Towards an IoT Platform with Edge Intelligence Capabilities

Vladimir FLORIAN[1]*, Gabriel NEAGU[1]

[1]National Institute for Research and Development in Informatics,

8-10 Marshal Averescu Blvd., Bucharest 1, RO-011455, ROMANIA

vladimir@ici.ro (**Corresponding author*), gneagu@ici.ro

Abstract: A specific feature of the *IoT* systems consists in continuously generated data by sensors and smart devices, which makes necessary enabling real time pre-processing and filtering as close as possible to their location. To cope with this specificity, the intermediary architectural level of *Fog computing* has been considered for this class of systems. The paper presents a solution for implementing this architectural extension to a current, cloud-oriented pilot *IoT* platform. The theoretical background on which this solution is based includes as its main topics the *Fog computing*, the Edge analytics and the Publish/subscribe interaction model. Based on their analysis, the architecture extension requirements specific to each topic are detailed. The adopted approach combines features and functionalities of both content-based and topic-based publish/subscribe models, with the aim to promote Edge Analytics principles by moving computation closer to where data resides and providing required performance for data-in-motion analysis.

Keywords: *Fog computing*, Edge analytics, Internet of Things, Health monitoring, Publish/subscribe interaction model, Subscription matching, Event delivery architecture.

1. Introduction

Fog computing, a distributed computing paradigm proposed by Enterprise Networking Labs of Cisco Systems [6], extends Cloud computing and services to the edge of the network. A more recent definition emphasizes the connection to the cloud [19]: "*Fog computing* is a geographically distributed computing architecture with a resource pool that consists of one or more ubiquitously connected heterogeneous devices (including edge devices) at the edge of network and not exclusively seamlessly backed by cloud services, to collaboratively provide elastic computation, storage and communication (and many other new services and tasks) in isolated environments to a large scale of clients in proximity".

Before *IoT* emerged, the specifics of data intensive and *Big Data* analytics applications consisted in moving the data to the location where processing and storing took place. This paradigm is working well where large volumes of historical data or static data are involved, like for example, batch processing seismic or satellite image data. It also works for cases where data transmission requires low bandwidth and not real time.

The *IoT* system specific use cases imply that sensors or smart devices that monitor variables like temperature, humidity, vibration, acceleration or numerous others, generate data that need to be handled by back-end systems very fast and very often (e. g. every millisecond). This huge amount of data streaming from things could saturate networks, or exceed storage and processing capabilities.

Lots of data being generated at the edges need support for sophisticated analysis. For example, data that falls within normal parameters might be ignored or routed to lower-cost storage for archival, while that which falls outside the norm could trigger an alert and be sent to the *Big Data* analytics platform. For that reason, it is better to do some of the pre-processing or filtering of the data closer to where the data is being created. One solution that aims for scalability and efficiency is to distribute the analytics to the edge, or very close to it [2]. Smart devices, or the devices themselves, with relative low cost computational power could run (some part of) analytics. Additionally, upwards in the hierarchy, intelligent gateways with potentially more compute power could enable more complex distributed analytics and intelligence, close to the edge.

In order to provide Edge analytics capabilities the *Fog computing* platform must include software components that support provisioning additional services like data distribution and control, data storage, aggregation and analysis, metadata management and data quality assurance. The implementation of these components is best suited as a distributed middleware layer named Gateway Tier.

IoT Gateways play a central role in many Internet of Things applications. They are the link between (wireless) low-power sensor networks, on the one side, and Cloud or enterprise systems on the other side. Typically, an *IoT* Gateway is based on a powerful processor like e. g. ARM Cortex or

https://doi.org/10.24846/v27i1y201807

Intel SoC and runs a Linux operating system. The applications running on these devices, typically, acquire some data from connected sensors or devices, pre-process this data, and send it to a Cloud or enterprise server for further processing and/or storage.

