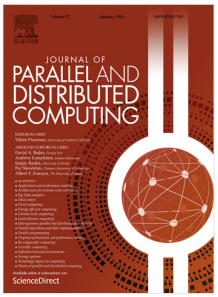
Accepted Manuscript



Edge computing framework for enabling situation awareness in IoT based smart city

SK Alamgir Hossain, Md. Anisur Rahman, M. Anwar Hossain

PII: S0743-7315(18)30613-0
DOI: https://doi.org/10.1016/j.jpdc.2018.08.009
Reference: YJPDC 3936

To appear in: *J. Parallel Distrib. Comput.*

Received date : 28 February 2018
Revised date : 30 May 2018
Accepted date : 21 August 2018

Please cite this article as: S.A. Hossain, et al., Edge computing framework for enabling situation awareness in IoT based smart city, *J. Parallel Distrib. Comput.* (2018), https://doi.org/10.1016/j.jpdc.2018.08.009

***Highlights (for review)**

- Development of edge computing framework to process IoT data
- Elaboration of the process of IoT data at different levels
- Introduced situation image (s-image) and explained situation identification
- Edge computing provides competitive advantage in terms of latency

Edge Computing Framework for Enabling Situation Awareness in IoT based Smart City

SK Alamgir Hossain[1], Md. Anisur Rahman[2], M. Anwar Hossain[3]

Abstract

The Internet of Things (IoT) offers a lot of benefits for building smart cities. Such cities will be able to utilize a huge number of heterogeneous IoT devices that can generate a sheer volume of data. So, considering this heterogeneity, one of the major challenges in smart cities is how to process this data and identify different situations for decision-makers on the basis of this data. The traditional cloud computing approach can provide enormous computing and storage facilities to support data processing. However, it requires all the data to be moved to the cloud from the edge devices of the user endpoint, thus introducing a high latency. In this paper, we used the edge computing approach to minimize such latency. Besides, as major portion of data is generated from the user endpoint, processing this data in the edge can significantly improve the performance. Our experiment shows that processing raw IoT data at the edge devices is effective in terms of latency and provides situational awareness for the decision makers of smart city in a seamless fashion.

Keywords: Internet of Things, Smart City, Situation Awareness, Edge Computing

[1]Computer Science and Engineering Discipline, Khulna University, Bangladesh. alamgir@cseku.ac.bd.
[2]Computer Science and Engineering Discipline, Khulna University, Bangladesh. anis@cseku.ac.bd.
[3]College of Computer and Information Sciences, King Saud University, Saudi Arabia. mahossain@ksu.edu.sa.

1. Introduction

Today, most people use the Internet [1] for communication and content sharing, which has also become a global platform for communication between connected objects and smart devices, introducing a concept called the "Internet of Things (IoT)". IoT is expanding at a unprecedented rate and according to the International Data Corporation (IDC), an estimated 50 billion connected objects will be in use by 2020 (Fig. 1). This brings about the challenge of how to efficiently process the huge volume of IoT data and use it for human benefit.

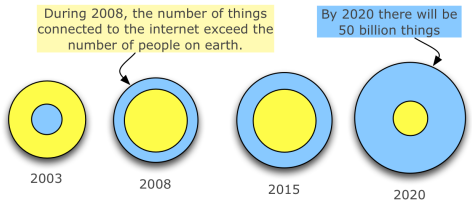


Figure 1: Growth of things connected to the Internet.

Nowadays, IoT has become a part of smart city [2, 3, 4]. IoT-based smart cities has the potential to bring a number of benefits in the management and optimization of traditional public services, such as transport and parking, surveillance and maintenance of public areas, preservation of cultural heritage, garbage collection, and salubrity of hospitals and schools.

Data from the IoT-embedded city infrastructure can be collected and processed to understand the situation of the city and offer new services in future [5] in addition to taking necessary actions. A situation [6] is a condition at a moment in a particular location. In other words, a situation is a set of things that happen and the conditions that exist at a particular time and place. In the IoT, a situation is an abstraction for a pattern of observations made by a distributed system, such as a sensor network. With the expansion of cities, lots of events will likely occur and any inefficient approach to process and handle those situations will hamper timely decision-making. In order to address this challenge in the smart city context, different projects that couple the IoT with cloud

technology to reap the benefits of cloud storage, processing, and infrastructure are already underway [7, 8, 9, 10].

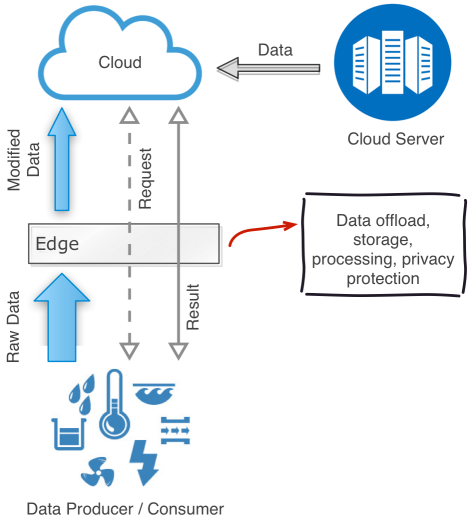


Figure 2: Edge computing paradigm.

In cloud-based processing, all the sensor data are processed through a remote cloud server, but it is costly in terms of processing and storage to send all data to the cloud. In this aspect, the emerging edge computing [11] can help save a lot of bandwidth and may increase the processing speed. Edge computing, also called fog computing, is a method of optimizing cloud computing systems by performing data processing at the edge of the network.

Using edge computing services, the data that needs to be moved to the remote server significantly decreases. This method mainly has three benefits: it can work with large data sizes, it guarantees a low latency, and it focuses on location awareness. Figure 2 illustrates a typical edge computing approach where the edge is the closest processing unit of the data sources (the sensors), it processes the data and sends the filtered data to the cloud server. Here, the main functionality of edges is data offloading, storage, processing, and privacy protection. This paper contributes by providing an edge computing framework to process situations in a IoT smart city that would help the decision makers

to be situation aware and provide relevant services to city residents.

The rest of this paper is organized as follows. In Section 2, we comment on some research works on edge computing, situations, and IoT smart cities that are related to our approach. Section 3 depicts the proposed situation detection mechanism. In Section 4, we show the architecture of our proposed framework. Section 5 presents the implementation issues, development challenges and the outcome that we found from our examination. At the end, we provide the conclusion and discuss future work directions of the paper in Section 6.

2. Related Works

Situation awareness (SA) has previously been studied in different domains, as composite events in distributed event based systems, service composition in multi-agent systems, and macro-programming in sensor networks. The term was first used by Endsley [6] in the military field. Although many extensions were developed by Endsley from his original SA idea, the combination of SA with sensor networks was missing. Recently, researchers took this into consideration and published several work. Among them, one of the promising works was developed by Oliver et al. [12]. They reviewed three different algorithms for recognizing situations, and finally they introduced a distributed commitment machine algorithm for situation understanding. Like all event-driven systems, the speed of situation understanding depends on the data volume: when the data volume grows, the speed decreases significantly. Although existing works try to minimize the problem using the Service Oriented Architecture paradigm [13] and fog computing [14], among others, no complete framework for SA in the IoT and cloud be highlighted properly.

