CITY IoT

Vipuvoimaa EU:lta 2014–2020

European Union
European Regional Development Fund

CityIoT Case: MQTT and 360-degree Video User Data

Antti Luoto

May 28, 2020

6Aika · Uudenmaan liitto Nylands förbund · OULU · TAMPERE · OULUN YLIOPISTO · OAMK · Tampereen yliopisto Tampere University

Contents

1 Introduction

The document has been written for a project called CityIoT[1]. One of the aims of CityIoT project was to define an open and operator independent data integration platform for smart cities and a coherent reference architecture. This document has been written for audience that has a basic knowledge of smart city platform FIWARE. See, for example, a document about Tampere University FIWARE platform[2].

Despite that the platform has been used by Tampere University in multiple pilots in CityIoT, none of those used an important IoT technology called Message Queuing Telemetry Transport[3] (MQTT). The case described in this report was designed to fill that gap, in addition to giving an example of how to use Tampere University FIWARE platform.

360-degree video users provide a motivating data source for experimenting with a smart city platform. The popularity of 360-degree videos has increased recently and they have applications in multiple smart city related domains such as surveillance, remote working, robotics, traffic etc. 360-degree videos can be watched with smart phones that are equipped with various sensors. The data collected by those sensors can be sent to other parties, such as a smart city platform, using MQTT.

The 360-degree video user data collected in the described case is just an example. Tampere University FIWARE platform could be easily used for collecting and visualizing other data sent via MQTT as well.

The rest of the report is organized as follows. Section 2 gives background information about MQTT and 360-degree video user logging. Section 3 presents the components we used to build the experiment. Section 4 tells the steps that are required for setting up MQTT for FIWARE. Section 5 presents the visualizations that were produced by using the dashboard tool Grafana. Section 6 discusses other important topics that were left with little consideration during the case. Section 7 concludes the report.

2 Background

The case described in this document was about using MQTT to send 360-degree video user data to FIWARE platform for visualization.

2.1 MQTT

MQTT is a lightweight publish-subscribe data transfer protocol aimed for constrained devices. While MQTT is mainly used to send data from low-resource IoT sensors to (edge) gateways, we think it can be used with smart phones as well, especially when a lot of small data packets are sent, request-response is

[1] https://www.cityiot.fi/english
[2] https://drive.google.com/file/d/1yueGrdArlFmz8ZzchTXWuhbgC9dKUuGN/view
[3] http://mqtt.org/

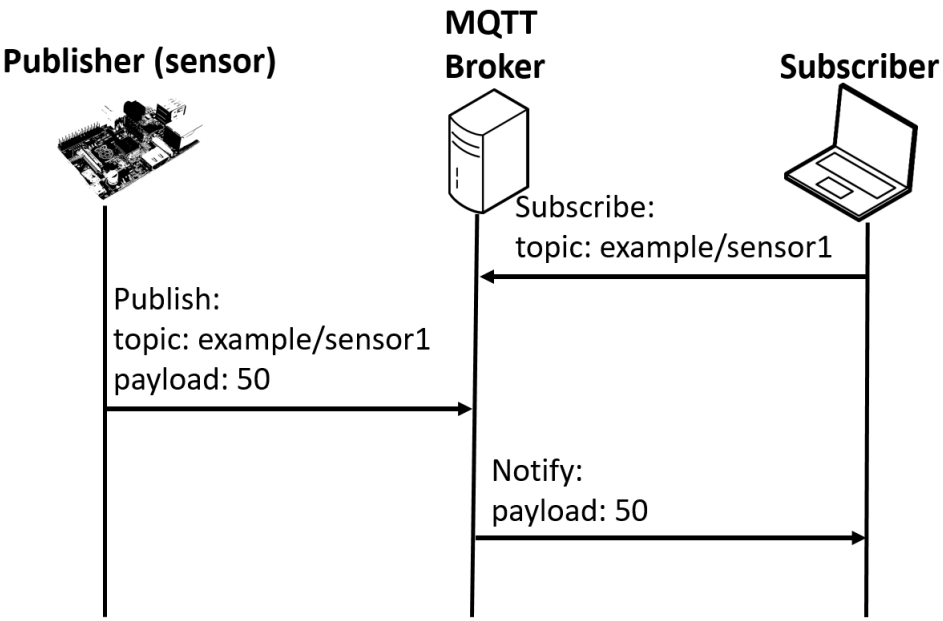

Figure 1: Example sequence diagram of an IoT sensor publishing data to subscribing back end.

not needed, and there can be multiple simultaneous data sources. In addition, MQTT is among the most popular IoT protocols.

MQTT has four main concepts: publisher, subscriber, broker, and topic. Publishers publish messages to topics that are managed by the broker. The broker then forwards the messages to the subscribers that have subscribed to certain topics of interest. For example, a publisher sends a message to topic 'example/sensor1' which is then delivered, by the broker, to subscribers that have subscribed that topic. Figure 1 presents a sequence diagram of the communication between a subscriber, a publisher and a broker. There could be multiple publishers and multiple subscribers using the same topic. The example payload 50 is a value measured by the sensor, for example, 50 Celsius degrees.

Figure 2 presents an overview of how mobile phones communicate with Tampere University FIWARE platform via MQTT. The idea is that smart phones publish data to MQTT broker that has subscriptions from IoT Agent in FIWARE platform. While the database and the visualization concepts presented in the Figure are not a part of MQTT communication, they express that the data gathered from mobile phones is stored to a database and can be used for producing visualizations with FIWARE platform.

