Appendix E: Database Integration

Contents

Abstract

This Appendix presents the design of the prototype data management platform that control the data collection from the sensors. The main functional blocks of the system are identified and discussed. Then, the implementation of the prototype platform will be described on both the on-premises and the cloud-based approaches. Results and demonstrations are also given for further data estimation and planning.

1 Introduction

In a Smart City scenario, as the number of sensors increases, the volume of data becomes overwhelming. In addition, as there are many types of sensors from many manufacturers coexist in the same network, interpreting the collected data is hard as the raw data formatting is different for each of them. Although many IoT platforms are commercially available, significant effort will still be required to configure and test them with multi-vendor, multi-communication type IoT devices, as some of the platforms might focus on certain communication interface type (for example based on cellular network, like CISCO-Jasper or Ericsson IoT Accelerator), or certain data type (for example: streaming high BW video devices or small BW but massive amount of sensors like garbage can, or car parking detector). As a result, a flexible, adaptable and customizable centralized data management structure is one of the most important components in a Smart City setup to prevent service/device provider lock-in. The basic functionalities of this platform consist of four main services as illustrated below:

- **Data Aggregation** is responsible for establishing connections to the devices and sensors for collecting sensing data as well as acting as a proxy between the IoT network and the external network to receive and forward reconfiguration commands to the devices.
- **Data Format Translation**. In the context of a Smart City, there may be many types of devices from different manufacturers. Each of the device may produce raw data in different formats and hence, uniformly format the data before storage is needed for the ease of processing the data latter on.
- **Database and Device Log** receive and archive sensing data from the IoT network. For resilience purpose, the content may be distributed across many nodes in a redundancy manner.
- **Data visualization** represents collected data in terms of charts or maps for the ease of management.

To facilitate this functionalities, two main approaches can be identified based on the placement of the data management platform: **On-premises** and **Cloud-based approach**. In the On-premise approach, all the above functionalities are hosted on local computing facilities such as a data center. In this case, the operational, administration and maintenance of these facilities will be on the shoulder of the data center's operator. This approach offer the benefits of performance and data privacy. Since the data sink is placed close to its sources, the performance of data archiving in terms of bandwidth and delay can be guaranteed easily. This is especially advantageous for real-time data. In addition, since the data is stored locally, the operator has the complete control over the data, and its privacy. Privacy control is very important in the context of Smart City where thousands of video streams of citizens are recorded and stored. However, it can be costly to setup, maintain and scale up this back-end infrastructure to support the growing Smart City IoT network and applications.

In Cloud-based approach, many of these functionalities are designated to the cloud computing facility. Cloud Computing brings about the availability of virtually unlimited storage, processing capacity and computing resources at low cost that can be virtualized and leased to users on demand. To catch up with the trend, many cloud providers such as Amazon, IBM, and Microsoft are racing to adopt their Cloud Computing platforms for delivering various IoT services over the Internet. Many motivations that drive the development of IoT and Cloud Computing integration are identified as below.

- **Interoperability:** Generally, IoT comprises of large number of heterogeneous devices and technologies that often needs to interoperate in seamless manner with each other as well as back-end servers. To cope with this problem, Cloud Computing often offers a unified platform to aggregate data and communication with IoT devices, which greatly simplifies the integration of new devices and streamlines data flows between IoT network and back-end management components.
- **Scalable and rapid deployment:** It is often not easy to scale up an on-premises setup. To increase storage and computing capacity, new physical servers have to be purchased and integrated into the existing system, a lengthy and potentially costly process. Resources have to be allocated for peak performance demands even if most of the time they stay idle. Cloud Computing platform enables

rapid resource allocation and integration through well-defined APIs and automated processes, which can cope well with the dynamic demands of the IoT network.

- **Ubiquitous access:** Cloud Computing platform allows IoT applications to be accessed anywhere and anytime through web portals or built-in apps. It enables effective and cheap solution to connect, track, and manage IoT devices, data, back-end operations from anywhere at anytime.

In this appendix, both the On-premises and the Cloud-based approaches will be investigated for the data management platform. For Cloud-based approach, Microsoft Azure Cloud will be used to implement upon as it is arguably one of the best cloud platform available on the market. In each case, the placement and implementation details of each of the main functionalities will be identified.

2 On-Premises data management

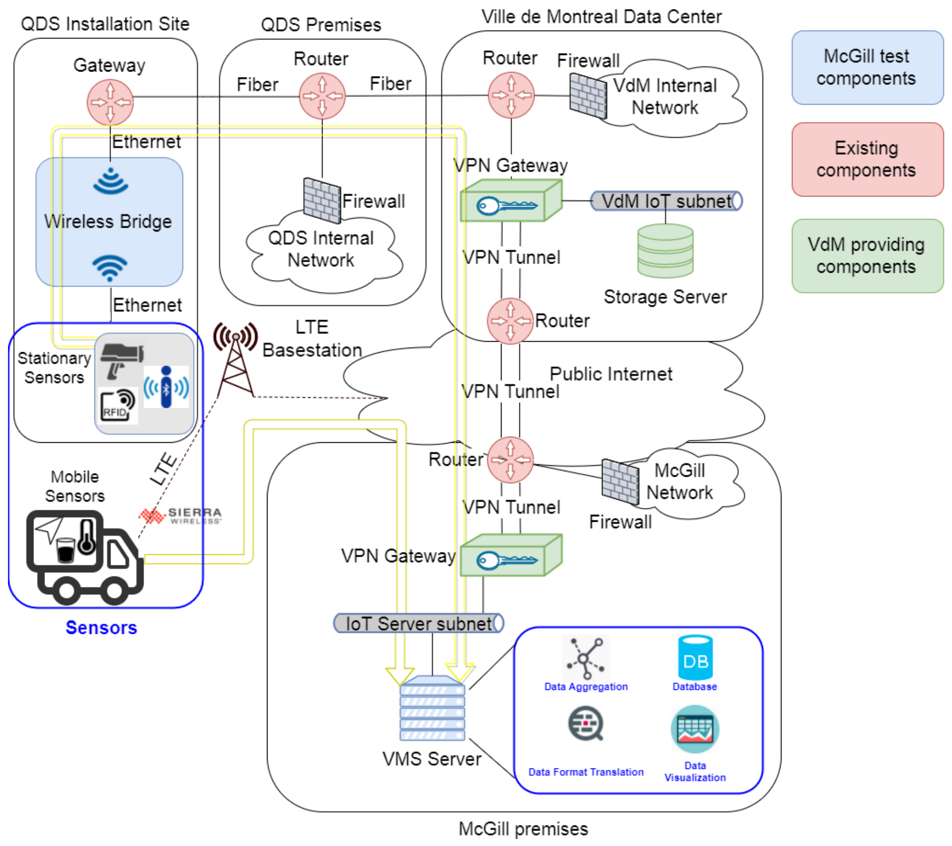


Figure 1: On-premises data management architecture.