The development of *IoT* systems has enabled a new class of applications with real time processing and high data transfer rates requirements. These applications collect the individual streams of data generated by *IoT* connected devices in order to gain important insights, optimize processes, etc. To cope with inherit heterogeneity of devices, platforms and services, specific to *IoT* systems, data oriented architectural solutions have been proposed, based on the *IoT* data cycle. An example is the Data Tweet architecture, which implement four generic services: data collection, data dissemination, data consumption, and configuration management [9].

In health monitoring, due to the fact that unlabeled and continuous data are usually dealing with, the data mining tasks must be preceded by pre-processing such as: data cleaning, noise removing, data filtering and compressing as a part of any physiological data monitoring framework [4].

The aim of the paper is to present a solution for the extension of the current, cloud oriented architecture of a pilot *IoT* platform for health monitoring [18], according to the above mentioned architectural trend. The current pilot belongs to the context-aware platforms, integrating three emergent technologies (Wireless Sensor Networks, Internet of Things and Cloud Computing) [13].

The remaining part of the paper is organized as follows: Section 2 presents the main theoretical background topics that were considered for to achieve the above mentioned aim: *Fog computing*, Edge analytics and Publish/ subscribe Interaction Model. On this basis, Section 3 synthesizes the extension requirements for the *IoT* platform architecture. The proposed extension solution is presented in Section 4. In the framework provided by the publish/subscribe model two additional architectural sub-layers are detailed: content-based matching and event delivery. Some concluding remarks are included in the final section.

2. Theoretical Background

2.1 Fog Computing

Fog computing services are situated between end devices and the Cloud, offering various benefits such as low latency, location awareness and mobility support. Fog and Cloud use the same resource types (networking, compute, and storage), and the same mechanisms and attributes (virtualization, multi-tenancy, containerization) [5]. However, there exist some fundamental differences. Fog addresses applications and services that do not fit well the Cloud paradigm, benefitting more from computing and storage distribution simultaneously with using Core Cloud services.

The *Fog computing* model supports very well *IoT* applications, characterized by latency constraints and requirements for mobility and geographical distribution [6].

From architectural and system perspective, the Fog is a complex, highly virtualized, distributed infrastructure encompassing several vertical levels [5], comprising a number of Fog nodes. These are heterogeneous resources (servers, embedded systems, edge routers, access points, set-top boxes), with a wide variety of hardware and software capabilities, placed between end devices and traditional Cloud computing Data Centers.

A *Fog computing* platform provides compute, storage and networking services supporting various communication protocols, various end devices such as sensors, actuators or mobile phone sensors, with heterogeneous interfaces and ensuring integration with the Cloud [8]. The components of a Fog based application are running on the Fog nodes as well as on edge devices [16], between sensors and the Cloud (i.e. on smart gateways, routers or other dedicated Fog devices)(Figure 1).

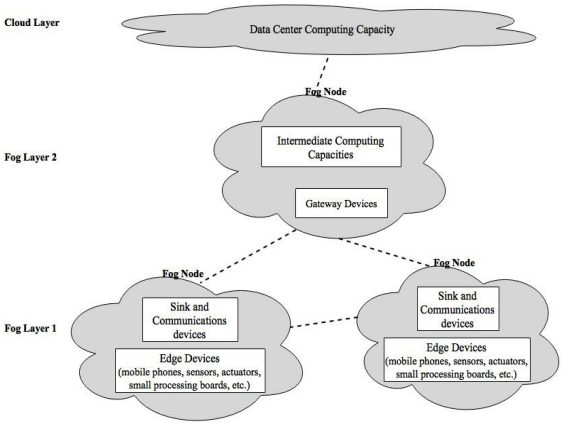

Figure 1. The Fog conceptual architecture

In contrast with Cloud computing, where resources are deployed and managed in a centralized

mode, paper [6] points out that because the geo-distributed nature of the sensor placement, there is a need for a distributed intelligent platform at the Edge able to provide management and orchestration mechanisms.

The Fog software architecture components can be included in one of the following functional layers:

- *Fog Abstraction Layer* [5]. It hides the underlying platform heterogeneity, ensuring uniform and transparent access to resources for monitoring and management. It relies on virtualization technologies and exposes specialized APIs.