Among different works that leverage the combination of clouds and the IoT to provide real-time SA, the authors of [15] paid attention to merging cloud computing technologies in order to offer new location-aware services and reduce the latency. Based on this idea, several academic prototypes [16, 17] as well as commercial services (ThingWorx [18] and SmartThings [19]) have re-

cently emerged. At the urban scale, SmartSantander[4] introduced a city-wide experimental research where around 20,000 sensors have been deployed in three different locations. These sensors monitor different situations like parks and gardens irrigation, parking management, traffic monitoring, weather conditions monitoring, and others. By persuasion from SmartSantander, S. Chakrabarty et al. [20] depicted protected IoT engineering for keen urban communities. They introduced a design containing four essential IoT engineering obstacles that empower a protected smart city to mitigate digital assaults starting at the IoT hubs themselves. The authors concurred that the IoT is in its underlying stage and parcel of research are necessary.

For SA, millions of sensors need to send a stream of data (a collection of unbounded data objects that may have multiple data attributes in a particular order) to the cloud server for storage or processing. But this is a very inefficient process. Instead of this approach, the data may be processed in the edge of the network. Inspired by this efficient approach, researchers recently proposed several methods of data processing in the IoT. Among them, in [21], an infrastructure for the IoT using fog computing, called Indie fog, was proposed. In that paper, the authors highlighted two types of Indie fog deployment: (1) integrated form and (2) collaborative form. In the integrated form, all the functionalities are provided by a gateway device that is integrated into the Indie fog server. On the other hand, in the collaborative form, all the functionalities are provided by a workstation as a data source. Another work is [22], where the authors used an existing mobile phone network as an edge node to process the data. They used mobile phones as they can be used as an important data source in an IoT environment. Other researchers [23, 24] also focused their attention on optimizing the IoT data processing using mobile phones. In these works, mobile data are sent to the fog servers, and then the fog servers send the filtered or processed data to the cloud servers.

Data pre-processing is an important task in IoT environments, as data come

[4]Smartsantander, www.smartsantander.eu

from noisy environments that require pre-processing before analysis. Real-time pre-processing is necessary in many cases like stream data processing, and its main target is to produce high-quality data. This is a mandatory step that many well-known techniques like normalization, transformation, integration, and cleaning use. As the number of sensor data streams is large, in stream data pre-processing, we need an efficient solution. There are many existing pre-processing methods for data streams. Those techniques can be grouped into three main categories: instance reduction, dimensionality reduction, and feature space simplification. There are many kinds of instance reduction pre-processing mechanisms available. Of those techniques, the following three are the most popular:

1. *Instance-Based learning Algorithm (IB)* [25]: this method is good for static-type data. It works on the basis of a confidence interval test, and it distinguishes whether a case is added in the database or needs to wait until its appropriate insertion.
2. *The Locally Weighted Forgetting (LWF)* [26]: this method works on the basis of the k-nearest neighbors algorithm and it filters the data by discarding data whose weights are below the threshold.
3. *Salganicoff* [27]: this method can work in both static and dynamic environments. This method is similar to the IB method, but the difference is that in IB all the data are considered, but in Salganicoff only the newest data from the stream are considered. The main problem of this method is that, in order to detect new data, it stores old data in a queue, which requires a lot of memory.

There are also different well-known techniques in dimensionality reduction pre-processing. Some of those techniques are as follows:

1. *Katakis et al. [28]*: in this method, a feature filter method is proposed to detect relevant features. This method is suitable for online data ranking (filtering).

2. *Carvalho et al. [29]*: in this method, the weight is computed by an online classifier to identify the feature and the feature is scored on the basis of the difference between the positive and the negative weights of each feature.
3. *Masud et al. [30]*: in this method, a deviation weight measure is used to distinguish the features. It uses an unsupervised technique like the highest frequency in the data.

The last type of stream pre-processing is feature space simplification. In this method, the main target is to normalize the feature space. This algorithm returns nominal features that can be used for any data processing purposes.

After pre-processing, the IoT devices that are in the environment must be discoverable so that they can be queried for data. An efficient look-up mechanism is necessary, but real-world objects are difficult to look up. So, instead of the physical look-up, it is better to use a virtual table where device insertion and removal can be managed dynamically. One of the promising works in this direction is Snoogle [31]. Our preference for Snoogle over more recent works like Context-Aware Dynamic Discovery of Things (CADDOT) [32] or agent-based discovery [33] stems from the fact that Snoogle is more lightweight, in addition to its security and privacy protection features for sensitive data. In addition, Snoogle allows the indexing of arbitrary kinds of terms, not just numeric ones. Snoogle assigns keywords to sensors as well as links with the associated descriptions of the sensors. Then, it stores the keyword with the description in storage with appropriate indexing. Each Index Point (IP) is linked with all the sensors that are connected to it as every sensor in the same area will be assigned the same IP. This way, a two-tier hierarchy is used to maintain two types of IPs. In the top tier, it keeps the key IP, and in the bottom tier, it maintains all the IPs that are associated with the key IP. Here, the user may perform two types of queries: (1) a local query, which is performed in a specific IP, and (2) a global query or distributed query, where the user search is placed in the key IP. For this search every IP needs to be flooded to find the specific sensor object (see Fig. 3).

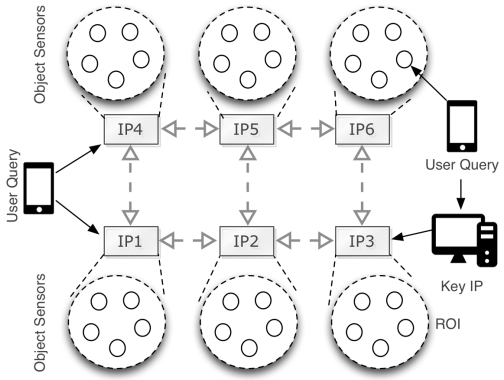


Figure 3: Snoogle [31], IoT discovery and search in the physical space.

From the above study, we found that IoT-based SA has become the cornerstone of smart city development with a push for connected objects, connected devices, and connected everything. So, with the aim of overcoming the disadvantages of the systems mentioned above and inspired by [23, 34, 35], we introduced Edge computing framework for SA in IoT-based smart cities.

3. Proposed Framework for Situation Detection

The primary purpose of the proposed framework is to detect the current situation that represent the state of transportation, healthcare, security and other aspects in a smart city. Efficient situation understanding allows the city officials to provide services to the citizens and perform actions accordingly.

The process of situation detection and awareness lies in the understanding of how the raw IoT data is propagated from the distributed deployment sources to the application level for the decision makers. We propose a layered approach to demonstrate this data propagation, as shown in Figure 4. The figure shows that the data propagation is accomplished in four steps, which are described in the following:

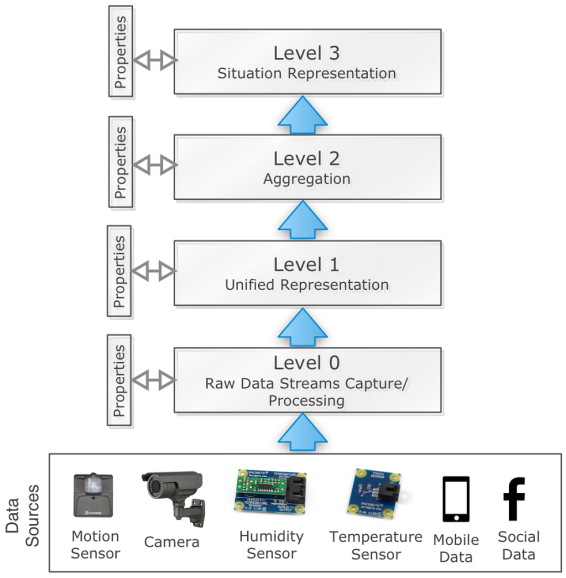


Figure 4: Proposed Situation Detection Steps from Sensor Data.

