data in realistic network environment easier. This way we can send log analysis results to the clients without a need for specific request-response for continuous updating. As a log analysis example, we discuss an SQL algorithms for calculating the most popular region of interest and k-means clustering. Such data analysis could be used, for example, to show announcements or advertisements on popular regions of interest on the video.

ACKNOWLEDGMENTS

The authors would like to thank Business Finland for funding the project 360 Video Intelligence.

REFERENCES

[1] Archit Agarwal, Rajesh Singh, Anita Gehlot, Gautam Gupta, and Mohit Choudhary. 2017. IOT Enabled Home Automation through Nodered and MQTT. *International Journal of Control Theory and Applications* 10, 18 (2017), 255–260.

[2] Sepideh Avizheh, Tam Thanh Doan, Xi Liu, and Reihaneh Safavi-Naini. 2017. A Secure Event Logging System for Smart Homes. In *Proceedings of the 2017 Workshop on Internet of Things Security and Privacy*. ACM, 37–42.

[3] Paolo Bellavista and Alessandro Zanni. 2016. Towards better scalability for IoT-cloud interactions via combined exploitation of MQTT and CoAP. In *Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI), 2016 IEEE 2nd International Forum on*. IEEE, 1–6.

[4] Hsiang Wen Chen and Fuchun Joseph Lin. 2014. Converging MQTT resources in ETSI standards based M2M platform. In *Internet of Things (iThings), 2014 IEEE International Conference on, and Green Computing and Communications (GreenCom), IEEE and Cyber, Physical and Social Computing (CPSCom), IEEE*. IEEE, 292–295.

[5] Xavier Corbillon, Francesca De Simone, and Gwendal Simon. 2017. 360-Degree Video Head Movement Dataset. In *Proceedings of the 8th ACM on Multimedia Systems Conference*. ACM, 199–204.

[6] Jasenka Dizdarevic, Francisco Carpio, Admela Jukan, and Xavi Masip-Bruin. 2018. Survey of Communication Protocols for Internet-of-Things and Related Challenges of Fog and Cloud Computing Integration. *arXiv preprint arXiv:1804.01747* (2018).

[7] Andrew T Duchowski, Margaux M Price, Miriah Meyer, and Pilar Orero. 2012. Aggregate gaze visualization with real-time heatmaps. In *Proceedings of the Symposium on Eye Tracking Research and Applications*. ACM, 13–20.

[8] Paul Fremantle. 2014. A reference architecture for the internet of things. *WSO2 White paper* (2014).

[9] Nicholas Harris and Josh Curry. 2018. Development and range testing of a LoRaWAN system in an urban environment. *World Academy of Science, Engineering and Technology, International Journal of Electrical, Computer, Energetic, Electronic and Communication Engineering* 12, 1 (2018), 43–51.

[10] Pietari Heino. 2018. *360-videoiden katselustatistiikan visualisointi*. Master's thesis. Tampere University of Technology (TUT), Tampere, Finland.

[11] Stefan Herle, Ralf Becker, and Jörg Blankenbach. 2016. Bridging GeoMQTT and REST. In *Geospatial Sensor Webs Conference*.

[12] HiveMQ. [n. d.]. MQTT Security Fundamentals: MQTT Payload Encryption. ([n. d.]).

[13] Anil K Jain. 2010. Data clustering: 50 years beyond K-means. *Pattern recognition letters* 31, 8 (2010), 651–666.

[14] Vasileios Karagiannis, Periklis Chatzimisios, Francisco Vazquez-Gallego, and Jesus Alonso-Zarate. 2015. A survey on application layer protocols for the internet of things. *Transaction on IoT and Cloud Computing* 3, 1 (2015), 11–17.

[15] Ravi Kishore Kodali and Venkata Sundeep Kumar Gorantla. 2017. Weather tracking system using MQTT and SQLite. In *2017 3rd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT)*. IEEE, 205–208.

[16] Ravi Kishore Kodali and Borade Samar Sarjerao. 2017. MQTT based air quality monitoring. In *Humanitarian Technology Conference (R10-HTC), 2017 IEEE Region 10*. IEEE, 742–745.

[17] Trupti M Kodinariya and Prashant R Makwana. 2013. Review on determining number of Cluster in K-Means Clustering. *International Journal* 1, 6 (2013), 90–95.