- *Orchestration Layer* and Policy-based services [10]. It contains several components for data management and sharing across the Fog platform and for dynamic service orchestration. The orchestration concerns the management of the life-cycle of services and policy based security management. This layer provides a specialized Data API and an Orchestration API through which supports data aggregation, decision making, data sharing and migration. Based on these, core analytics and intelligence services can be further implemented.

- *Service Verticals Layer* [5]. The Service verticals are domain specific applications residing on the platform and functioning in multi-tenancy mode.

The concrete implementation of the Fog (Figure 1) is organized as a hierarchical structure (tree or connected graph), composed of interconnected "Fog nodes", which exist at many layers of the hierarchy forming a rich interconnect topology with links between themselves, to smart objects and to Cloud layers.

A node is a heterogeneous virtualized component, scalable and adaptable, deployable in a variety of environments. It can be configured to perform specific functions, required at the various levels of the hierarchy. Fog nodes lower in the network hierarchy (closer to the endpoints) could have relatively simple hardware and software configurations and modest capacity and performance specifications, while those higher in the hierarchy (closer to the Cloud) could be more complex, with performance equal to high-end servers and high bandwidth networking equipment.

2.2 Edge Analytics

Edge analytics is an approach to data collection and analysis in which some analytical computation is performed on data at sensors and other in-network devices instead of sending it to a centralized data store. Edge analytics introduces an additional layer of intelligence and automation where the data is created, at the edge of the network. So, decisions on what information is worth sending upwards into a cloud or on-premises data store for later use can be taken. The advantages of this approach include:

- Reduction of network bottlenecks and congestions;

- Faster response times, enabling real-time analysis and decision-making, by analyzing data as it are generated;

- Greater scalability, by pushing analytics algorithms to sensors and network devices it is easy to maintain performance, even as the number of connected devices being deployed in the network and the amount of data being generated and collected increases;

- *Big Data* Analytics is performed on actionable data (ingested data is filtered and only necessary data is analyzed or sent on for further analysis);

- Reduction of the risk to personal privacy by limiting data exposure through embedding of raw data processing at the sensing points.

2.3 Publish/subscribe Interaction Model

A typical publish/subscribe system is based on a loosely coupled, communication model that provides a spatial, temporal, and synchronization decoupling between publishers and subscribers. Publishers generate events called publications. Information is transferred in form of notifications following the production of events. Subscribers register their interest in an event, or a pattern of events, and are subsequently asynchronously notified of events generated by publishers. Event delivery consists in finding the interested subscribers and sending them appropriate notifications about those events. It relies on an event notification service providing storage and management for subscriptions and efficient delivery of events [11]. It acts as a neutral mediator between publishers, acting as producers of events, and subscribers, acting as consumers.

The Event Service or Notification Service basically realizes the followings: (i) stores all the subscriptions associated with the respective subscribers, (ii) receives all the notifications from publishers and (iii) dispatches the published notifications to the correct subscribers [17]. In case of distributed implementation, it is composed of a set of nodes distributed over a communication network usually using an overlay network of brokers as a substrate [3]. The publishing and subscription calls are processed by any of the brokers that share the load of managing subscriptions and publications.

By decoupling of message sources and destinations the publish/subscribe interaction pattern provides scalability as it can handle vast amounts of publishers (data sources) and subscribers (data sinks) [11]. Additionally it can provide a control of data flowing from the end node devices in the platform to the processing components.

The performance of a publish/subscribe system is determined by the manner it solves two related problems: the matching between the events and subscriptions, and event delivery [3]. Event-subscription matching is the problem of finding all the subscriptions that match a given notification, especially to efficiently match large numbers of events and subscriptions. Event delivery is the task of delivering the notification to the set of interested subscribers selected by matching. This implies the routing of messages through the system and delivering them to subscribers ensuring certain Quality of Service requirements.

Different ways for specifying the notifications of interest have led to the identification of several subscription models providing different levels of expressiveness for subscriptions. Depending on the subscription model used publish/ subscribe systems can be classified into three broad categories: topic-based, subject-based and content-based schemes.