3.1. Raw Data Stream Capture/Processing

The first step of situation detection is to capture and process the raw IoT data. However, if the raw data are sent directly to the remote cloud server for processing, the battery life will be reduced drastically. For this reason different on-line and off-line mechanism like data compression, aggregation and query modeling is applied to reduce the energy consumption cost. In our proposed approach, IoT data is annotated with its location and frequency of creation. We used ADMM (Alternating Direction Method of Multipliers) model [36], which we will explain in later section, sufficiently matches our case where we have a large set of data that we want to split and process into separate processing units, i.e. in the edge servers.

Fig. 5 demonstrates how the sensors are connected to the edge server. According to this figure, let us consider we have total N number of sensors that have the common objective as represented in the function $f(x)$. In our proposed technique each sensor has an identity with the format $[where(location).who(role).what(description)]$. We use this kind of format for two reasons: one

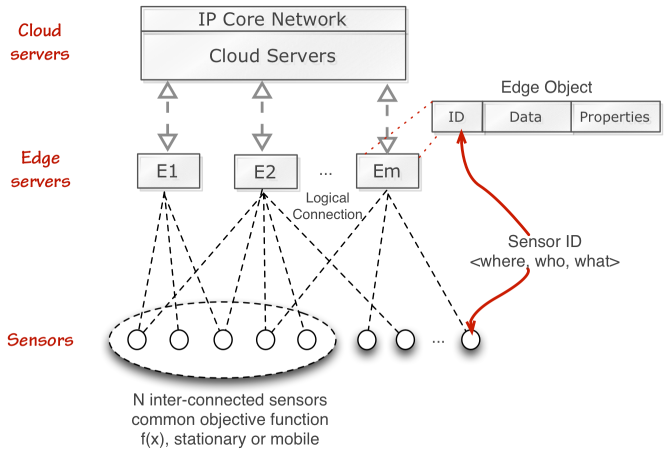


Figure 5: Proposed distributed IoT data processing through the edge servers.

is for effective lookup from large collection of sensor and another is for mobility. Also this type of naming is human friendly. In smart city many of the sensors are mobile so traditional static IP or GUID (global identifier) will not work as we may need to update the sensor dynamically. To illustrate, we provide an example as: say a mobile sensor like a cell phone GPS (phone having id *mobile*3 and GPS having id *gps*1) is tracking and currently it is in a building called *building*1 then our system will generate a dynamic id for the mobile device as [*building*1.*mobile*3.*gps*1]. It should be noted here that for every communication, we need IP or other protocol so this dynamic name will be mapped with IP address/MAC address/URI. As many of the sensors in a smart city are mobile, all the connection between the sensors and edge will be logical connection. All the streams that are produced by IoT devices are splitted into data objects which are called edge objects (see Fig. 5). These edge objects are sent to the edge servers for further processing. The structure of this edge object and the data distribution process are described in the following sections.

3.1.1. Edge Object Structure

In order to make the edge objects each node samples the sensor stream data with a fixed window size. Each edge object is constituted with ID, data and

properties. ID is an identifier (that we described earlier), data contain the IoT data and properties contain functional or control information that are necessary to perform the operation in the edge. Each edge server extracts the data from the edge object and performs operation based on the control message that is in the properties segment in the edge object. Fig 6 represents the XML structure of all the properties like *sensor_id*, *job_type*,*sequence_no* etc. of edge object.

```
<?xml version="1.0" encoding="UTF-8"?>
<edge_obj_properties id="ec089cf1-e6c2-439e-93a3-060b37524d5f">
  <sensor_id>building1.phone3.gps1</sensor_id>
  <sequence_number>5</sequence_number>
  <created_at>02/10/2017 10:32:56</created_at>
  <updated_at>02/10/2017 10:34:32</updated_at>
  <stream_start_time>10:22:56</stream_start_time>
  <stream_end_time>10:32:56</stream_end_time>
  <job_type>MAX/MIN/SUM/AVG/MID/MEDIAN/STD</job_type>
  <ack>processing</ack>
  <edge_serve_details server_id="1205">
    <location>
      <latitude>37.090240</latitude>
      <longitude>-95.712891</longitude>
    </location>
  </edge_serve_details>
</edge_obj_properties>
```

Figure 6: Edge object properties as a xml format.

3.1.2. Data Distribution

As there are lot of devices, distributed processing is required for data analysis. Processing multiple data streams in a centralized cloud server is inefficient. Instead of this if the data is processed through the edge server the overall performance will be improved significantly. And this way a decentralized and parallel processing of IoT data can be achieved. But the question is how to send the data to decentralized processing units without significant loss of performance? We found that ADMM [36] provides a technique that is used to solve classical convex optimization problem. In this approach it breaks a problem into some smaller problems so that each smaller task can be handled in an easier way. We can represent our distributed processing of data through ADMM. We have N number of IoT devices that have a common objective function $f(x)$ and our tar-

get is to minimize (eq. 1) the data or filter through the edge server so that only the refined data can be sent to the cloud server for further storage or situation visualization.

$$f(x) = \sum_{i=1}^{N} f_i(x) \tag{1}$$

where x is a global variable that is unknown and f_i refers to the term with respect to the i^{th} sensor. Now let $\{x_i \in R^n\}_{i=1}^{N}$ is a local variable and z is a common global variable then the above equation can be rewritten as:

$$\min_{x} f(x) = \min_{\{x_1,...,x_N,z\}} \sum_{i=1}^{N} f_i(x_i) \tag{2}$$

where $x_i = z, i = 1, ..., N$ which is originally called the global consensus problem [37], since the constraint is that all the local variables should agree, i.e., should be equal. The augmented Lagrangian of problem Eq. 2 can be further written as:

$$L_\mu(x_1, ..., x_N, z, y) = \sum_{i=1}^{N} (f_i(x_i) + y_i^T(x_i - z) + \frac{\mu}{2}||x_i - z||_F^2) \tag{3}$$

The resulting ADMM algorithm from Eq. 3 is:

$$x_i^{k+1} := argmin_{x_i}(f_i(x_i) + y_i^{kT}(x_i - z^k) + \frac{\mu}{2}||x_i - z^k||_F^2) \tag{4}$$

$$z_i^{k+1} := \frac{1}{|N_i|} \sum_{i \in N_i} (x_i^{k+1} + \frac{1}{\mu y_i^k}) \tag{5}$$

where N_i represents closest neighbour set of the i-th devices.

$$y_i^{k+1} := y_i^k + \mu(x_i^{k+1} - z^{k+1}) \tag{6}$$

The first step (Eq. 4) and last step (Eq. 6) are preformed individually in each edge, and the second step (Eq. 5) is completed in the cloud server.