[18] Wen-Chih Lo, Ching-Ling Fan, Jean Lee, Chun-Ying Huang, Kuan-Ta Chen, and Cheng-Hsin Hsu. 2017. 360Âř Video Viewing Dataset in Head-Mounted Virtual Reality. In *Proceedings of the 8th ACM on Multimedia Systems Conference*. ACM, 211–216.

[19] Thomas Löwe, Michael Stengel, Emmy-Charlotte Förster, Steve Grogorick, and Marcus Magnor. 2015. Visualization and analysis of head movement and gaze data for immersive video in head-mounted displays. In *Proceedings of the Workshop on Eye Tracking and Visualization (ETVIS)*, Vol. 1.

[20] Antti Luoto. 2017. Towards Framework for Choosing 360-degree Video SDK. In *SIGMAP*. 81–86.

[21] Antti Luoto. 2018. Lightweight Visualization and User Logging for Mobile 360-degree Videos. In *11th Workshop on Software Engineering and Architectures for Realtime Interactive Systems (SEARIS) (to be published)*. IEEE.

[22] Antti Luoto and Kari Systä. 2018. Fighting network restrictions of request-response pattern with MQTT. *IET Software* (2018).

[23] Stephen Nicholas. 2013. Power Profiling: HTTPS Long Polling vs. MQTT with SSL, on Android. (2013).

[24] Carlos Ordonez. 2006. Integrating K-means clustering with a relational DBMS using SQL. *IEEE transactions on Knowledge and Data engineering* 18, 2 (2006), 188–201.

[25] Ken Peffers, Tuure Tuunanen, Marcus A Rothenberger, and Samir Chatterjee. 2007. A design science research methodology for information systems research. *Journal of management information systems* 24, 3 (2007), 45–77.

[26] Periscopedata. 2015. Multi-dimensional Clustering Using K-Means in Postgres SQL. (2015).

[27] Feng Qian, Lusheng Ji, Bo Han, and Vijay Gopalakrishnan. 2016. Optimizing 360 video delivery over cellular networks. In *Proceedings of the 5th Workshop on All Things Cellular: Operations, Applications and Challenges*. ACM, 1–6.

[28] James M Ritchie, Raymond CW Sung, Heather Rea, Theodore Lim, Jonathan R Corney, and Iris Howley. 2008. The use of non-intrusive user logging to capture engineering rationale, knowledge and intent during the product life cycle. In *Management of Engineering & Technology, 2008. PICMET 2008. Portland International Conference on*. IEEE, 981–989.

[29] I Skerrett. 2016. IoT Developer Survey 2016. *Eclipse IoT Working Group, IEEE IoT and Agile IoT* (2016), 1–39.

[30] Minoru Uehara. 2015. A case study on developing cloud of things devices. In *Complex, Intelligent, and Software Intensive Systems (CISIS), 2015 Ninth International Conference on*. IEEE, 44–49.

[31] Vijay Vaishnavi and William Kuechler. 2004. Design research in information systems. (2004).

[32] Monika Wenger, Alois Zoitl, Jan Olaf Blech, Ian Peake, and Lasith Fernando. 2015. Cloud based monitoring of timed events for industrial automation. In *Parallel and Distributed Systems (ICPADS), 2015 IEEE 21st International Conference on*. IEEE, 827–830.

[33] Chenglei Wu, Zhihao Tan, Zhi Wang, and Shiqiang Yang. 2017. A Dataset for Exploring User Behaviors in VR Spherical Video Streaming. In *Proceedings of the 8th ACM on Multimedia Systems Conference*. ACM, 193–198.

[34] Tetsuya Yokotani and Yuya Sasaki. 2016. Comparison with HTTP and MQTT on required network resources for IoT. In *Control, Electronics, Renewable Energy and Communications (ICCEREC), 2016 International Conference on*. IEEE, 1–6.

[35] Anatolijs Zabasta, Kaspars Kondratjevs, Janis Peksa, and Nadezda Kunicina. 2017. MQTT enabled service broker for implementation arrowhead core systems for automation of control of utility'systems. In *Advances in Information, Electronic and Electrical Engineering (AIEEE), 2017 5th IEEE Workshop on*. IEEE, 1–6.

[36] Lucy Zhang. 2011. Building Facebook Messenger. (2011).