Figure 1 illustrates the data flows and the four fundamental components of the on-premises data management platform: data aggregation; data format translation; the data base and the data visualizer. The *data aggregation* is in charge of connection establishment and raw data collection from different types of sensors in the network. *Data format translation* is responsible for parsing the data, reformatting them into a unified format and connecting to the database. The *database* stores the collected data for future processing. The *data visualization* provides a mean for users to visualize the collected data from the database. In the on-premises implementation, all of these components are deployed in a management server.

2.1 Data Aggregation

Due to the wide range of setups and types of sensors, sensor data are collected in one of the three methods as shown in Figure 2. Stationary devices, in particular traffic radars and RFID-UHF readers, have a static IP address, so TCP server is used for data collection because connection to these can be done from many destinations simultaneously using TCP clients. Mobile devices such as temperature and ultrasonic level sensors cannot use the same approach because their IP addresses are constantly changed. As a result, those sensors send data through their TCP clients. These devices can only send data to one configured device at a time. In addition, Sierra Ailink's GPS sensor does not support TCP client mode and its TCP server mode doesn't work (due to firmware bugs), so in our prototype platform, the GPS send data through UDP server sockets.

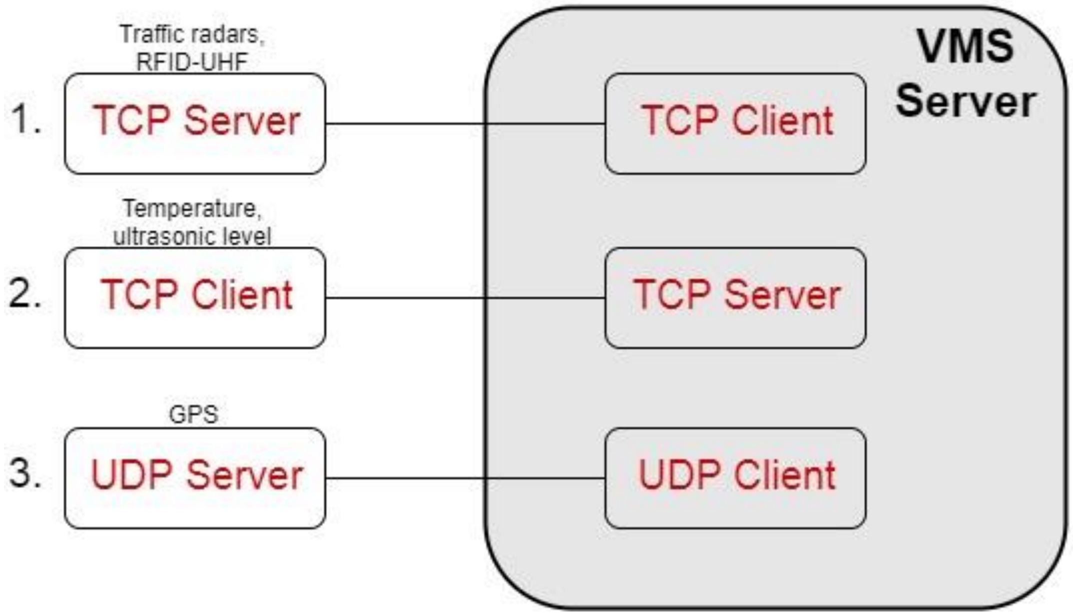


Figure 2: Types of socket connections.

2.2 Data Format Translation

2.1.2 Raw data format interpretation

Stationary sensors

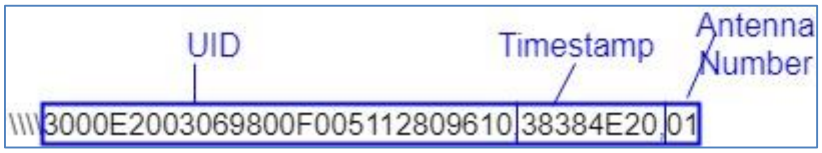
- **Radar sensors.** Table 1 shows the raw data format of messages from radar sensors. Basically, there are four types of messages in which two are used for real-time and two are used for statistics purposes. It is noted that the speed values in the reports are scaled up by 10 so scaling down by 10 should be done to obtain the actual speed.

Table 1: Formatting of traffic radar reports.

Message type	Message formats
Target detected report	***$RDTGT,D1,S1,L1,D2,S2,L2,…,Dn,Sn,Ln*CSUM<CR><LF>*** ***Meaning:*** This message type is use for real-time report and includes information about all the detected targets in a pre-defined period (currently set at 1s). ***D_i:*** The direction of the ith target (1 approaching, -1 receding). ***S_i:*** The speed of the ith detected target (Note: speed = S_i/10). ***L_i:*** The detected level of the signal reflection from the target. ***CSUM:*** message checksum.
Target counts report	***$RDCNT,D,S,L,aprCNT,rcdCNT*CSUM<CR><LF>*** Meaning: This message type is sent whenever a new valid target is detected.