3.2. Unified Representation

The second step of data propagation is to convert the processed IoT data to a uniform representation before storing them in the off-line storage. This is important as the result of processed IoT data of many sensors need to overlap and need to give a combined output. For example, consider the processing of weather condition for a given area. Here we need to merge the result from many sensors that are producing data in different pattern, format etc. So before sending the data to the cloud server we can pass the data to a model to extract a unified representation. In our proposed model the edge server output data is converted to a unified format such as 'what-when-where' structure. For the unified representation we used the concept W^5HH [38], which is actually a series of questions that extract appropriate result from a problem. This approach is popular in software process and project management.

Table 1: W^5HH Principle used to uniform the sensor data.

Principle	**DataType**	**ExampleData**
What data is generated	Sensor specific data	Temperature, wind_direction, wind_speed
When data is taken	Date Time	10/10/2016 10am, yesterday 1pm
Where the data is coming	Sensor location	Indoor, Outdoor, Home, Office, Bed Room1
Who is generating the data	Sensor id	1, 2, tempS1, humidityS1
Why (in what condition)	Sensor state/activity	Motion, Hot weather
How long data is generating	How Long	Ten minutes, one hour
How much resources is needed	Meta Data	Data length, data type

Table 1 demonstrates the list of questionnaire and the data type it will produce in each of the case of W^5HH principle.

3.3. Aggregation

All the unified represented data are stored in the cloud server and are fed to a visual display or control centre to trigger any event. The unifiedly represented data is aggregated to produce the situation. In the aggregation step the unified data is converted to a 2-dimensional spatial representation for easy visualization and mapping at the situation level. It combines data and represents in an image like representation, which we call Situation Image (*S-image*). Each S-image has mainly 2 parts: S-image element and S-image set. Each S-image element is a collection of key-value pair where key is the state or factor name (i.e. temperature, humidity, sound etc) and the value is the cumulative value for the key that is calculated from the edge server. It should be noted here that in a real world environment besides the *key* and *values* the system needs to store more information like location (*latitude*, *longitude*), meta data (*datalength*, *datatype*) etc. In the real scenario continuous Situation Image will be generated over a period. The continuously generated finite set of situation images is called situation image set or S-image set. Figure 7 demonstrates the structure of a S-image set.

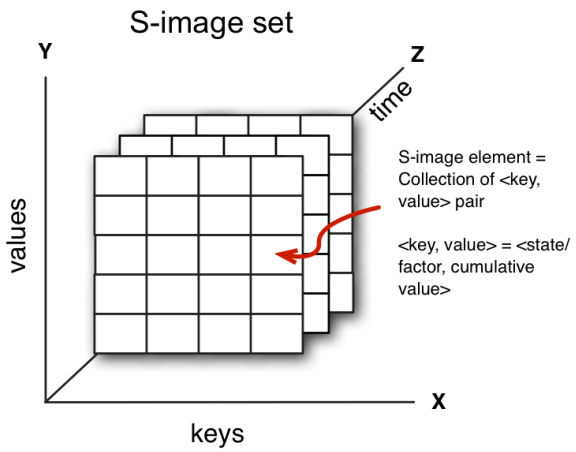


Figure 7: S-image set.

3.4. Situation Representation

We know that situation is a set of information of an environment over a period of time. So our S-image set that we described in earlier section can actually be used to produce the situation. This process is actually a classification task where each cell of the situation image represents a particular event. We can model the situation modeling through the classical situation calculus [39]. According to situation calculus a situation is a series of actions which is started with a special situation called initial situation S_0. In our case S_0 is the situation that is generated from the first S-image over a period of time. According to situation calculus a new situation will be created from the initial situation S_0 when an action d_0 is triggered. In our S-image set continuous S-image is created. This does not mean that all the S-image will be used for situation generation. The next situation will be generated if any action is triggered or if the value of some S-image element crosses certain threshold, T. Sometimes actions may have some preconditions. For example if we say open the camera if the location is free. This task can be written as $Poss(open(v), s) \leftrightarrow is\_free(v, s)$, here $Poss$ is a special function that is used for the execution of an action. The process of data capture to situation representation are illustrated in the following Algorithm 1.

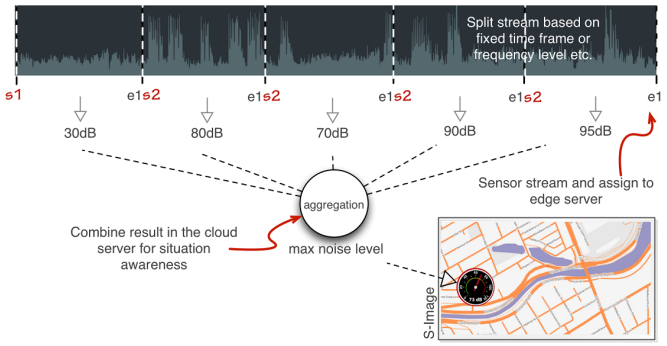


Figure 8: A simplified view of data stream processing through the edge server to the situation image.

Figure 8 demonstrates a simplified view of all the 4 steps that we just described. Consider a sound sensor (microphone) data is being partitioned into

Algorithm 1: *Situation identification*

```
/* This algorithm generates situation based on the data
   receive from the sensor networks. The sensors may be
   individual or may be grouped into some set those have
   same objective.                                        */
while true do
    m^e ← fetchSensorStreamData(m, K);
    e* ← partitionStream(m^e);
    /* Consider the server will process the task in First In
       First Out (FIFO) pattern                            */
    assignToEdgeServer(e*, FIFO);
    while data available in edge server do
        /* Send the edge server data after applying W^5HH
           representation and finally upload the unifiedly
           represented data to the the cloud server        */
        S-image set, η = AggregateCloudData()
        initialize(lastSituation);
        if last_Situation = null then
            last_Situation ← convertToInitialSituation(η[0])
            /* render the last_Situation to any display device */
            continue
        foreach S_image ∈ η do
            if (S_image ∩ last_Situation) ∨ (value(S_image) > Threshold, T_i)
            then
                /* means there is a change in the S-image */
                last_Situation ← convertToSituation(S_image, last_Situation)
                /* render the last_Situation to any display device
                   */
```

edge object and is being sent to the edge network, the edge server processes the data based on the properties and finally applies W^5HH and sends the result to the cloud server. The cloud server aggregates the data that it receives from edges and finally the *S-image* is created by proper placement of the situation elements. The situation image is stored in the cloud storage to calculate the situation.