Topic-based and subject-based publish/ subscribe models enable information consumers to register to a set of predefined topics organized into a hierarchy. Most of the presently implemented open source messaging systems can be broadly categorized as topic-based. These provide mechanisms to identify a subject (topic) and to maintain a list of subscribers for that subject. When events occur, they notify each subscriber on the subscription list. They are based on standards and protocols for message passing at “Session layer” (MQTT, AMQP, DDS, etc.), proposed by different standardization organizations. They are IP applications and use TCP or UDP as Transport layer standard protocols.

The content-based publish/subscribe model (or property-based [11]), introduces a subscription scheme based on the actual content of the published messages, following an attribute/value schema. In other terms, events are not classified according to some predefined external criterion (e.g., topic name), but according to the properties of the events themselves.

In a topic-based system, processes exchange information through a set of predefined subjects (topics) which represent many-to-many distinct (and fixed) logical channels. In a content-based system, messages are only delivered to a subscriber if the attributes or content of those messages match constraints defined by the subscriber. The subscriber is responsible for classifying the messages.

Content-based publish/subscribe systems allow more flexibility in specifying subscriber interests as subscriptions are related to specific information content. Subscriptions specify filters on event contents. Only those events with attributes matching the filters are delivered to the subscriber. Thus, it becomes possible the implementation of distributed hierarchical publish/subscribe mechanisms. Based on these, messages can intelligently be routed to their final destination based on their content. On the intermediate nodes of the hierarchy several in-network processing and data filtering algorithms can be embedded.

However, the complexity of content-based event-subscriber matching requires sophisticated subscription management strategies to be implemented. Consequently, the routing only of the filtered events is preferred. Therefore, the overhead for subscription processing and event matching is considerably higher in content-based matching than in topic-based matching.

Event routing is the core mechanism behind a distributed publish/subscribe system [3]. The event routing algorithms used have to provide performance and scalability. That is, an increase of the number of brokers, subscriptions and publications should not cause a substantial degradation of performance.

Event delivery, in the case of content-based publish/subscribe system, is a problem of multicasting publications to interested subscribers when the set of destination hosts can change with every message, i. e. dynamic multicasting. Solving this problem becomes more difficult if the system has to scale to support a large number of publishers, subscribers and events.

3. Architecture Extension Requirements

The essential features that must be implemented by an ideal *Fog computing* platform for *IoT* applications have been emphasized in [14]. These, could be summarized as following.

- dynamic discovery of Internet objects;
- dynamic configuration and device management;
- multi-protocol support both at communication level and application level;
- support for mobility of gateway devices and temporary connections among the gateways and the smart objects (devices) on the edge;
- multi-source and heterogeneous data management capabilities;
- context discovery and awareness;
- capability to use semantic metadata to annotate a given resource;
- limited computational capabilities of the gateway devices (only the most common data analytical capabilities, such as average, ignoring outliers, etc.);
- security and privacy ensuring capabilities;
- ability to communicate with cloud *IoT* platforms.

• In order to support the development and embedding of edge analytics applications the platform has to:

- provide capabilities so that *IoT* devices and apps can adapt continuously and react locally to their environments and also to commands from neighboring devices;
- enable developers to access, configure, and adjust the algorithms implemented;
- be able adapt and accommodate algorithms to the device size at the *IoT* edge or at gateways;
- support execution of all or most *IoT* processing locally, reducing or eliminating the need to round-trip many capabilities back to cloud-based computing clusters;
- capability to scale to support the size, volume, and speed of data from the various *IoT* devices;
- capability to distribute execution of analytic algorithms dynamically out to disparate edge devices in order to maximize end-to-end application speed, throughput, and agility;

From this perspective it becomes obvious that it is necessary a middleware solution able to fulfill the various requirements of both simple and complex Fog nodes, including sensor and also actuator controllers.