	D: The direction for the new counted target (1 approaching, -1 receding). ***S:*** The speed for the new counted target (speed = S/10). ***L:*** The detected level of the signal reflection from the target. ***aprCNT:*** The cumulative counter for the approaching targets. ***rcdCNT:*** The cumulative counter for the receding targets. ***CSUM:*** Message check sum.
Approaching target statistics report	***$RDSTA,count,avgSpeed,minSpeed,maxSpeed,roadOCP,tmpCNT*CSUM<CR><LF>*** **Meaning:** The message is sent periodically at the beginning of each statistics report (currently set at 15 minutes) for ***approaching*** targets. *All values are relative to the time period from the last sent report*. ***count:*** The accumulated count during the report period. ***avgSpeed:*** The average speed for all approaching targets in the report period. ***minSpeed:*** The minimum of all approaching targets in the report period. ***maxSpeed:*** The maximum of all approaching targets in the report period. ***roadOCP:*** The road occupation percentage for the report period calculated as number of samplings with at least one valid approaching detected target divided by the total number of samplings in the report period. ***tmpCNT:*** The temporary counter of all approaching targets in the report period. ***CSUM:*** Message check sum.
Receding target statistics report	***$RDSTR,count,avgSpeed,minSpeed,maxSpeed,roadOCP,tmpCNT*CSUM<CR><LF>*** ***Meaning:*** The message is sent periodically at the beginning of each statistics report (currently set at 15 minutes) for ***receding*** targets. *All values are relative to the time period from the last sent report*. ***count:*** The accumulated count during the report period. ***avgSpeed:*** The average speed for all approaching targets in the report period. ***minSpeed:*** The minimum of all approaching targets in the report period. ***maxSpeed:*** The maximum of all approaching targets in the report period. ***roadOCP:*** The road occupation percentage for the report period calculated as number of samplings with at least one valid approaching detected target divided by the total number of samplings in the report period. tmpCNT: The temporary counter of all receding targets in the defined time period. CSUM: Message check sum.

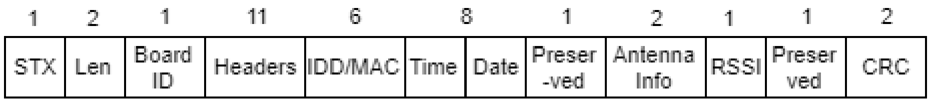
- **RFID-UHF readers.** As shown in Figure 3, the raw data from RFID-UHF readers consists of three values, separated by commas. The first value is the UID, second value is the timestamp and third is the antenna number. The timestamp is in a hex format. The first two bytes, converted to decimal represent the hour, the next two the minutes and the next four bytes represent the milliseconds. For example, 003901F4 in hex evaluates to 0hr:57min:0sec:500ms. Due to a firmware bug which causes falsified data, the current timestamp, as seen in Figure 3 is not valid. As a result, the timestamp values from the RFID-UHF readers in the database are automatically set to having a 'bad' status until it is fixed.



```
\\\3000E2003069800F005112809610,38384E20,01
```

Figure 3: Raw data from RFID-UHF readers.

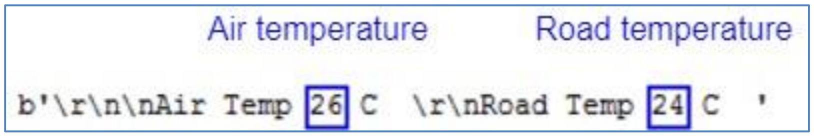
- **RFID-BLE readers.** As shown in Figure 4, the raw data from RFID-BLE reader is rather long and consists of many fields. Among the fields, it is important to point out the most important fields that are relevant to the applications in MSCPS are: Board ID (use to identify readers in a wide scale setup), IDD/MAC (ID of the BLE tag), Date and Time and RSSI (received signal strength).



1	2	1	11	6	8	1	2	1	1	2
STX	Len	Board ID	Headers	IDD/MAC	Time	Date	Preser-ved	Antenna Info	RSSI	Preserved	CRC

Figure 4: Raw data from FRID-BLE reader.

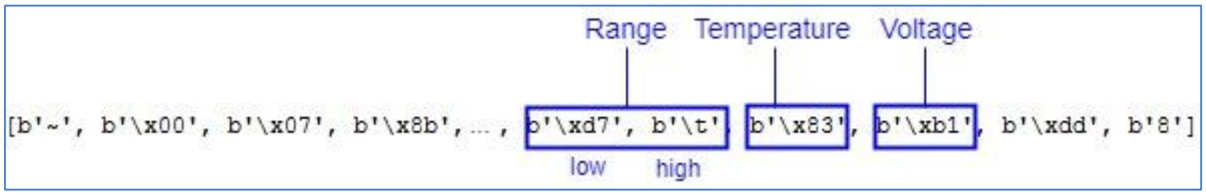
Mobile sensors



```
b'\r\n\nAir Temp 26 C  \r\nRoad Temp 24 C  '
```

Figure 5: Raw data from temperature sensors.

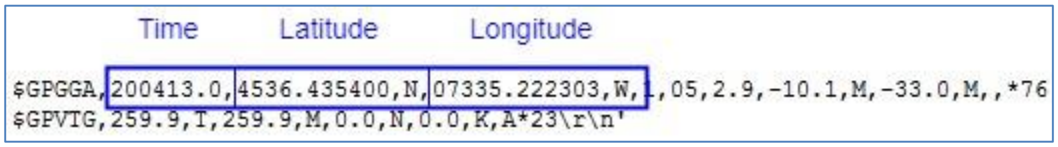
- **Temperature Sensors.** Figure 5 illustrates the raw data from the temperature sensors including the ambient temperature and the road surface temperature data.



```
[b'~', b'\x00', b'\x07', b'\x8b', ... , b'\xd7', b'\t', b'\x83', b'\xb1', b'\xdd', b'8']
```

Figure 6: Raw data from Ultrasonic Level Sensors.

- **Ultrasonic Level Sensors.** The raw data sent from Ultrasonic sensors are encapsulated in a frame of 40 bytes. The raw data are 4 bytes from the sixth byte to the third byte from the end of the frame as illustrate in Figure 6. The sixth and fifth bytes from the end of the frame represents the range which are used to calculate the distance from the sensor to the salt surface. The actual distance in inches is calculated from the range (converted to decimal) and divided by 128, then subtracted by 28 (the sensor is placed 28 inches above the bottom of the salt trunk). The temperature reading in degree Celsius can be calculated from the value of the forth byte from the end (convert to decimal) by multiplying by 0.587085 and subtracted by 50. The voltage value is the third byte from the end, identifies the battery voltage and can be calculated by converting to decimal, subtracting by 14 and dividing by 40.



```
$GPGGA,200413.0,4536.435400,N,07335.222303,W,1,05,2.9,-10.1,M,-33.0,M,,*76
$GPVTG,259.9,T,259.9,M,0.0,N,0.0,K,A*23\r\n'
```

Figure 7: Raw data from GPS sensors.