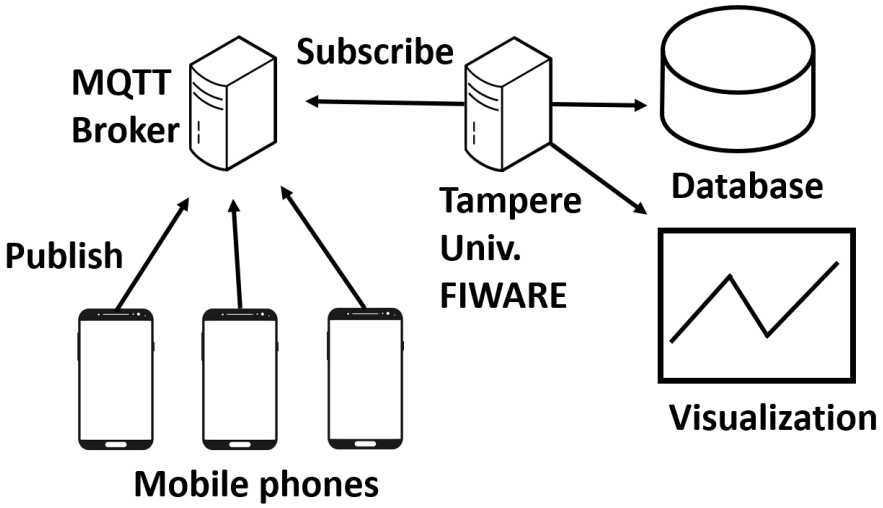

Figure 2: An overview of how MQTT integrates to Tampere University FIWARE platform.

2.2 360-degree Video User Logging

When watching 360-degree videos with smart phones, the idea is that a video adapts to the phone orientation so that when a phone is turned, the played video turns accordingly in a spherical manner. The most important data gathered from users is the rotation of their device while playing 360-degree video. Yaw, pitch and roll format is one way of presenting rotations in a spherical space. Yaw is the vertical rotation, pitch is the horizontal rotation, and roll is the rotation around front-to-back axis. We use that format as well but without roll. The main reason for that is that the API of Google VR SDK used by the smart phone application does not provide the roll angle.

3 Implementation

The architecture of FIWARE platform used at Tampere University can be seen in Figure 3. IoT Agent is missing from Figure but it is a FIWARE core component communicating with Orion Context Broker and a MQTT broker. We used Mosquitto MQTT broker[4] installed on the same server with our FIWARE platform. For more information about the components see the document about Tampere University FIWARE platform (Footnote 2).

[4] https://mosquitto.org/

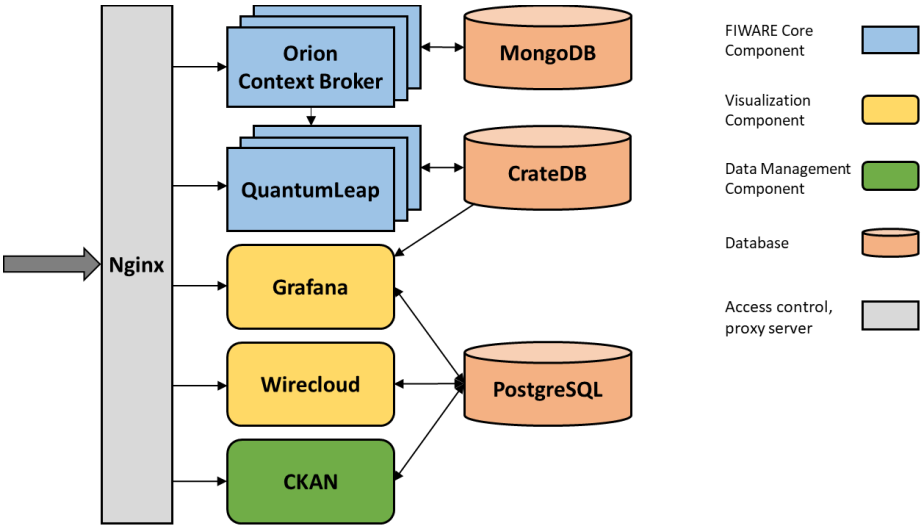

Figure 3: FIWARE platform architecture used at Tampere University.

3.1 IoT Agent

We had an existing FIWARE platform with an HTTP API but without MQTT support. We had to install FIWARE IoT Agent to enable MQTT. According to FIWARE documentation[5]: "An IoT Agent is a component that lets groups of devices send their data to and be managed from a FIWARE NGSI Context Broker using their own native protocols. IoT Agents should also be able to deal with security aspects of the FIWARE platform (authentication and authorization of the channel) and provide other common services to the device programmer." There are different IoT Agents available such as IoT Agent for the Ultralight (UL) 2.0 protocol and IoT Agent for JSON both of which support MQTT. We decided to to use Ultralight 2.0 this time since it was easy to convert our existing simple JSON data to Ultralight format.

"Ultralight 2.0 is a lightweight text based protocol aimed to constrained devices and communications where the bandwidth and device memory may be limited resources [sic]."[6] Ultralight 2.0 does not order the use of communication protocol, only the format the of the text payload. In Ultralight messages the payload is composed of a list of key-value pairs separated by the vertical bar character (|). For example, payload a|3|b|xyz has two attributes, one named 'a' with value '3' and another named 'b' with value 'xyz'. Such a message could contain values measured by a sensor sending them to an IoT platform. Ultralight, as such, was not very essential in this experiment. The format is just explained here to clarify a few things later, and as a secondary research result, using Ultralight was plain and simple.

[5] https://github.com/telefonicaid/iotagent-node-lib/tree/d76d0216f6d2247bcc2131ebbf81c74867afa447

[6] https://github.com/telefonicaid/iotagent-ul/tree/8557733aaac1a7428f295eec7b74dac8b805e91e

Figure 4: Screenshot of the smart phone application.

In this experiment, the HTTP API of IoT Agent was used for setting up the MQTT service and provisioning IoT devices. After being provisioned, the devices can send data to the platform via MQTT.

3.2 Smart Phone Application

The smart phone application uses Google VR SDK for Android[7]. It has a 360-degree video player with basic controls (play, stop, volume control etc.). The application is designed to be used hand-held instead of being a part of a head-mounted display. MQTT library used in the application is Eclipse Paho. Figure 4 presents an example view on the application which is playing a 360-degree video recorded from bicycle helmet while cycling in traffic. In the video player of Google VR SDK, yaw is between -180 and 180 degrees, and pitch is between -90 and 90 degrees.