According to [12] the specific requirements a publish/subscribe system should fulfill to be used in cloud-based *IoT* deployments are the followings:

- Capability to match the publishers and subscribers based on symbolic addresses and also to address and contact certain devices;
- Capability to filter the subscriptions and to control the expressiveness of the filters; while a topic-based filtering is mandatory, being suitable for basic subscriptions to certain sensors, a content-based scheme is highly desirable being able to inform on complex events like thresholds, alarms etc.;
- Support for QoS; some sensor data messages might require guaranteed delivery of messages. The middleware should therefore enable annotating subscriptions and messages with QoS requirements;
- Capability to accommodate various message formats required by the heterogeneity of the sensor devices used.

4. Proposed Extension Solution

4.1 Publish/Subscribe Framework

In the *IoT* context, the objective of data analysis is to extract information from the low level sensor data and deliver them to the high level knowledge representation. Regardless of the specific algorithms used, a generic architecture for analytical processing and mining data gathered from *IoT* data sources must support several or all of the following functional capabilities: data pre-

processing, feature extraction and selection, data aggregation and modeling or input features learning [4]. From a technology perspective, this means it is needed a system that can do the following:

- Capture data in form of message streams coming in high volumes and at high velocity; each message may not be huge in size, but the throughput can be quite substantial;

- Ensures the required capacity and scalability to process these streams and run these transformations and aggregations very quickly. Sometimes, it is necessary to access data from other sources while running the process;

- Provide quick insight to processing results, and persist these messages, at scale.

Presently, there is a trend to implement these functionalities using publish/subscribe architectural pattern and protocols [12]. In order to accommodate the Edge analytics model with the platform, an appropriate publish/subscribe model has to be adopted.

Therefore, our approach in fulfilling the architecture extension requirements (see Section 3) is based on principles presented in [15]. They try to avoid the difficulty to implement a general optimal solution with a monolithic approach by solving two sub-problems of the content based publish/subscribe, separately. It results a layering of the system, in two sub-layers: content-based matching and event delivery.

The adopted approach and the architecture designed (Figure 2), is relying on using as a base component, a trusted, on the shelf available, topic-based message broker. For this purpose, several distributed, massively scalable, extensible message brokers (servers) open-source solutions are available. These are able to support both MQTT V3.1 and V3.1.1 protocol specifications, and at the same time, other industrial used protocols like: AMQP, MQTT-SN, CoAP, WebSocket, STOMP, or SockJS. Such a system is able to manage the communication tasks for a large number of devices, including basic requirements of *IoT* architectures.

For more complex tasks (like edge analytics), the topic-based message broker is complemented with a mechanism which transforms the topic notifications provided as outputs, in complex event streams. The event streams are matched with higher level attribute-based subscriptions in order to detect complex event occurrences. This matching problem is treated as pattern matching over event streams.

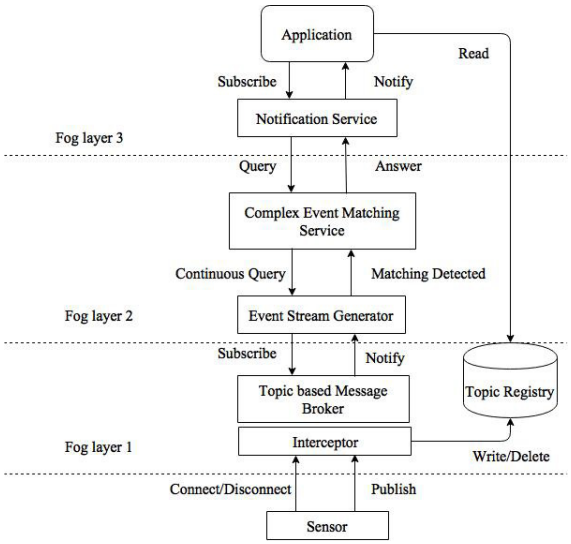

Figure 2. The proposed extended architecture

4.2 The event stream generation mechanism

A stream generation process is started by a *IoT* data producer, usually a sensor, that connects to the broker, creates a topic and begins acting as a topic publisher by sending messages to the broker. Topics are identified by name. Their names look very similar to URLs. A topic is a simple string that can have more hierarchy levels, which are separated by a slash. A sample topic for sending temperature data from a sensor could be `room_no/patient_id/temperature`. The messages send by the publisher are identified by the publisher id, topic name and in their payload, contain the actual data that is sent (in this example temperature value).