4. System Architecture

In this section we present different components of our proposed architecture and their functional descriptions. The components of the system are depicted in Figure 9 as a block diagram. The architecture has four main components: Device Module, Edge network, IoT Cloud Middleware and Subscribers. The device module includes sensors and other objects that may sense the environment. In reality things that can produce time dependent data series may be considered as a sensor. Edge network is the network of servers that will work as a processing unit for the sensors. IoT Cloud Middleware is responsible to publish the message and subscribe the clients to receive the messages. This module is also responsible for message routing. Finally the subscribers receive the services from the system. In our case the display devices where the situation for the city will be presented are consider as the subscribers. Generally any object that wants to receive data from the platform may be considered as a subscriber. It is also possible for an object to be both a publisher and subscriber. We provide the functional description of each of the four components in the following sections:

4.1. Device Module

This module is responsible for all the physical or logical linkage between the IoT device and the edge network. Usually by using different standard communication technology like NFC (Near Field Communication), GSM, WLAN, GPS and sensor networks together with SIM-card technology etc. are used to

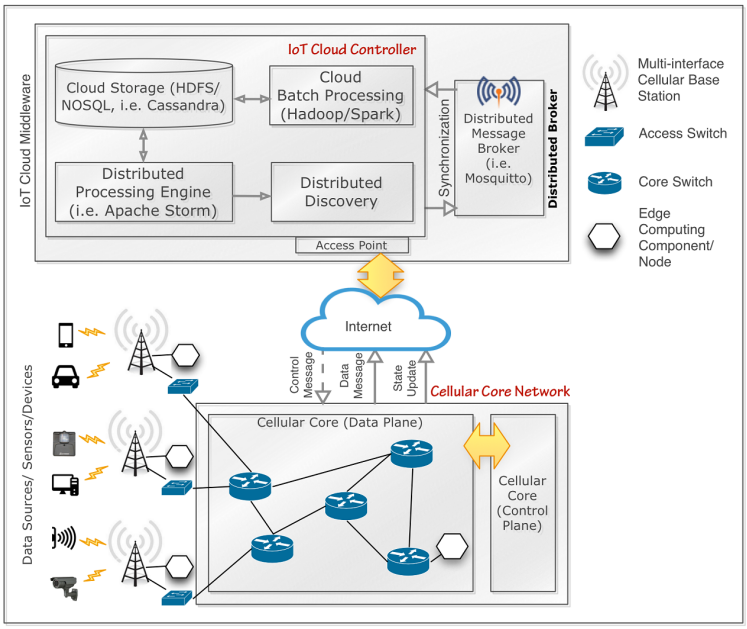


Figure 9: Our proposed architecture.

connect the device module to the physical sensors. For every connection mechanism it is important to save the bandwidth. The data from the sensors to the edge network may be transferred either asynchronously or synchronously. The main functionality of the device module is to send data but these modules also sense control message that are generated by the IoT middleware (i.e it act as a publisher). The control message is important in many cases to control the sensors. For example zoom, pan, tilt and resolution functionality control of a camera sensor, precision, accuracy, sensitivity, range, calibration control of temperature, humidity, accelerometer, IR, distance sensors.

4.2. Edge Network

In the traditional IoT the sensor data is sent to the cloud server for processing. But in this centralized approach has lot of scalability problem. A better solution if we process the data in the edge of the network and send the refined data to the cloud the overall performance will be enhanced significantly. In the

previous section we presented our technique about how to process the data in the edge server. We found that today a huge number of WSN based multi-interface cellular base station is using in cellular network. Using this WSN the IoT devices can be connected, may moving and send data to the edge server at any time. Edge node can be directly connected with the WSN or may connect with the core switch. Cellular core network has two main components one is the data plane and the other is the control plane. The data plane is responsible to route the data to its destination and control plane is responsible to control the network.

4.3. IoT Cloud Middleware

In our proposed architecture we introduced a cloud based IoT middleware that controls and processes IoT data. IoT cloud Middleware has the following three modules:

4.3.1. IoT Cloud Controller

The main functionality of this module is to coordinate synchronized communication with other modules as well as to perform system management services. The controller sends or receives messages from the Distributed Message Broker to create message route between subscribers and IoT devices. The controller also maintains a repository of IoT data and different meta-data information that are necessary to discover, filter and manage the different services. In our architecture we prefer to use Apache Cassandra [40] for storage management. Our preference on Cassandra over other NoSQL database (like MongoDB) or other HDFS based cloud storage is due to its ability to scale while still being reliable. It is possible to deploy Cassandra across multiple servers without lots of overhead. A distributed processing engine is responsible to process unbounded streams of data specially in case of real-time processing. Finally as the IoT devices are developed by different manufactures without following a common standard, interoperability becomes a big issue. In our architecture we achieved interoperability between the devices and the platform by using SOAP based

Web Services [41].

4.3.2. Distributed Message Broker

Generally a message broker is a program that can receive message, translate it and finally can send. As we already discussed in earlier section that pub/sub based message delivery is the most popular in IoT scenario, we used MQTT [42] based pub/sub distributed message delivery mechanism for asynchronous messaging. Unlike REST's GET/POST method of data send MQTT clients publish and subscribe to a dynamic or predefined topic. A topic is a string and consists of one or more topic level. Using this kind of structure we can create a user friendly and self descriptive naming structure. For example in Fig. 10 we can group all the temperature sensors that are located in the ground floor living room of my house. So when any subscriber will subscriber to my home ground floor living room temperature sensors will receive all the temperate sensors update that are located to this location. In this way a simple and lightweight topic level the entire MQTT asynchronous messaging transport to subscribers.

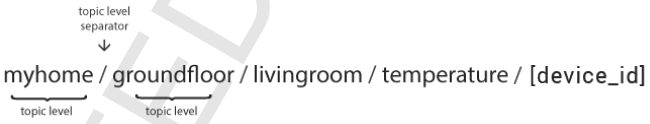


Figure 10: MQTT topic.

4.4. Subscribers

Subscribers are those who want to consume service, which can be a user or a device or any other source that are sending the message. In the architecture, clients can discover and subscribe to the system through the cloud controller. It should be noted here that a single software program may have both sensor and client role in the system. For example a face detection subscriber may produce face video data as well as may look face image inside the video through appropriate subscription to the system.

5. Implementation and Result

This section describes the implementation details of the prototype system that reflects the architecture presented in Section 4.

5.1. Experimental Setup

A prototype implementation of our framework is developed using JDK version 8 and Laravel Framework 5.6. The goal of our implementation is to test the performance of our approach. In order to test the performance we used two well known open IoT dataset (CityPulse [43] and City of Chicago [44]). Several dataset exist in the web for IoT but we selected those two for our experiment because of its availability in different formats and real-time access through api. In our setup we used Amazon AWS [5] cloud based elastic computing infrastructure and for the edge server we used eight personal computers (specification: 2.9 GHz Core i5, 8GB RAM, Windows 10). All the edge computers were connected to the Amazon AWS through Gigabit Ethernet. We performed our test in two cases (off-line and real-time) one is in batch mode and another is in the real mode. In the off-line mode we programmed a special component in our system that can take the data from the data set and pass it to the edge modules. For our test we created total four EC2 instances and S3 storage in Amazon AWS.

For the real scenario we setup a set of sensors as listed in Table 2 on some predefined locations in our city. We selected the location in a manner that we can freely access the environment. The area is approximately $0.45km^2$. Fig. 11 shows a map view of the deployment. In this map view different pin in the map is representing the different nodes. We installed nodes to detect light level, environment noise level, temperature of the environment, traffic presence specially car presence. After selecting the test bed location our main challenge was to place the sensor nodes in appropriate locations as well as to provide uninterrupted power. Practically it is more convenient to use solar energy for battery charges; we used small solar panel to charge the node batteries. Solar

[5] Amazon AWS, https://aws.amazon.com/

Table 2: Sensors used for our testing.

Node Type	**Quantity**	**Sensors**
Humidity	08	Humidity
Light	23	Light, Temperature, Acceleration
Noise	03	Noise, Acceleration
Parking	26	Occupancy
Smart Phone	15	Smart Phone Sensors (also act as edge processing unit)
TOTAL	**75 Nodes**	**160+ Sensors**

Figure 11: Prototype web application interface for live situation monitoring.