- **GPS sensors.** The raw data from GPS sensors is shown in Figure 7, including the timestamp in GMT, the latitude and the longtitude.

2.1.2 Unified data format

In this project, in order to unify the data formatting for the ease of access, the data format proposed by VdM was adopted. The collected data are stored in JSON files with the following values: *deviceID* and *Blocks*.

- **deviceID**. Each JSON file only stored data for a single device, identified by the deviceID
- **Blocks** value is a list of dictionaries. Each of those dictionaries corresponds to a specific value or a set of values. There is information about the format, the description of the values, the creation date-timestamp, the expiry date-timestamp, the unit and status of the value, and the value itself.

- **Format** is ODNGF1 (Open Data Node Format Version 1) corresponds to the version of the unified data format.
- **Desc** describes the data presented.
- **CreateUtc** is the creation time and date of the data. The timestamp is created when the data is received at the management server.
- **ExpiredUtc** is the expiration time of the data. Currently, “0000-00-00T00:00:00” is used, which indicated that the data is always valid.
- **Unit** helps with the interpreting of the **Value**. If there are multiple values, the unit would be “array”. If the multiple values are of different types, the unit would be “object”.
- **Status** is used to show if the data would be falsified. It could be “bad”, “uncertain” or “good”. The status is left empty for now as there is not yet a well-defined procedure for determining this.

2.3 Database

The database to store collected data is deployed using MongoDB as it is open-source, scalable and is capable of distributed database implementation. The database is organized into four collections as the followings:

RadarRT Collection: storing real-time reports from the traffic radar (Target detected and target count reports). The collection consists of JSON documents, each contains up to 1000 blocks for the ease of data access as well as management. The document name is defined as: *<name of the radar>+”_rt”*, ex. *traf_rad_001_rt*.

An illustrative example of one document is as the following:

```
{
        “deviceID” : “traf_rad_001”
        “Blocks” : [
                {                                               // block 1
                        “Format”: “ODNF1”
                        “Desc”: “Detected targets”              // Target detected report
                        “CreateUtc”:  “2017-05-17T10:58:23”
                        “ExpireUtc”:  “2017-05-17T10:58:53”     //If the data is valid at all times, the
                                                                value will be “0000-00-00T00:00:00”
                        “Unit”: “array”
                        “Status”: “good”                        //only good reports will be saved
                        “Value”: [                              //The values of each target
                                {“Dir1” : ” ” “Spd1” : ” ” “SS1” : “ ”}
                                {“Dir2” : ” ” “Spd2” : ” ” “SS2” : “ ”}
                                {“Dirn” : ” ” “Spdn” : ” ” “SSn” : “ ”}
                        ]
                },
                {                                               // block 2
                        “Format”: “ODNF1”
                        “Desc”: “New target”                    // Target count report
                        “CreateUtc”:  “2017-05-17T10:58:23”
                        “ExpireUtc”:  “2017-05-17T10:58:53”     //If the data is valid at all times, the
                                                                value will be “0000-00-00T00:00:00”
                        "Unit": "object"
                        "Status": "good"                //only good reports will be saved
                        "Value": {
                                "Dir" : " "
                                "Spd" : " "
                                "SS" : " "
                                "aprCNT" : " "
                                "rcdCNT" : " "
                        }
        }
```

RadarST Collection: storing statistics reports from the traffic radar (Approaching and Receding Target statistics reports). The collection consists of JSON documents. The document name is defined as: *<name of the radar>+"_STA_"+<date>*, ex. *Traf_Rad_001_STA_2017-05-17*.

An illustrative example of the block structure in this collection is as the following:

```
        {

                "Format": "ODNF1"
                "Desc": "Radar Statistic"               //Approaching and receding reports
                "CreateUtc":  "2017-05-17T10:58:23"
                "ExpireUtc":  "0000-00-00T00:00:00"     //data is valid at all times
                "Unit": "object"
                "Status": "good"                        //only good reports will be saved
                "Value": {
                        "aaSpd" : " "                   //aproaching average speed
                        "amSpd" : " "                   //apr. minimum speed
                        "aMSpd" : " "                   //apr. Maximum speed
                        "ardOCP" : " "                  //apr. road occupation percentage
                        "aprCNT" : " "                  //number of apr. targets in period
                        "raSpd" : " "                   //receding average speed
                        "rmSpd" : " "                   //rece. minimum speed
                        "rMSpd" : " "                   //rece. Maximum speed
                        "rrdOCP" : " "                  //rece. road occupation percentage
                        "rcdCNT" : " "                  //number of rece. targets in period
                        "period" : "15 min"
                }
        }
```

Salt_Truck Collection: storing collected data from the salt truck. The collection consists of JSON documents, each contains up to 60 blocks for the ease of data access as well as management. The document name is defined as: *<name of the truck>+<date and time>*, ex. *anjou_salt01_ 2017-05-1710:58:23*.

An illustrative example of the block structure in this collection is as the following:

```
        {
                "Format": "ODNF1"
```

```
                    "Desc": "Salt Truck data"
                    "CreateUtc":  "2017-05-17T10:58:23"
                    "ExpireUtc":  "0000-00-00T00:00:00"     //data is valid at all times
                    "Unit": "object"
                    "Status": "good"                        //only good reports will be saved
                    "Value": {
                            "Lat" : " "
                            "Lon" : " "
                            "airTemp" : " "
                            "rdTemp" : " "
                            "Level" : " "
                            "Batt" : " "
                    }
            }
```

RFID Collection: storing collected data from the RFID readers. The collection consists of JSON documents. The document name is the name of the server, ex. *sch6_1_sv01*.
An illustrative example of the block structure in this collection is as the following:

```
            {
                    "Format": "ODNF1"
                    "Desc": "RFID tags data"
                    "CreateUtc":  "2017-05-17T10:58:23"
                    "ExpireUtc":  "0000-00-00T00:00:00"     //data is valid at all times
                    "Unit": "object"
                    "Status": "good"                        //only good reports will be saved
                    "Value": {
                            "UID" : " "
                            "Timecode" : " "
                            "Antenna" : " "
                    }
            }
```