3.3 Visualization

Grafana[8] is an open source visualization platform for multiple databases. It is relatively easy to create dashboards with its web user interface. We used version

[7] https://developers.google.com/vr/develop/
[8] https://grafana.com/

6.5.3.

The basic graph visualization provided by Grafana is not conventional for visualizing user traces of non-streaming videos since charts require having real-world timestamps on X axis. Luckily, a Grafana plugin named Plotly allows using any data on X axis. With this feature, it is possible to set video time on X axis. Installing Grafana plugins is easy in Tampere University FIWARE platform. The plugin name just need to be added to a file containing environment variables in addition to restarting Grafana.

3.4 QuantumLeap

QuantumLeap application stack contains a database that can be used to store the FIWARE NGSIv2 data as time series data. In FIWARE the data is not automatically copied from Orion to history component unless there is a subscription for that. Thus, if an entity should be sent to Orion and stored in a history component, a new subscription should be added. The normal way to store FIWARE NGSIv2 data with QuantumLeap is to make a subscription to Orion context broker where the `/notify` endpoint of the QuantumLeap API is given as the target address for any notifications based on the subscription parameters. Here is an example of subscription for getting all the data from a certain FIWARE service to Quantumleap made with command line tool curl:

```
curl -iX POST \
'https://tlt-cityiot.rd.tuni.fi/orion/v2/subscriptions' \
-H 'Content-Type: application/json' \
-H 'fiware-service: 360video' \
-H 'fiware-servicepath: /' \
-d '{"description": "360video subscription",
     "subject": {
       "entities": [
         {
           "idPattern": ".*"
         }
       ]
     },
     "notification": {
       "attrs": [],
       "http": {
         "url": "http://quantumleap:8668/v2/notify"
       },
       "onlyChangedAttrs": true,
       "metadata": [
         "dateCreated",
         "dateModified",
         "timestamp",
         "TimeInstant"
```

```
      ]
    }
}'
```

FIWARE services and service paths should be familiar to those who know the basic concepts of FIWARE, but they are explained here briefly. FIWARE service is a multi-tenancy feature which ensures that entities, attributes and subscriptions inside one service are invisible to other services[9]. In our case, we used a service named 360video. Service path is a hierarchical scope provided by Orion Context Broker where entities can be divided to hierarchies[10].

We can make a subscription for all entities (with idPattern value ".*") because we are using fiware-service 360video that separates our data from other services. Empty 'attrs' list indicates that updates to all attributes are subscribed. However, it is not always convenient to send all the attributes if only few has updated, so we used 'onlyChangedAttrs' field to pass only the changed attributes to QuantumLeap.

3.5 Data Model

Data is managed as context entities in FIWARE. Entities can represent various physical or logical objects such as devices, vehicles, weather observations and buildings. Each entity has an id and type which together uniquely identifies the entity. A FIWARE data model for a certain domain is defined by specifying the required entity types and their attributes which includes attribute names and types. The entity data is represented as JSON and the data model is documented by defining a JSON schema for the entity types and writing a human readable markdown document describing the entities and their attributes.

FIWARE provides ready-made data models that are designed for data portability[11]. There are data models available for domains such as Smart Cities, Smart Agrifood, Smart Environment, Smart Sensoring, Smart Enery, Smart Water, etc. However, we did not find an useful data model for our 360-degree video user data so we had to make up a new one. However, IoT Agent assumes that entities sending data are devices which makes data modelling a bit confusing. In our case, we add entities that are not clear devices. For example, view orientation is not a device but it is formed with values that originate from smart phone sensors.

In short, our data model consists of two different kind of entities: '360-degree video' and 'view session'. The first one simply gathers static attributes related to a 360-degree video such as URL and filename. The second one contains view orientation in yaw and pitch format, video time, and has a reference to the 360-degree video that is watched during the view session. A view session is identified by an entity id (also known as device_id when using IoT Agent)

[9] https://fiware-orion.readthedocs.io/en/master/user/multitenancy/index.html
[10] https://fiware-orion.readthedocs.io/en/master/user/service_path/index.html
[11] https://www.fiware.org/developers/data-models/

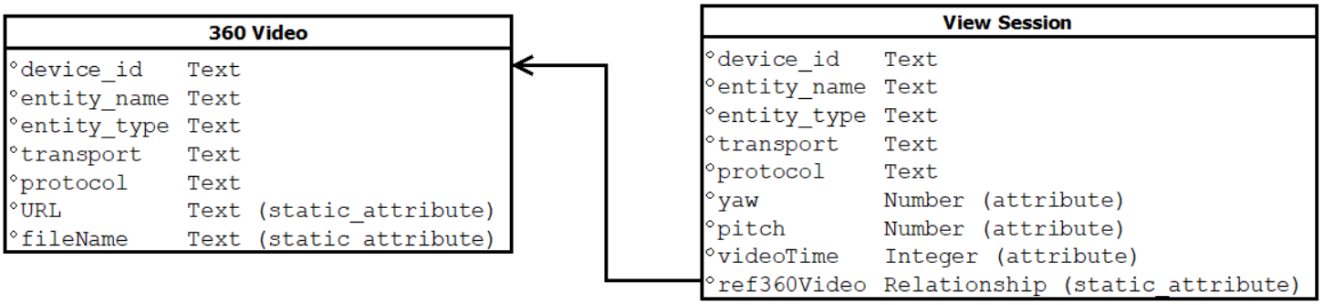

Figure 5: Data model used for creating entities via IoT Agent.

and the combination of yaw and pitch forms the view orientation at a certain moment in video time.