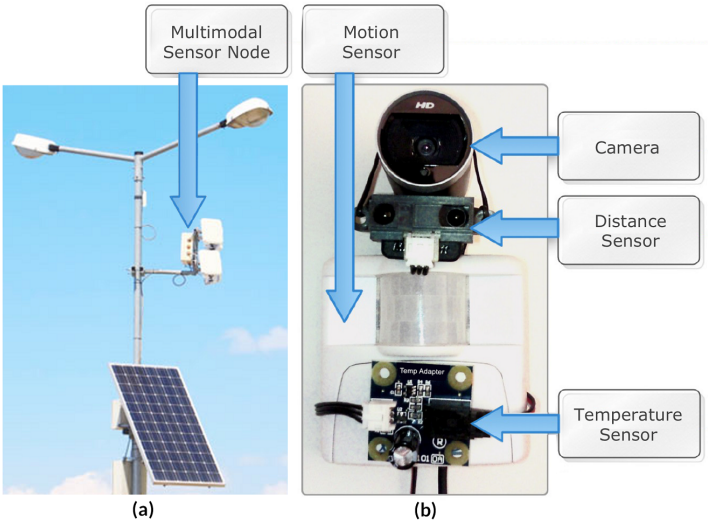


Figure 12: (a) Sensor Node in the lamppost. (b) Multi-modal Sensor Node.

power has the main problem the battery power may be finished during the night but we placed our nodes in the road side lamppost (as illustrated by the picture in Fig. 12.a) which is powered at night by the municipality authority. Some of our node needs more power we connected those node in the lamppost so that it is can be functional at night. We developed a multi-modal sensor node as shown in Fig. 12.b). Each sensor node contain camera, distance, motion, and temperature sensor. Finally we used a metal box to protect the sensor node from weather or human access. In this scenario we consider the mobile phone as an edge processing unit. If any specific event is currently occurring in the city the system will alert the city officials that are subscribed to that service. User also can report the occurrence of such events and it will automatically be propagated to other eligible subscribers.

5.2. Quantitative Results

In order to evaluate the performance of our prototype we conducted several measurement studies. All those studies can be used to justify the acceptability of our approach. At first we test the performance of our system with the datasets

that we listed in the previous section and then we test the processing time and situation detection performance by using the real test environment.

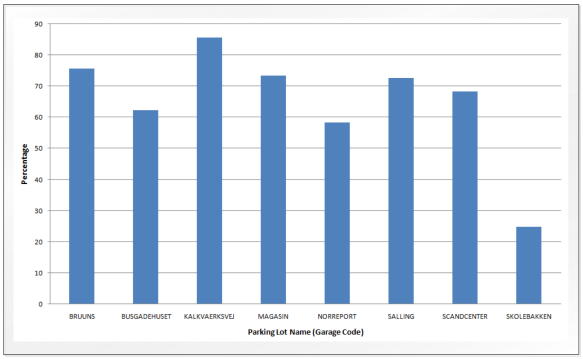


Figure 13: Average percentage of parking space free during the six month period.

5.2.1. Processing Time in Off-line Scenario

We run our prototype in off-line mode with some predefined cases. In the first case we selected the parking data from CityPulse[43] dataset, in this subset 8 parking lots information from City of Aarhus was recorded that was taken 6 months duration and contains total 55,264 data points. We developed a thread hook that will fetch the data from the data set and split to Edge object and finally send to the edge servers and after processing the result will combine and send to the cloud server, the cloud server then annotate the date for situation processing. Through this test the system will show the situation of each parking lot (average percentage of busy) during the 6 month period. We calculate the same operation with and without our edge processing technique. The situation result is demonstrating in Fig 13. According to this figure we see that most of the time the parking was free and to calculate this result our approach required 102ms whereas the non edge version of the system consumed 3.5s to produce the result.

The Chicago park management authority maintain different sensors (humidity, pressure, wind speed, temperature etc) to monitor the public space. Using

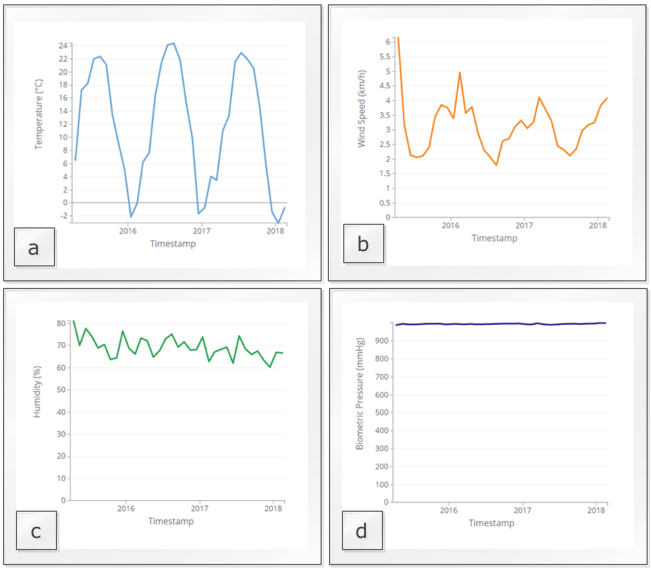


Figure 14: Beach Weather Stations [44]: a) temperature b) wind speed c) humidity d)pressure sensors data.

this open data[6] we test the performance of our prototype. In the data set contain total 66,111 samples; in each sample contain different sensors data (Air Temperature, Wet Bulb Temperature, Humidity, Rain Intensity, Interval Rain, Total Rain, Wind Direction, Wind Speed, Barometric Pressure, Solar Radiation etc). Among this data we used Temperature, Humidity, Wind Speed and Pressure data for our experiment that are plotted in the graph representing in the Fig 14. We passed this data to our prototype with objective function $avg(datapoint)$, by this objective function it take the data and average the data points and finally send the filtered data to the cloud server for situation, we compared the system result with the graph. Our result is almost identical as shown in Fig 14 and for the test it consumed total 400ms where as the non edge version of the system consumed 10.5s for the operation.

[6]City of Chicago, Data Set, https://data.cityofchicago.org/d/77jv-5zb8

5.2.2. Processing Time in Real-time Scenario

In order to measure how our system is performing in terms of time we embedded monitoring hooks in the system and recorded the response time. Finally we analysis the processing time from the sensor node to the subscribers by using the following Equation 7.

$$D = I + t_1 + \alpha_1 + t_2 + \alpha_2 + \beta \tag{7}$$

In this equation α_1 and α_2 are the processing time of the edge and cloud platform respectively. t_1 and t_2 are the transmission delay to send the data from the sensors to the edge and from edge to the cloud servers, I is the sensors average sensing time from the environment, and β is the user receiving time. As the edge devices are with close proximity with the sensors so the time t_1 is approximately zero, so the actual delay D will be $I + \alpha_1 + t_2 + \alpha_2 + \beta$.

After any event occur in the environment the system approximately requires $D = (352 + 256 + 3500 + 243)ms$ to finish the task. In our test the sensor processing time is 352ms, network delay from edge to cloud is $256ms$ and total processing time in edge and cloud is $3500ms$ respectively and delay in the user is $243ms$. In Figure 15 the performance of the sensor and cloud platform modules is depicted.