2.3 Data Visualization

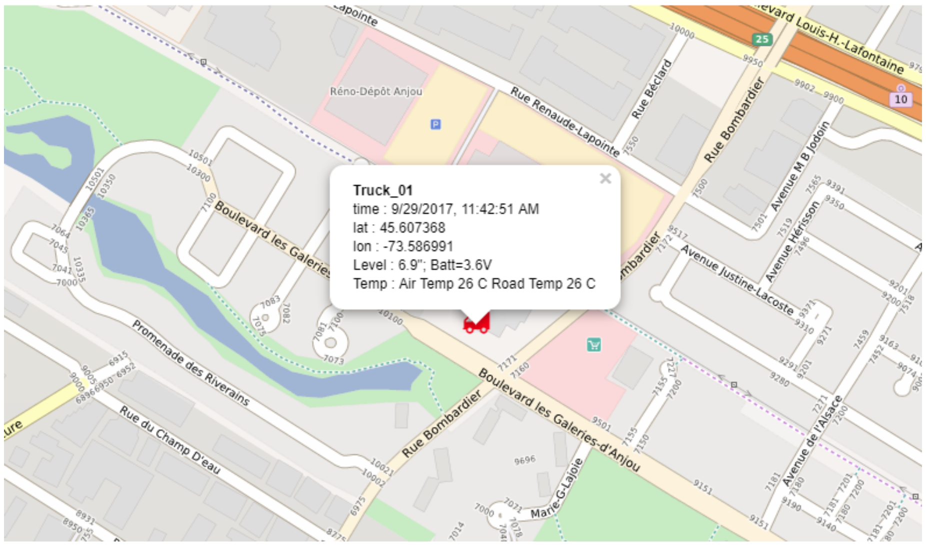

Truck_01
time : 9/29/2017, 11:42:51 AM
lat : 45.607368
lon : -73.586991
Level : 6.9"; Batt=3.6V
Temp : Air Temp 26 C Road Temp 26 C

Figure 8: Visualizing salt truck data.

For the salt truck, collected data can be visualizing in real-time by requesting the data directly from the Data Aggregation block and visualizing directly on Open Street Map. In this case, the location of the truck as well as the readings from its sensors can be displayed in real-time as illustrated in Figure 8. Moreover, data can also be queried from the database by scripts and graphing using different tools. For example, Figure 9 presents the histogram of vehicle speeds from a radar sensor.

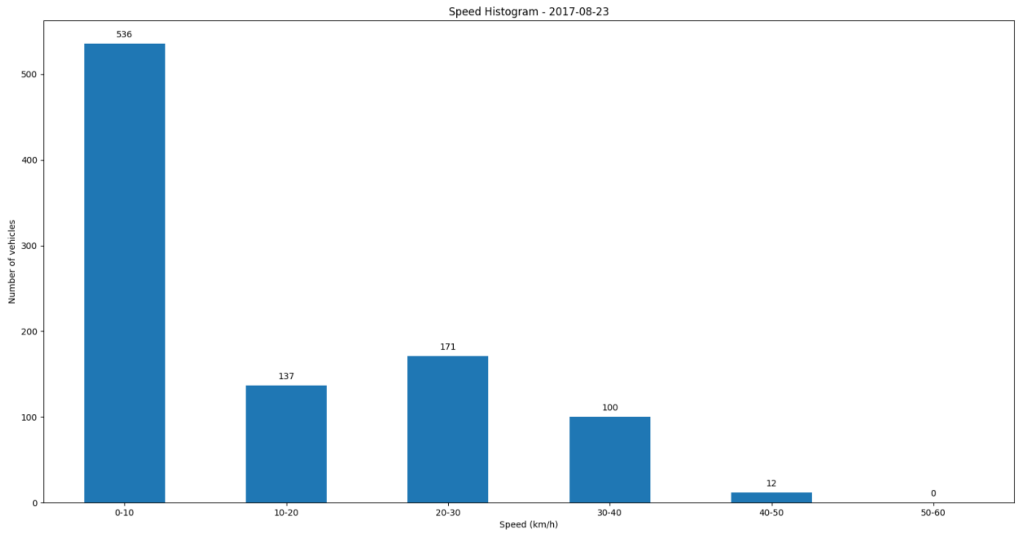


Figure 9: Speed histogram visualization from radar sensor data.

3 Cloud-based data management

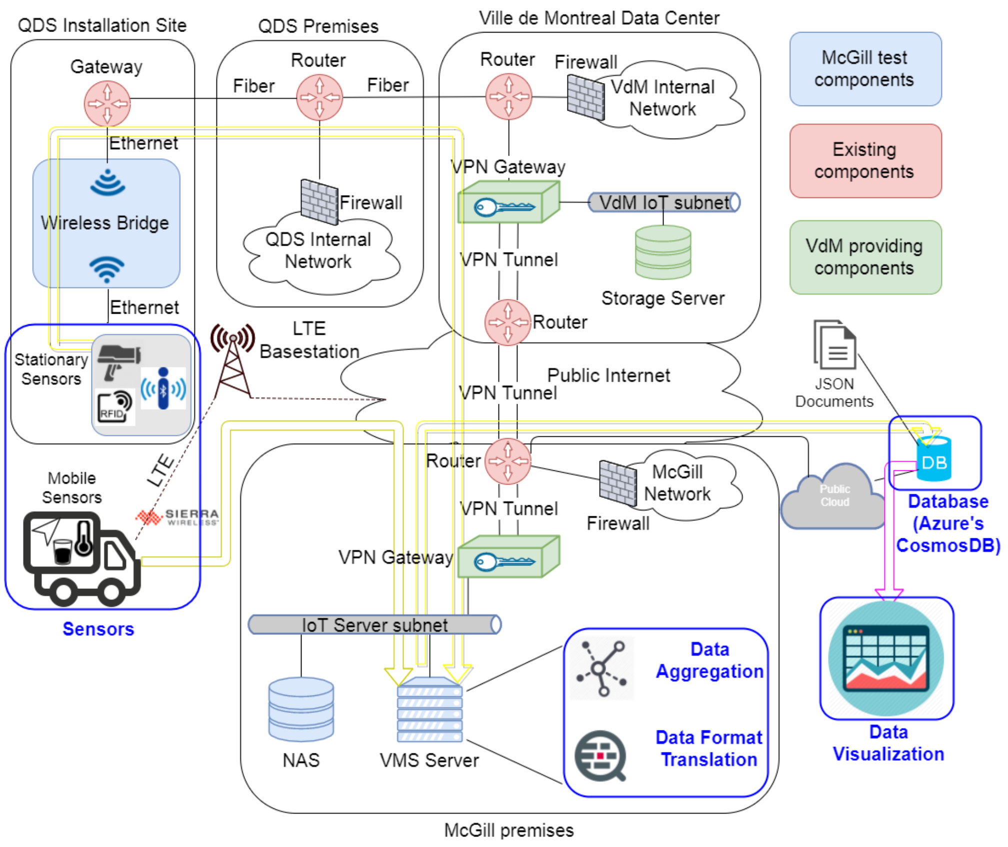


Figure 10: Cloud-based data management.