The same information is summarized in Figure 5. In addition, the Figure shows all the attributes that were used when provisioning devices via IoT Agent. In 360 Video, URL and fileName are attributes that are not directly provided by the device model used by the IoT Agent so they are defined as static_attributes. The concept of static_attributes is explained more in detail later in this report. In contrast, the basic device attributes device_id, entity_name, entity_type, transport and protocol are directly in the device model which means, for example, that their data types do not need to be explicitly defined when provisioning new devices and their values cannot be updated via the chosen IoT protocol. In View Session, the attributes yaw, pitch, and videoTime are defined as active measurement attributes, and ref360Video is a relationship to 360-degree video defined as static_attribute. The relationship is also indicated with the arrow from ref360Video attribute to the device_id of 360-degree video. In addition, View Session contains the same basic attributes as 360 Video (device_id, entity_name, etc.).

Note that we did not especially concentrate on data modelling during this case. For example, 360-degree video could have more interesting and important attributes especially from extensibility perspective. We were more interested in trying MQTT in action and experimenting with visualizations despite that data modelling is an important part of using FIWARE.

An entity created via IoT Agent can be retrieved from Orion that stores data in MongoDB. MongoDB stores only the latest values for attributes. An entity of View Session as received from Orion API looks like this:

```
{
    "id": "Device:ViewSession001",
    "type": "Device",
    "TimeInstant": {
        "type": "DateTime",
        "value": "2020-04-13T11:27:23.00Z",
        "metadata": {}
```

```
    },
    "pitch": {
        "type": "Number",
        "value": -19.994644,
        "metadata": {
            "TimeInstant": {
                "type": "DateTime",
                "value": "2020-04-13T11:27:23.00Z"
            }
        }
    },
    "ref360Video": {
        "type": "Relationship",
        "value": "360Video",
        "metadata": {
            "TimeInstant": {
                "type": "DateTime",
                "value": "2020-04-13T11:27:23.00Z"
            }
        }
    },
    "videoTime": {
        "type": "Integer",
        "value": "33",
        "metadata": {
            "TimeInstant": {
                "type": "DateTime",
                "value": "2020-04-13T11:27:23.00Z"
            }
        }
    },
    "yaw": {
        "type": "Number",
        "value": 15.825549,
        "metadata": {
            "TimeInstant": {
                "type": "DateTime",
                "value": "2020-04-13T11:27:23.00Z"
            }
        }
    }
}
```


Orion stores only the latest view orientation value but we are more interested in all the view orientations within a view session. These values are stored in QuantumLeap which is our data source for visualizations. We do not directly

handle the data stored in Orion. View session data looks like this when it is retrieved from QuantumLeap API with a query that lists the three latest updates:


```
{
    "attributes": [
        {
            "attrName": "TimeInstant",
            "values": [
                "2020-04-09T08:20:45.000",
                "2020-04-09T08:20:45.000",
                "2020-04-09T08:20:45.000"
            ]
        },
        {
            "attrName": "pitch",
            "values": [
                -10.191946,
                -10.221946,
                -10.268016
            ]
        },
        {
            "attrName": "ref360Video",
            "values": [
                null,
                null,
                null
            ]
        },
        {
            "attrName": "videoTime",
            "values": [
                13530,
                13001,
                13032
            ]
        },
        {
            "attrName": "yaw",
            "values": [
                -1.1523688,
                -10.540424,
                -9.677847
            ]
        }
```

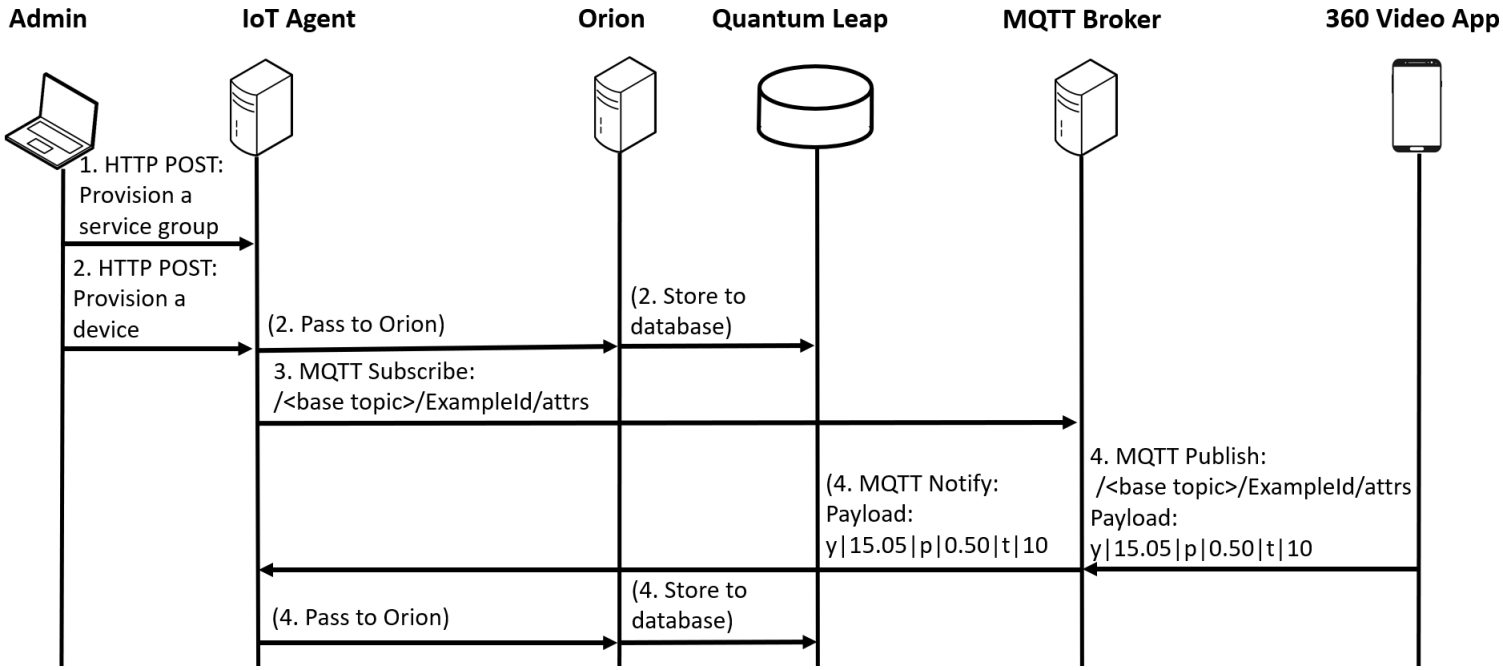

Figure 6: The steps for setting up devices communicating via MQTT.

```
    ],
    "entityId": "Device:ViewSession001",
    "index": [
        "2020-04-09T08:20:45.000",
        "2020-04-09T08:20:45.000",
        "2020-04-09T08:20:45.000"
    ]
}
```