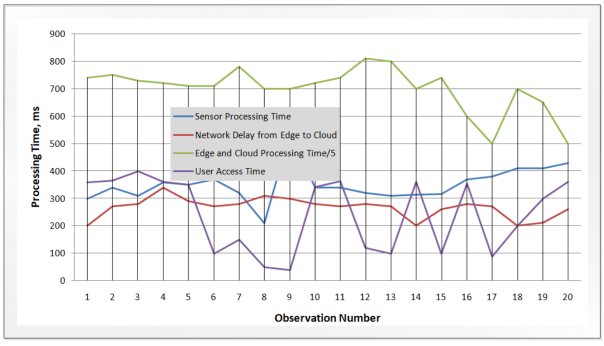


Figure 15: Processing Time of Different Modules.

5.2.3. Situation Detection

In order to test the situation detection performance we monitor the environment approximately 4 weeks and calculated the number of event occurred versus the event detected by our method. Out of approximately 181 different events like high temperature, low humidity, hot sunny day our prototype able to detect 176 events perfectly. Sometimes our algorithm shows misleading information due to the communication delay as well as the server load.

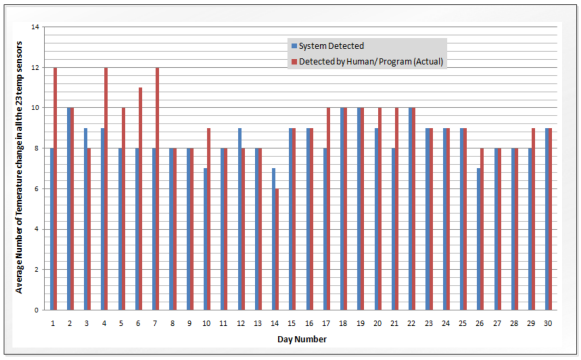


Figure 16: Situation detection performance on 23 temperature sensor over 4 weeks.

Figure 16 demonstrates the situation detection performance over temperature sensors that we used in our prototype test case. There are total 5506 temperature raise or drop during the measured days. As we performed our test during sunny day so we consider the threshold temperature as 25°C. So a separate sensor is placed in the same location where the nodes are located to calculate the actual temperature raise or fall from the threshold. From this analysis we found that our cloud system produced almost identical result as the actual data. It should be noted here that sometimes our system shows more raise or drop than the actual because sometime for a continuous run or direct sunlight the temperature sensor shows out performance.

5.3. Qualitative Results

We have performed usability tests to qualitatively measure the proposed system. The usability test consists of ten volunteers of different age groups

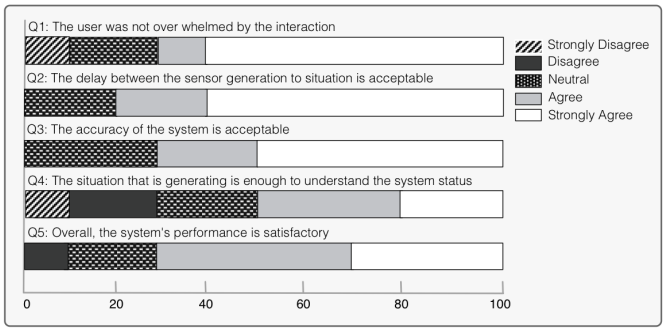


Figure 17: User response percentage on the proposed system.

and academic backgrounds. The users were requested to use the developed prototype. Based on their situation observation they were asked to answer several questions. The answers to the questions are in the range of 1-5 (the higher the rating, the greater the satisfaction) on a Likert scale. Fig. 17 shows the users' responses in percentages. The responses show the suitability of the proposed system.

6. Conclusion

SA ensures that the right information goes to the right people for faster and more efficient communication, especially in the event of an emergency. The question is, given the huge volume of data generated by different IoT devices, is it an important issue to know which data need to be selected and processed? For SA, it is important to process raw low-level IoT data at different levels and generate high-level abstract information for decision-makers. In this paper, we presented an edge computing framework for SA in an IoT-based smart city environment. We performed several quantitative and qualitative studies with our implemented prototype. Our study shows that processing raw IoT data at the edge devices is effective in terms of latency and provides situational awareness for the decision makers. In future we want to evaluate the system further to measure the performance in more real world scenario. We believe that the proposed edge computing framework for SA in an IoT-based smart

city environment will bring new direction in the IoT research.

Acknowledgement

This work was supported by the Information and Communication Technology Division (ICT Division), Government of the People's Republic of Bangladesh [S-Number 028.33.077.17-77, 02-04-2018].

References

[1] I. D. C. (IDC), Worldwide smart connected device shipments, Tech. rep. (March 2012).
URL http://www.idc.com/getdoc.jsp?containerId=prUS23398412

[2] A. Urbieta, A. González-Beltrán, S. B. Mokhtar, M. A. Hossain, L. Capra, Adaptive and context-aware service composition for iot-based smart cities, Future Generation Computer Systems 76 (2017) 262–274.

[3] H. Schaffers, N. Komninos, M. Pallot, B. Trousse, M. Nilsson, A. Oliveira, Smart cities and the future internet: Towards cooperation frameworks for open innovation, in: The Future Internet Assembly, Springer, 2011, pp. 431–446.

[4] J. Bélissent, Getting clever about smart cities: new opportunities require new business models (2010).

[5] D. Cuff, M. Hansen, J. Kang, Urban sensing: out of the woods, Communications of the ACM 51 (3) (2008) 24–33.

[6] M. R. Endsley, Design and evaluation for situation awareness enhancement, in: Proceedings of the Human Factors Society annual meeting, Vol. 32, SAGE Publications Sage CA: Los Angeles, CA, 1988, pp. 97–101.

[7] B. Vanelli, M. P. da Silva, G. Manerichi, A. S. R. Pinto, M. A. R. Dantas, M. Ferrandin, A. Boava, Internet of things data storage infrastructure in

the cloud using nosql databases, IEEE Latin America Transactions 15 (4) (2017) 737–743.

[8] N. H. Ab Rahman, N. D. W. Cahyani, K.-K. R. Choo, Cloud incident handling and forensic-by-design: cloud storage as a case study, Concurrency and Computation: Practice and Experience 29 (14).

[9] L. Atzori, A. Iera, G. Morabito, The internet of things: A survey, Computer networks 54 (15) (2010) 2787–2805.

[10] A. Laya, V.-l. Bratu, J. Markendahl, Who is investing in machine-to-machine communications?, in: 24th European Regional ITS Conference, Florence 2013, no. 88475, International Telecommunications Society (ITS), 2013.

[11] N. Antonopoulos, L. Gillam, Cloud computing, Springer, 2010.

[12] R. Cardell-Oliver, W. Liu, Representation and recognition of situations in sensor networks, IEEE Communications Magazine 48 (3).

[13] B. Cheng, D. Zhu, S. Zhao, J. Chen, Situation-aware iot service coordination using the event-driven soa paradigm, IEEE Transactions on Network and Service Management 13 (2) (2016) 349–361.

[14] E. Saurez, K. Hong, D. Lillethun, U. Ramachandran, B. Ottenwälder, Incremental deployment and migration of geo-distributed situation awareness applications in the fog, in: Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems, ACM, 2016, pp. 258–269.

[15] F. Bonomi, R. Milito, J. Zhu, S. Addepalli, Fog computing and its role in the internet of things, in: Proceedings of the first edition of the MCC workshop on Mobile cloud computing, ACM, 2012, pp. 13–16.