For cloud-based data management, as illustrated in Figure 10 the Database is moved to the Cloud (using Azure CosmosDB) so that Data Visualization can be done from any device with Internet connection. The Data Aggregation and Data Formatting Translation are retained on-premises. It is noted that these two functional blocks can be moved to the cloud; however, Data Aggregation is built on-premises due to security reasons (prevent direct access from the Internet to the sensors) and Data Formatting Translation functional blocks, which can be implemented in Azure's IoT hub, is implemented on-premises for the ease of implementation and because of the time limitation.

3.1 Data Aggregation and Data Format Translation

These two functional blocks are retained on-premises and hence, the implementation details are the same as the on-premises implementation.

3.2 Database

In the cloud setup, Azure CosmosDB is used. In order to store data to the database, a url and a key (simple string) are required and can be accessed from the 'keys' page in Azure. In the current setup, the database has unlimited storage and can store multiple JSON documents.
The database can be connected to using Python scripts along with a url and key variables. Those values can be found in the 'keys' pages of the collection next to the field "url" and "primary key" on Azure Portal page.
The Azure CosmosDB Database cost is based on a provisioned amount of Request Units (RUs) and the amount of storage used. Storage costs $0.33 CAD per GB per month ($167.19 per month for 500GB) [7]. The storage costs work as a pay-as-you-go system. RUs are Azure's custom unit for CPU, memory and

IOPS (input/output operations per second) usage. The collection has 2500 RUs/sec provisioned (lowest amount for a collection with unlimited storage) which costs $199.02 CAD per month. For the amount of radars that are currently live, that amount of RUs/sec is more than enough. Whenever all of the provided RUs in a second are used, then the execution would be delayed to the next second(s). Azure's CosmosDB has an upload speed of 1.51 MB/s from the East of Canada [8].

Table 2: Amount of data upload form different types of sensors.

Sensors	Consumption (kB/hour)	Incoming data rate	Writing to database rate
Radars	~20,000 (approximation)	20 sentences per second	Once every ~3 min
Temperature	1	Once every 15 min	Once every 15 min
Level	1	Once every 15 min	Once every 15 min
GPS	2	Once every 15 min	Once every 15 min

Table 3: Data retention prediction for all currently live sensors

Sensors	1 Day	1 Month	1 Year
8 radars	3.840 GB	115.2 GB	1.4016 TB
1 Temperature sensor	24 KB	720 KB	8.760 MB
1 Level sensor	24 KB	720 KB	8.760 MB
1 GPS sensor	48 KB	1.440 MB	17.520 MB

Since data storage is costly in Cloud-based setups, the data consumptions of different type of sensors are monitored. As illustrated in Table 2, traffic sensors have the most data recorded with approximately 1MB file being uploaded every 3 minutes. For data retention, Table 3 shows the required storage for all of the currently live sensors for 1 day, 1 month and 1 year.

3.3 Data Visualization

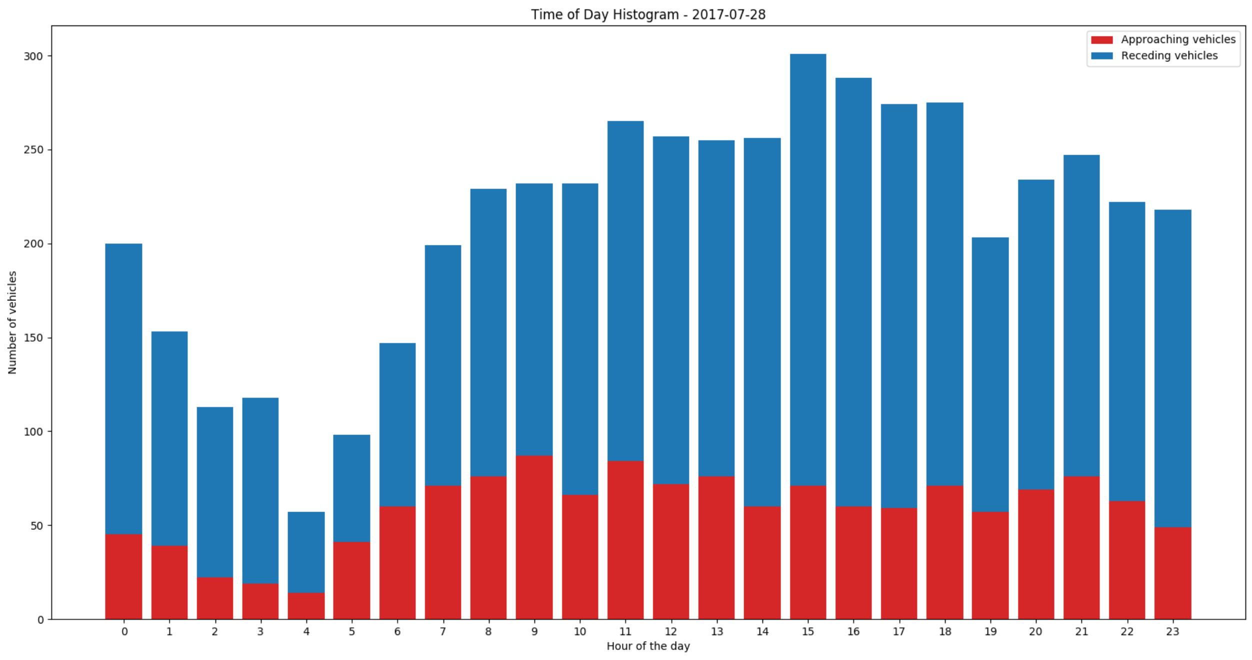


Figure 11: Number of approaching and receding vehicles at different time in a day.