4 Setting up MQTT for FIWARE

After the IoT Agent was installed and QuantumLeap subscription made, the following steps were required for setting up devices communicating via MQTT[12]. Figure 6 summarises the steps and presents a communication flow between the related parties. The operations in brackets are not explained in detail in Section 4 since they occur automatically and developer usually does not need to care about them. The payload used in MQTT messages is just an example. It could be any rational message in Ultralight format.

[12] https://github.com/FIWARE/tutorials.IoT-over-MQTT/tree/c1c27aa7d29a388001d62d0c51f2d8df66208123

1. **Provisioning a service group**. The idea of 'service group' is to create a top level MQTT topic for a group of devices related to the same service. A service group is provisioned by making an HTTP POST to IoT Agent Service API (<host>/iot/services). The payload of the POST request defines the base MQTT topic (field 'apikey') for a group of devices. Here is an example HTTP POST with command line tool curl:

```
curl -iX POST \
'https://<host>/iotagent-ul/iot/services' \
-H 'Content-Type: application/json' \
-H 'fiware-service: 360video' \
-H 'fiware-servicepath: /' \
-d '{ "services": [
        {
            "apikey": "cityiot-mqtt",
            "cbroker": "http://orion:1026",
            "entity_type": "Device",
            "resource": "/iot/d"
        }
    ]
}'
```

   The HTTP API for IoT Agent is described in the API documentation [13]. Note that we experienced problems when using the instructions given in FIWARE's MQTT with Ultralight tutorial (Footnote 12). The tutorial states that: "The resource attribute is left blank since HTTP communication is not being used". Our experience was that the resource attribute needs value "/iot/d/" or otherwise value updates with MQTT do not work. Another difference with the tutorial is that, in our experience, the entity_type attribute needs to have the same value as the devices have in their entity_type attribute when being provisioned. For example, if the service group entity_type is "Device", then all the things in that service group also need be of entity_type "Device". However, this needs further investigation and potential bugs could be fixed by modifying the open source code.

2. **Provisioning a device**: HTTP POST request to IoT Agent devices API (<host>/iot/devices), which contains information about a new device. In our case, 'device' is a vague concept meaning that any entity can be a 'device'. For example, the 'devices' we provision are 360-degree videos and view sessions of a 360-degree video. While 'view session' is not a real device, it has updating sensor values for yaw and pitch angles during video playback. Our 360-degree video has only static data so the HTTP POST for provisioning is not very interesting. In contrast, provisioning a view

[13] https://telefonicaiotiotagents.docs.apiary.io

orientation has more interesting details. Here is an example HTTP POST made with curl that adds an entity for 360-degree video view orientation:

```
curl -iX POST 'http://<host>/iot/devices' \
-H 'Content-Type: application/json' \
-H 'fiware-service: 360video' \
-H 'fiware-servicepath: /' \
-d '{ "devices": [
        {
            "device_id": "ViewSession001",
            "entity_type": "Device",
            "transport": "MQTT",
            "attributes": [
                {
                    "object_id": "y",
                    "name": "yaw",
                    "type": "Integer"
                },
                {
                    "object_id": "p",
                    "name": "pitch",
                    "type": "Integer"
                },
                {
                    "object_id": "t",
                    "name": "videoTime",
                    "type": "Integer"
                }
            ],
            "static_attributes": [
                {
                    "name":"ref360Video",
                    "type": "Relationship",
                    "value": "360VideoExample"
                }
            ]
        }
    ]
}'
```

The HTTP POST is elaborated in the following. In short, the base URL for IoT Agent is <host>/iot/. 'Devices' resource (<host>/iot/devices) is used to publish information to context broker via IoT Agent. There are two mandatory HTTP headers: fiware-service and fiware-servicepath. However, we did not use service paths as they are only used in HTTP requests and we concentrated on MQTT. The actual data is sent as a

JSON object that has an attribute named 'devices' which contains a list of devices to be provisioned.

3. **MQTT subscription**: in the earlier curl command, setting the value 'MQTT' for attribute 'transport' is enough for the IoT Agent to subscribe to the service group base topic (made in step 1) extended with the device id, for example, as follows /<api-key>/<device-id>. Note that FIWARE's MQTT topics start with an extra slash character which is not a recommended practice[14].

   The 'attributes' list contains attributes that are active readings from the device. It has also a mapping from abbreviated Ultralight 2.0 attributes to actual entity attributes. For example, an entity attribute named 'yaw' can be mapped to a single character 'y'.

   Static attributes can be defined with 'static_attributes'. The idea of static attributes is that their values cannot be changed via the chosen IoT protocol and they are initialized in the provisioning phase. In our example, we add only one static attribute that is a reference to the video being watched. In addition, lazy attributes can be defined but we did not use them. The idea of those is that IoT Agent can inform the device to update the measurements.

4. **Publishing an MQTT message**: attributes of the provisioned entity can be updated with an MQTT message. For example, the above view orientation entity can be updated by publishing a message to the base topic, provided when the service group was provisioned (step 1), that is catenated with the wanted device id and '/attrs' string, for example, /<base topic>/ExampleId/attrs. The payload of the message follows the Ultralight 2.0 format, for example, "y|15.05|p|0.50|t|1234" where yaw is 15.05 degrees, pitch is 0.50 degrees and videoTime is 1234 milliseconds of video time.