Analogously, a stream generation process is terminated by the sensor that disconnects from the broker.

The listener component (Figure 2), implemented as a plugin of the topic-based message broker intercepts every connect/disconnect message and triggers the dynamic creation/destruction of an event stream generator instance. The event stream generator component acts as a subscriber to the topic of the corresponding message. It connects to and receives notifications from the broker. It attaches time stamps to the notifications received and transforms them into complex events i. e. tuples of attribute-value pairs.

The second task of the listener component is to update the topic registry component whenever a

topic is created/deleted. This implies creating and deleting a corresponding record in the registry for that topic.

4.3 Complex event matching service

Its objective is to implement a scalable and elastic event matching service in the *Fog computing* environment. Unlike the classical process of selecting from a large volume of subscriptions those that successfully match against the occurring events, the event matching aims to detect previous subscribed attribute-based patterns (subscriptions) in the incoming event streams.

Pattern matching over event streams is a processing paradigm where continuously arriving events are matched against complex patterns and the events used to match each pattern are selected for output. The patterns are formulated as complex queries that specify constraints on extent, order, values, and quantification of matching events.

The systems that implement this paradigm are called Complex Event Processing (CEP) systems. These match incoming events continuously against long-running queries that are previously registered by the user. The CEP model is inverse to the typical database management model. Whereas a typical database stores data, and runs queries against the data, a CEP system stores queries, and runs data through the queries.

The main techniques which the CEP systems are based on are: automata, tree structures, and logic-based rules. The automata based approach consists in using a SQL like declarative language to express the patterns which are further compiled into Nondeterministic Finite State Automata (NFAs), used by the event processing engine.

Presently, there are a number of open source implementations of such systems which fulfill the scalability, fault tolerance and distribution requirements to be used in implementing the framework. Prominent examples are SASE+ [1] and Cayuga [7].

4.4 Event delivery service

This component has a double-sided functionality. It provides the adaptation interface between the query/answer synchronous access mode required by the CEP matching service and the complex predicate subscriptions coming from the user applications in asynchronous mode. Also, it implements the notification of event occurrence to the subscribing applications. It has to be a distributed scalable notification service able to reside on several Fog nodes and to deliver messages to the subscribers by exposing a content based publish/subscribe specific API.

The applications are enabled to formulate complex subscriptions by accessing the Topic Registry.

5. Conclusions

By placing data, processing and services at the edge of the network instead of entirely in the Cloud, *Fog computing* and Edge analytics are able to support complex features as distributed analytics and edge intelligence. Thus, efficient applications for pervasive health monitoring can be developed, to support smart care decision making (like e. g.: early predictors and novel biomarkers discovery), in connected health scenarios.

The research contribution consists in the approach combining features and functionalities of both content-based and topic-based publish/subscribe models in order to provide a solution with high availability, computational efficiency and greater expressiveness in specifying and filtering the event contents.

The approach is considered to be validated within the effort to further develop the available *IoT* pilot platform.

Acknowledgements

This work was supported by the Institutional research programme PN 1609 “COGNOTIC - Systems, technologies, methods and models for knowledge development in ICT” (2016-2017), project PN 1609-04-01, funded by the Ministry of Research and Innovation.

REFERENCES

1. Agrawal, J., Diao, Y., Gyllstrom, D. & Immerman, N. (2008). Efficient pattern matching over event streams. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data* (pp. 147-160). ACM.

2. Anand, M. (2014). *The New Analytics Imperative*. CISCO Blog entry (11.12.2014). Available <http://blogs.cisco.com/news/the-new-analytics-imperative>.

3. Baldoni, R., Querzoni, L., Tarkoma, S. & Virgillito, A. (2009). Distributed event routing in publish/subscribe systems, *Middleware for Network Eccentric and Mobile Applications*, 219-244. Springer, Berlin, Heidelberg.

4. Banaee, H., Ahmed, M. U. & Loutfi, A. (2013). Data mining for wearable sensors in health monitoring systems: a review of recent trends and challenges, *Sensors*, *13*(12), 17472-17500.