For visualizing data to charts, a user-interactive program was implemented for traffic radar, temperature and ultrasonic level sensors. For Traffic radars two types of graphs can be created: A time-of-day histogram which shows the number of cars (receding and approaching) passing by in the different hours of the day as shown in Figure 11, and a speed histogram which shows the number of cars in different speed ranges. For temperature and ultrasonic level sensors, graphs that show the changes in the temperature or level value over the course of the day can be produced. All graphs are saved as .png files.

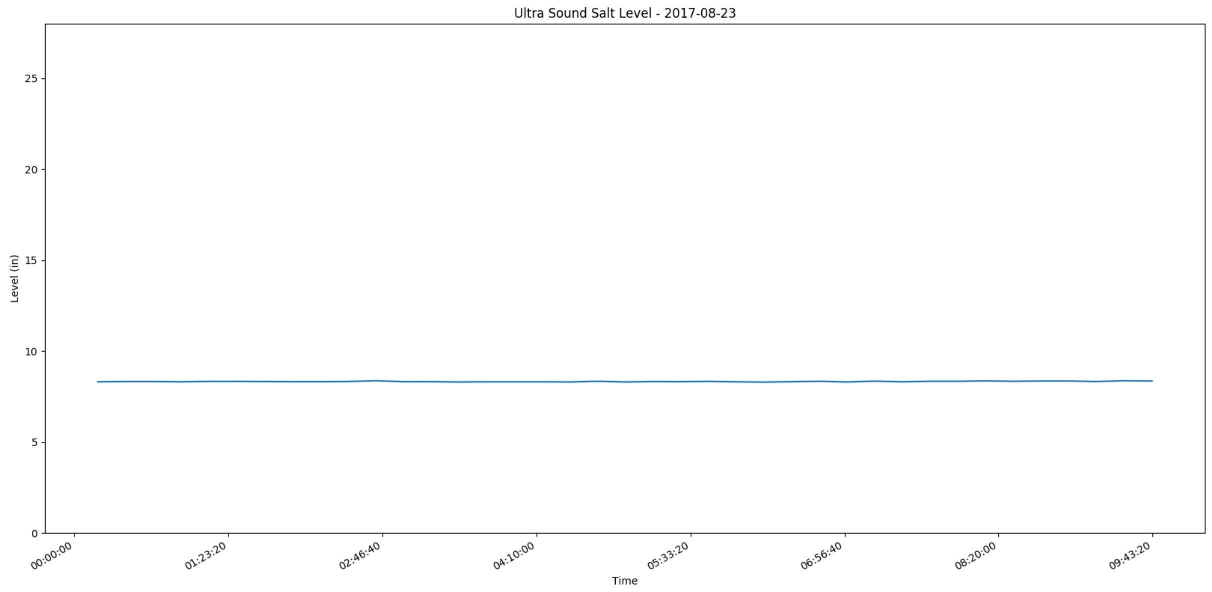


Figure 12: Visualization of Salt level data.

Figure 12 shows the change in salt level over the course of the day, measured by an ultrasonic level sensor. The salt truck is not in service so the salt level remains constant. Figure 13 shows the change in temperature over the course of the day, measure by a temperature sensor installed on a truck. The truck was inside the garage, so the temperature stayed pretty much constant.

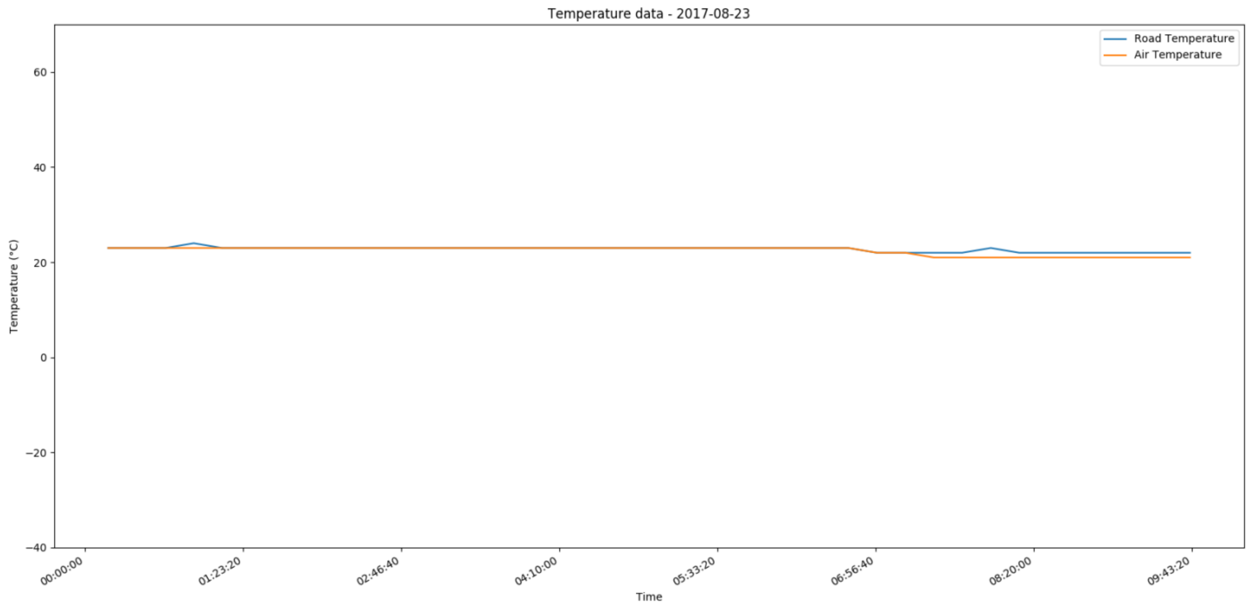
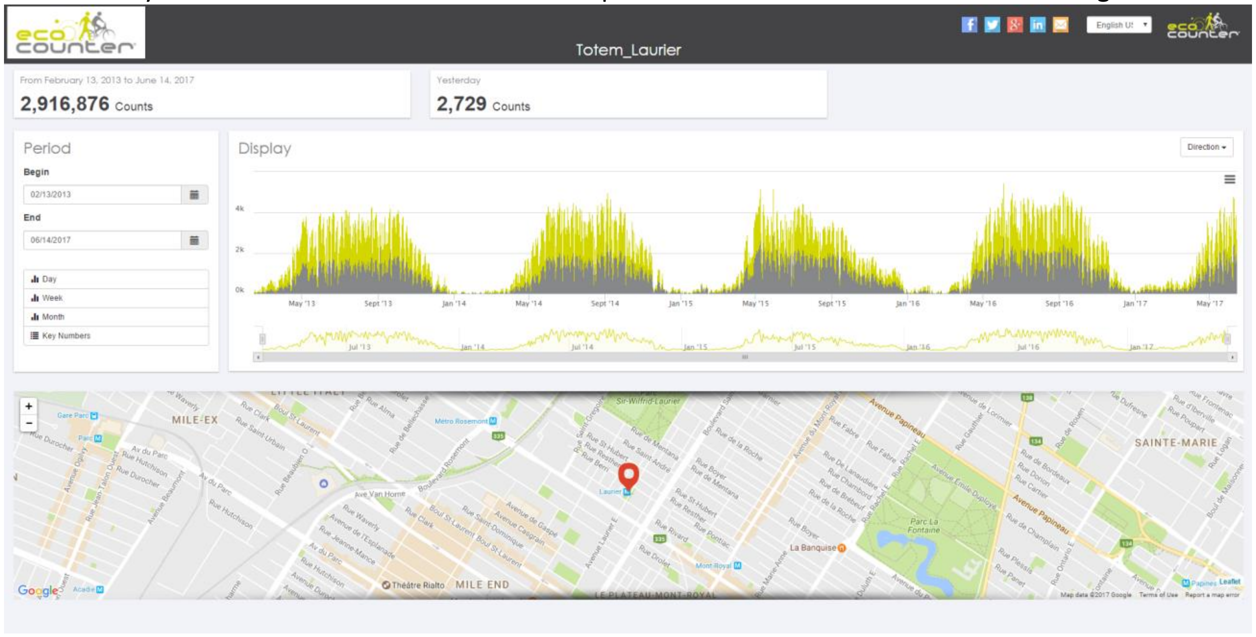