   IoT Agent subscribes, at least, to the following MQTT topic hierarchy in our example case. The topics that end with a single letter (y, p, t) can be used for updating a single attribute [15].

```
/cityiot-mqtt/ViewSession001/attrs
/cityiot-mqtt/ViewSession001/y
/cityiot-mqtt/ViewSession001/p
/cityiot-mqtt/ViewSession001/t
```

   The subscription from QuantumLeap to Orion can be tested with curl command that, for example, returns the last 5 updates for a certain entity:

```
curl -X GET \
```

[14] https://www.hivemq.com/blog/mqtt-essentials-part-5-mqtt-topics-best-practices/
[15] https://fiware-iotagent-ul.readthedocs.io/en/latest/usermanual/index.html

Figure 7: Yaw and pitch in real-world time.

```
-H 'fiware-service: 360video' \
-H 'fiware-servicepath: /' \
 'https://<host>/quantumleap/v2/entities/<entity_id>?lastN=5'
```

5 Example Visualizations with Grafana

This section discusses the Grafana visualizations and features that were used. We present the visualizations we made as Figures and remark other observations and experiences gathered while using the tool. The data in the visualizations is a result of an SQL query that is written individually for each visualization via Grafana UI. Grafana can access the data in CrateDB provided by QuantumLeap (as seen in Figure 3).

5.1 Graph

The graph visualization is a general purpose tool for visualizing data on timeline. FIWARE automatically generates a timestamp for logged data records which helps making timeline charts. However, timeline charts do not allow using other than real-world timestamp data on X axis. That makes analyzing user data of non-streaming videos difficult since the video time is often more important than the moment of time when the video was being watched. On the other hand, for streaming videos real-world timestamps can be useful because streaming fits better in real-world time.

Figure 7 shows an example how the visualization could work with a streamed video. The example has only one user trace but naturally there could be more simultaneous users. Graph does not support getting more information about the data point by hovering or clicking with mouse.

Graph tool also provides histograms which can show counted data. In 360-degree video user analysis, this feature can be used, for example, to see in which rotation angle the users have watched the most as in Figure 8. While the

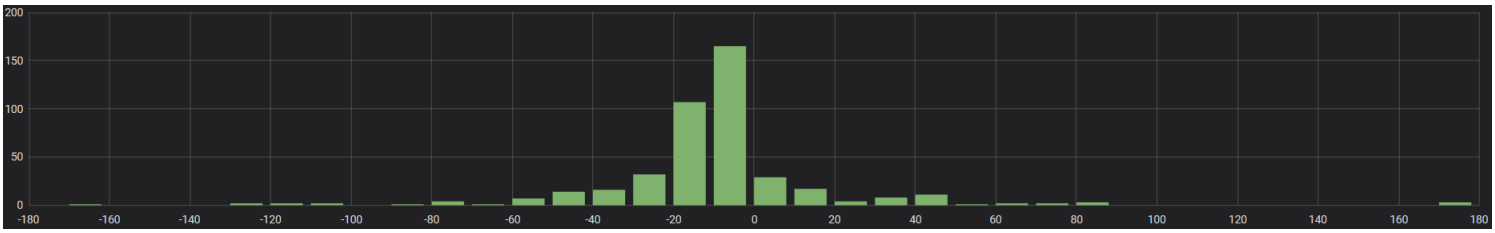

Figure 8: Yaw angles as histogram.

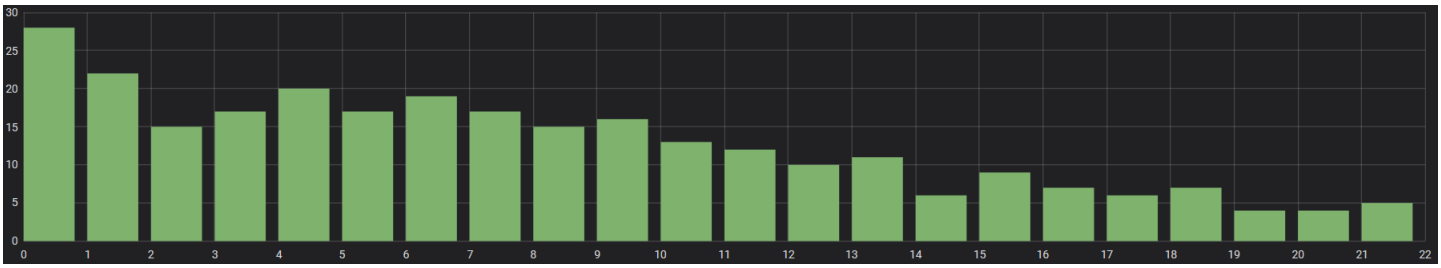

Figure 9: Most watched seconds of the video.

Figure presents yaw, the same graph can be naturally used for pitch. In another histogram example, we used a histogram to illustrate the most watched seconds of the video 9.

5.2 Plotly

Plotly plugin is not installed by default in Grafana, but in our experience, once installed it works well. Plotly allows using any data on X axis which makes it useful for making visualizations that are not dependent on real-world time.

Figure 10 presents a visualization, made with Plotly, that shows all the recorded view orientations within a single video. With the visualization, it is possible to get an overall impression where multiple users have watched during video playback. On X axis there is video time in milliseconds, and both the yaw and pitch are on Y axis in degrees.

Figure 11 presents a visualization of a single-user view session trace. The wanted view session can be selected using a drop-down menu on top of the dashboard.

Plotly[16] supports showing additional information about the data point with a hover tooltip. Only one field for additional information is available, so adding more information to the tooltip requires using string functions, such as concat, in SQL query.