5. Bonomi, F., Milito, R., Natarajan, P. & Zhu, J. (2014). *Fog computing*: A platform for internet of things and analytics, *Big Data and Internet of Things: A Roadmap for Smart Environments*, 169-186. Springer.

6. Bonomi, F., Milito, R., Zhu, J. & Addepalli, S. (2012). *Fog computing* and its role in the internet of things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, *AMC* (pp. 13-16).

7. Brenna, L., Demers, A., Gehrke, J., Hong, M., Ossher, J., Panda, B. & White, W. (June 2007). Cayuga: a high-performance event processing engine. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data* (pp. 1100-1102). ACM.

8. Dastjerdi, A. V., Gupta, H., Calheiros, R. N., Ghosh, S. K. & Buyya, R. (2016). *Fog computing*: Principles, Architectures, and Applications. In Buyya, R. & Dastjerdi, A. (eds.), *Internet of Things: Principles and Paradigms*, 61-75. Morgan Kaufmann, Burlington, Massachusetts.

9. Datta, S. K., Bonnet, C., Da Costa, R. P. F. & Jérôme Härri, J. (2016). DataTweet: An Architecture Enabling Data-Centric *IoT* Services. In *Proceedings of the IEEE Region 10 Symposium* (*TENSYMP*), 9-11 May 2016, Sanur, Indonesia (pp. 343-348).

10. Dsouza, C., Ahn, G.-J. & Taguinod, M. (2014). Policy-driven security management for *fog computing*: Preliminary framework and a case study. In *Proceedings of the IEEE Conference on Information Reuse and Integration* (*IRI*) (pp. 16-23).

11. Eugster, P. T., Felber, P., Guerraoui, R. & Kermarrec, A.-M. (2003). The many faces of publish/subscribe, *ACM Computing Surveys*, *35*(2), 114-131.

12. Happ, D., Karowski, N., Menzel, T., Handziski, V. & Wolisz, A. (2016). Meeting *IoT* Platform Requirements with Open Pub/Sub Solutions, *Annals of Telecommunications, 72*(1), 41-52.

13. Merezeanu, D., Vasilescu, G. & Dobrescu, R. (2016). Context-aware Control Platform for Sensor Network Integration in *IoT* and Cloud, *Studies in Informatics and Control*, *25*(4), 489-498.

14. Perera, C., Qin, Y., Estrella, J. C., Reiff-Marganiec, S. & Vasilakos, A. V. (August 2017). *Fog computing* for sustainable smart cities: a survey, *ACM Computing Surveys*, *50*(3), article no. 32.

15. Raiciu, C., Rosenblum, D. S. & Handley, M. (December 2006). Revisiting content-based publish/subscribe. In *Proceedings of the Int. Conference on Distributed Computing Systems Workshops* (*ICDCSW*) (pp. 19-25). IEEE Computer Society.

16. Tordera, E. M., Masip-Bruin, X., Garcia-Alminana, J., Jukan, A., Ren, G.-J., Zhu, J. & Farre, J. (2016). What is a Fog Node A Tutorial on Current Concepts towards a Common Definition, *arXiv preprint*, arXiv:1611.09193.

17. Virgillito, A. (2003). Publish/subscribe communication systems: from models to applications, *Ph.D. Thesis in Computer Engineering*. Università degli Studi di Roma "La Sapienza", Dipartimento di Informatica e Sistemistica, Roma.

18. Yi, S., Hao, Z., Qin, Z. & Li, Q. (2015). *Fog computing*: Platform and Applications. In *Proceedings of the Third IEEE Workshop on Hot Topics in Web Systems and Technologies* (*HotWeb*), 12-13 Nov. 2015, Washington (pp. 73-78).

19. Zamfir, M., Florian, V., Stanciu, A., Neagu, G., Preda, S. & Militaru, G. (2016). Towards a Platform for Prototyping *IoT* Health Monitoring Services. In: *Proceedings of the IESS 2016 Conference*, *Book series: Lecture Notes in Business Information Processing*, *147* (pp. 522-533). Springer.