[16] G. C. Fox, S. Kamburugamuve, R. D. Hartman, Architecture and measured characteristics of a cloud based internet of things, in: 2012 International

Conference on Collaboration Technologies and Systems (CTS), IEEE, 2012, pp. 6–12.

[17] F. Li, M. Vögler, M. Claeßens, S. Dustdar, Efficient and scalable iot service delivery on cloud., in: IEEE CLOUD, 2013, pp. 740–747.

[18] ThingWorx, Internet of things and m2m application platform., Tech. rep., PTC, http://www.thingworx.com, last accessed date: 26/05/2018.

[19] SmartThings, Smartthings open cloud., Tech. rep., Samsung, https://blog.smartthings.com/tag/smartthings-open-cloud/, last accessed date: 26/05/2018.

[20] S. Chakrabarty, D. W. Engels, A secure iot architecture for smart cities, in: 13th IEEE Annual Consumer Communications Networking Conference (CCNC), 2016, pp. 812–813. doi:10.1109/CCNC.2016.7444889.

[21] C. Chang, S. N. Srirama, R. Buyya, Indie fog: An efficient fog-computing infrastructure for the internet of things, Computer 50 (9) (2017) 92–98.

[22] R. Roman, J. Lopez, M. Mambo, Mobile edge computing, fog et al.: A survey and analysis of security threats and challenges, Future Generation Computer Systems 78 (2018) 680–698.

[23] T. H. Luan, L. Gao, Z. Li, Y. Xiang, G. Wei, L. Sun, Fog computing: Focusing on mobile users at the edge, arXiv preprint arXiv:1502.01815.

[24] K. Habak, M. Ammar, K. A. Harras, E. Zegura, Femto clouds: Leveraging mobile devices to provide cloud service at the edge, in: Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on, IEEE, 2015, pp. 9–16.

[25] D. W. Aha, D. Kibler, M. K. Albert, Instance-based learning algorithms, Machine learning 6 (1) (1991) 37–66.

[26] M. Salganicoff, Density-adaptive learning and forgetting, in: Proceedings of the Tenth International Conference on Machine Learning, Vol. 3, 1993, pp. 276–283.

[27] S. Marcos, Tolerating concept and sampling shift in lazy learning using prediction error context switching, in: Lazy learning, Springer, 1997, pp. 133–155.

[28] I. Katakis, G. Tsoumakas, I. Vlahavas, On the utility of incremental feature selection for the classification of textual data streams, in: Panhellenic Conference on Informatics, Springer, 2005, pp. 338–348.

[29] V. R. Carvalho, W. W. Cohen, Single-pass online learning: Performance, voting schemes and online feature selection, in: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2006, pp. 548–553.

[30] M. M. Masud, Q. Chen, J. Gao, L. Khan, J. Han, B. Thuraisingham, Classification and novel class detection of data streams in a dynamic feature space, in: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Springer, 2010, pp. 337–352.

[31] H. Wang, C. C. Tan, Q. Li, Snoogle: A search engine for the physical world, in: The 27th Conference on Computer Communications (INFOCOM), IEEE, 2008, pp. 1382–1390.

[32] C. Perera, P. P. Jayaraman, A. Zaslavsky, D. Georgakopoulos, P. Christen, Sensor discovery and configuration framework for the internet of things paradigm, in: Internet of Things (WF-IoT), 2014 IEEE World Forum on, IEEE, 2014, pp. 94–99.

[33] P. C. Ccori, L. C. C. De Biase, M. K. Zuffo, F. S. C. da Silva, Device discovery strategies for the iot, in: Consumer Electronics (ISCE), 2016 IEEE International Symposium on, IEEE, 2016, pp. 97–98.

[34] A. Zanella, N. Bui, A. Castellani, L. Vangelista, M. Zorzi, Internet of things for smart cities, IEEE Internet of Things Journal 1 (1) (2014) 22–32.

[35] Smartsantander, Future internet research and experimentation, Tech. rep., http://www.smartsantander.eu/ (2016).

[36] S. Boyd, Alternating direction method of multipliers, in: Talk at NIPS Workshop on Optimization and Machine Learning, 2011.

[37] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein, et al., Distributed optimization and statistical learning via the alternating direction method of multipliers, Foundations and Trends® in Machine Learning 3 (1) (2011) 1–122.

[38] B. Boehm, Anchoring the software process, IEEE software 13 (4) (1996) 73–82.

[39] H. Levesque, F. Pirri, R. Reiter, Foundations for the situation calculus (1998).

[40] Apache, Apache cassandra, Tech. rep., http://cassandra.apache.org, last accessed date: 26/05/2018.

[41] B. Suda, Soap web services, Retrieved June 29 (2003) 2010.

[42] OASIS, Mqtt 3.1.1 specification, Tech. rep., http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html, last accessed date: 26/05/2018.

[43] D. Puiu, P. Barnaghi, R. Tönjes, D. Kümper, M. I. Ali, A. Mileo, J. X. Parreira, M. Fischer, S. Kolozali, N. Farajidavar, et al., Citypulse: Large scale data analytics framework for smart cities, IEEE Access 4 (2016) 1086–1108.

[44] C. of Chicago, City of chicago open data, Tech. rep., https://data.cityofchicago.org/, last accessed date: 26/05/2018.

Author Biographies

	**SK Alamgir Hossain** received the B.Sc. Engg. degree in Computer Science and Engineering from Khulna University, Khulna, Bangladesh, in 2006, and the M.C.S. degree in Computer Science from the University of Ottawa, Ottawa, ON, Canada, in 2011. Currently he is working as a Ph.D. student in Computer Science and Engineering Discipline, Khulna University, Khulna, Bangladesh. At University of Ottawa, he was associated with the Multimedia Communications Research Laboratory (MCRLab), School of Information Technology and Engineering. He is also a faculty at Computer Science Department, Khulna University, Khulna, Bangladesh. He has authored or co-authored more than 20 publications including refereed journals, conference papers and book chapter. His research interests include Internet of Things (IoT), Smart Environment, Ambient Intelligence and Humanized Computing and Virtual reality with Haptic.
	**Md. Anisur Rahman** received his PhD degree in Computer Science at the University of Ottawa, Canada. He completed his Masters from Asian Institute of Technology, Thailand, in 2000. He is also a faculty at Computer Science Department, Khulna University, Bangladesh. His research interests are data integration, schema mappings, model management; He has published several research papers at international journals and conference proceedings.
	M. Anwar Hossain is an Associate Professor in the Department of Software Engineering, College of Computer and Information Sciences at King Saud University (KSU), Riyadh, KSA, since 2010. He obtained his master degree in Computer Science from the University of Ottawa, Canada, in 2005 and Ph.D. degree in Electrical and Computer Engineering from the same University in 2010. His current research includes multimedia surveillance and privacy, ambient intelligence, smart cities, internet of things, and media cloud computing. He has authored/co-authored over 100 research articles. Dr. Hossain has co-organized several IEEE/ACM workshops including IEEE ICME AAMS-PS 2011-13, IEEE ICME AMUSE 2014, ACM MM EMASC-2014, IEEE ISM CMAS-CITY2015, and IEEE ICME MMCloudCity-2016 workshop. He served as a guest editor of Springer Multimedia Tools and Applications journal and International Journal of Distributed Sensor Networks, and Springer Multimedia Systems journal. He has secured several grants for research and innovation. He is a senior member of IEEE and ACM