[16] https://grafana.com/grafana/plugins/natel-plotly-panel

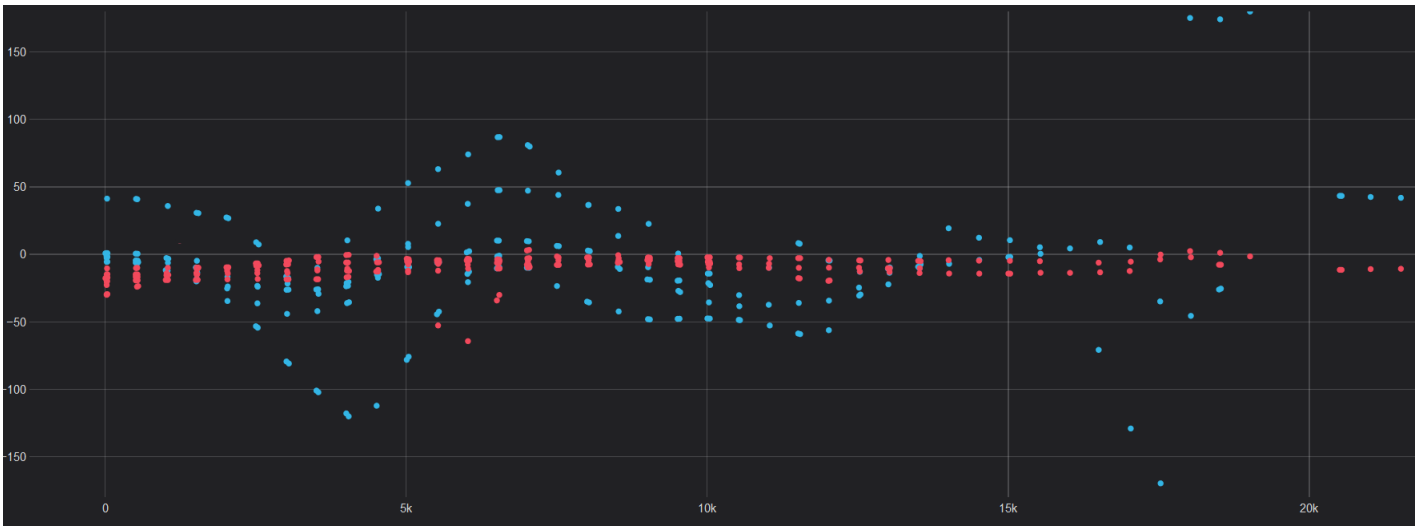

Figure 10: All the data from all the view sessions for a single video. Blue is for yaw and red is for pitch.

5.3 Table

The pros of Table visualization include a clear and precise view on the data. It can also show both video time and real-world time which tells when the certain moment of the video has been watched. However, it is not a good tool for getting an overall picture of the data. An example of Table visualization can be seen in Figure 12. Nothing prevents adding more columns to the view.

5.4 Radar and Windrose

We thought that rotation data would fit well with circular graphs such as those provided by Radar[17] and Windrose[18] Grafana plugins. However, it turned out that those plugins were relatively unstable. For example, making a query with an aggregate function seems to crash Windrose plugin. However, to grasp the idea, Figure 13 shows an example graph with Windrose plugin. While the information it shows is not particularly rational, it is still an example of special type of visualization. Windrose should be useful, for example, with weather data.

5.5 Drop-down Menus

Grafana allows users to create variables for which a value can be chosen with a drop-down menu[19]. A chosen value is then used in SQL queries that generate

[17] https://grafana.com/grafana/plugins/snuids-radar-panel
[18] https://grafana.com/grafana/plugins/fatcloud-windrose-panel
[19] https://grafana.com/docs/grafana/latest/reference/templating/

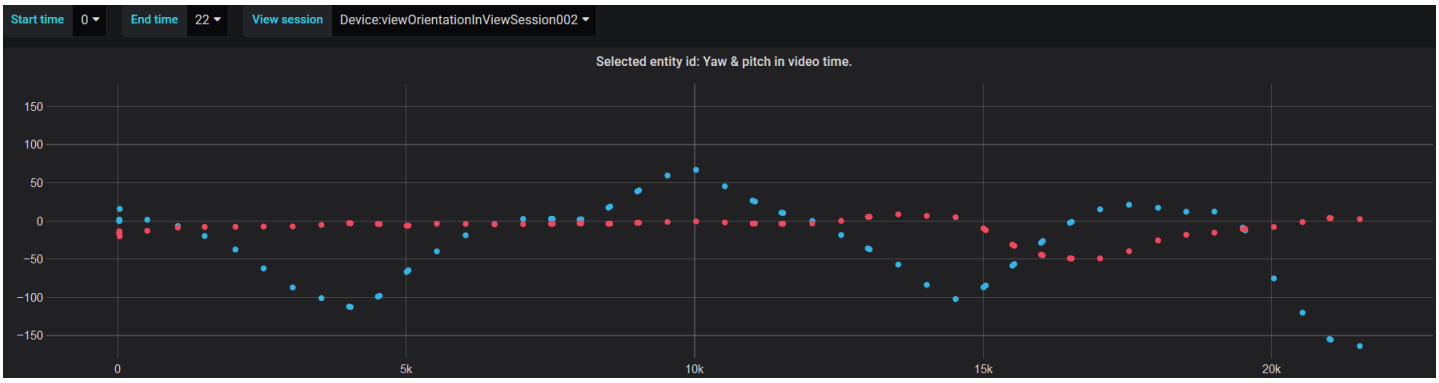

Figure 11: Selected view session.

Time	yaw	pitch	videotime
2020-04-09 11:20:45	-10.540424	-10.221946	13001
2020-04-09 11:20:44	-24.613462	-9.848647	12520
2020-04-09 11:20:43	-35.94802	-9.932204	11520
2020-04-09 11:20:42	-35.554394	-9.719233	10027
2020-04-09 11:20:41	-9.516491	-2.3394763	9017

Figure 12: Raw data in table format.

the graphs. That helps creating more dynamic graphs. Drop-down menus can be seen in top of Figure 11. The Figure has three drop-down menus: Start time, End time, and View session.