Figure 13: Visualization of air and road temperature data from salt truck.

4 Conclusion:

This Appendix presents the design of the prototype data management platform that control the data collection from the sensors, formatting, storage and visualization. Both the on-premises and cloud-based setups are investigated and main functional block placements and implementation is presented.
However, the platform is just a prototype and further improvements could be achieved. Firstly, a more user-friendly and interactive interface could be implemented such as the one as shown in Figure 14.



From February 13, 2013 to June 14, 2017
2,916,876 Counts

Yesterday
2,729 Counts

Begin
02/13/2013

End
06/14/2017

Figure 14: Graphical User Interface example for bicycle traffic data [9].

Moreover, for real-time data display, it is not recommended to query data directly from the database due to the high overhead and also pricing (for cloud-based setup). Instead, a MQTT broker can be used to store temporary data and supply them to the appropriate subscribers. In this project, for testing purpose a MQTT broker based on Node-Red was implemented for real-time data query and the MQTT topics are as the following:

Radar topics:

Table 4: Radar MQTT topics

Name in report	Location	IP addr : port	Topic on MQTT	Name in DB
VdM_IoT_Rad_001	SCH6 - WEST	192.168.10.53:10001	*odtf1/ca/qc/mtl/mobil/traf/radar/geolux/sch6/west*	*traf_rad_001*
VdM_IoT_Rad_002	SCH6 - NORTH	192.168.10.53:10002	*odtf1/ca/qc/mtl/mobil/traf/radar/geolux/sch6/north*	*traf_rad_002*
VdM_IoT_Rad_003	SUH28 - SOUTH	192.168.10.52:10001	*odtf1/ca/qc/mtl/mobil/traf/radar/geolux/suh28/south*	*traf_rad_003*
VdM_IoT_Rad_004	SUH28 - WEST	192.168.10.52:10002	*odtf1/ca/qc/mtl/mobil/traf/radar/geolux/suh28/west*	*traf_rad_004*
VdM_IoT_Rad_005	ONH4 - SOUTH	192.168.10.50:10001	*odtf1/ca/qc/mtl/mobil/traf/radar/geolux/onh4/south*	*traf_rad_005*
VdM_IoT_Rad_006	ONH4 - WEST	192.168.10.50:10002	*odtf1/ca/qc/mtl/mobil/traf/radar/geolux/onh4/west*	*traf_rad_006*
VdM_IoT_Rad_007	PKH5 - WEST	192.168.10.51:10001	*odtf1/ca/qc/mtl/mobil/traf/radar/geolux/pkh5/west*	*traf_rad_007*

Note: Only ***$RDTGT*** and ***&RDCNT*** reports from Radars will be published to MQTT broker, right after the data is received. The real-time radar monitoring software will use this data to visualize these data as a chart of targets and their speeds.

Truck topics: The name of the truck in database: *anjou_salt01* . Data will be published every time when they are received.

odtf1/ca/qc/mtl/mobil/traf/truck/sierra/anjou/salt01.

RFID topics: Data will be pulished every time when they are received

odtf1/ca/qc/mtl/invent/qds/rfid/obid/sch6_1/sv1. The name of the server in database: *sch6_1_sv01*

odtf1/ca/qc/mtl/invent/qds/rfid/bleg/sch6_2/sv1. The name of the server in database: *sch6_2_sv01*

5 References:

[1] http://www.eagle-tek.com/wp-content/uploads/2016/07/a34a30731adc016-1.pdf
[2] Massa Model M3 Wireless Ultrasonic Level Sensor – Developer's Guide, February 4, 2014 https://www.massa.com/wp-content/uploads/M3-Sensor-Developers-Guide-20140204.pdf
[3] http://www.gpsinformation.org/dale/nmea.htm#GGA
[4] https://stackoverflow.com/questions/13530762/how-to-know-bytes-size-of-python-object-like-arrays-and-dictionaries-the-simp, Denis Kanygin's answer
[5] https://azure.microsoft.com/en-ca/pricing/details/virtual-machines/windows/
[6] https://github.com/Blutude/Unified-IoT-smart-city-sensor-data
[7] https://azure.microsoft.com/en-gb/pricing/details/cosmos-db/
[8] http://www.azurespeed.com/
[9] http://www.eco-public.com/public2/?id=100117705#