For example, we made a variable called VideoStartTime that can be used to set the video start time to the wanted value. There are different kind of variables available in Grafana but we used only 'query' typed variables. Different databases naturally have different kind of queries. For example, Grafana has instructions[20] for making variables with PostgreSQL. In our case, an example query for a variable can be:

```
SELECT videoTime / 1000
FROM mt360video.etdevice
ORDER BY 1
```

The query is elaborated in the following:

- videoTime is divided by 1000 because it is stored as milliseconds. Using milliseconds in a drop-down menu would be inconvenient. The resulting videoTimes are rounded nicely to integers.
- QuantumLeap (Figure 3) makes a table for every FIWARE service and entity type pair where the used table name is

[20] https://grafana.com/docs/grafana/latest/features/datasources/postgres/#query-variable

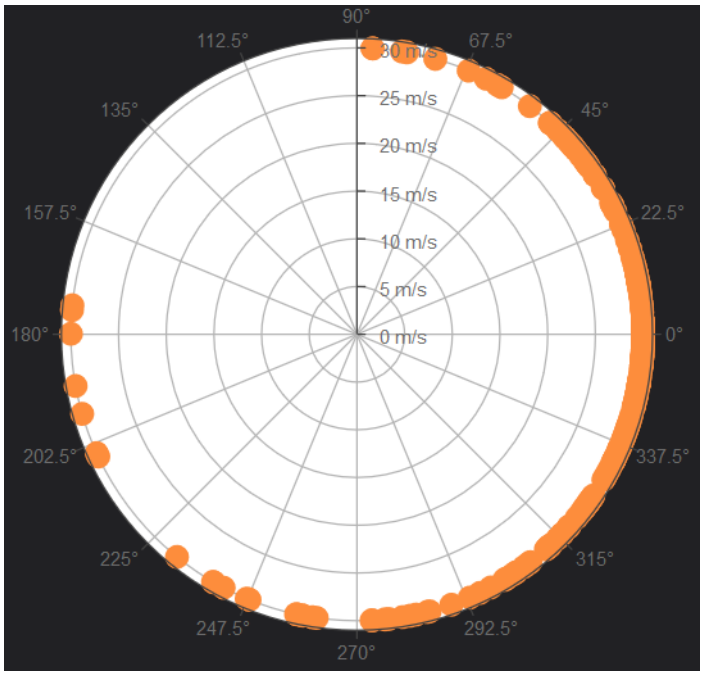

Figure 13: Yaw experimented in a wind rose.

mt<fiware_service>.et<entity_type>

- The result needs to be ordered to be user friendly in a drop-down menu.

The variable can then be used in SQL queries for the graphs in the following way:

```
SELECT yaw, pitch, videoTime, entity_Id
FROM mt360video.etdevice
WHERE entity_Id = '$entity_Id'
AND videoTime BETWEEN '$VideoStartTime' * 1000
AND '$VideoEndTime' * 1000
```

The variables are used in the queries by adding a dollar sign to the beginning of the variable name and wrapping the string with single quotation marks resulting in, for example, '$VideoStartTime'. The variables for video start and end times need to be multiplied by 1000 to convert seconds back to milliseconds. The resulting visualization, with video time set between 0 to 22 seconds, can be seen in Figure 11.

6 Open Questions

In the described case, we mostly concentrated on using IoT Agent with MQTT and making visualizations with Grafana. However, there are many aspects to be considered when creating a real implementation.

We were a bit cautious and sent data only twice in a second. A faster updating pace can be needed in some applications and the performance should be tested with multiple simultaneous users. Scaling of the system should improve the performance if needed. Araujo et al.[21] have studied scaling strategies of FIWARE but it must be noted that scaling has limitations as well.

Information security was not taken properly into account. However, MQTT has authentication and authorization but, for example, payload encryption could be difficult with IoT Agent. Naturally, this could be fixed by modifying the source code since it is open source. In addition, the general security aspects of Tampere University FIWARE platform has been discussed in [22].

When provisioning sensors (Section 4, step 2), a device id must be defined and generated. Since there is not a service for creating ids, one must be careful not to generate clashing ids. This could be a problem in a multi-user case. In this experiment, we had only a handful of devices so id management was easy enough to handle manually.

Concerning the multi-user challenges, the visualizations can get messy if there are a lot of overlapping traces, similarly to Figure 10. Drop-down menus can be used for limiting the shown information but a more user-friendly interaction would be nice. For example, it would be nice to be able to select (one or more) traces directly from the visualization by clicking with mouse.

In this case, we used only a single 22 seconds long video. For longer videos user experience will need to be considered better. For example, a drop-down menu for every second in video that is five minutes long would be an annoying thing to use, in addition to continuously manually calculating, for example, what is four minutes and 22 seconds as seconds (262 s).

Since we experimented with a single video, we did not experience the difficulties of visualizing multiple videos. Neither we did make comparisons between two videos. However, making such visualization should be basic usage of Grafana for those who know how to use it.

7 Conclusions

In general, the case was successful, using Tampere University FIWARE platform was relatively easy, and making visualizations with Grafana was mostly enjoyable. We managed to implement many useful visualizations with relatively low effort. The biggest challenges were related to some inconsistencies with FIWARE documentation as described in Section 4 (step 1) and to IoT Agent

[21] Araujo, Victor, et al. "Performance evaluation of FIWARE: A cloud-based IoT platform for smart cities." Journal of Parallel and Distributed Computing 132 (2019): 250-261.

[22] https://drive.google.com/file/d/17tg4hlTv4r1PaOtnmPW8HMfPBxpcPI5N/view

assuming device data model for the gathered data. In addition, it was a slight disappointment that Radar and Windrose plugins for Grafana were too unstable for making anything useful.

While we mostly discuss non-streaming videos, the approach explained in this paper could be more convenient with streaming videos since many smart city applications require streaming video. Using alerts in Grafana could be useful as well for automating parts of the analysis.

In addition to user analysis, many applications would benefit from analyzing the video content. For example, by using object detection algorithms it is possible to identify potential points of interest from the video such as, for example, cars or persons. Such data could be combined with the log data so that a log records tell whether a user had a car in their field of view. With such data, it would be possible to produce even more exiting visualizations.