# Developing IoT Applications in the Fog: a Distributed Dataflow Approach

Nam Ky Giang, Michael Blackstock, Rodger Lea, Victor C.M. Leung
Department of Electrical and
Computer Engineering
University of British Columbia
Vancouver, Canada
Email: {kyng, rodgerl, vleung}@ece.ubc.ca, mblackst@magic.ubc.ca

***Abstract*—In this paper we examine the development of IoT applications from the perspective of the Fog Computing paradigm, where computing infrastructure at the network edge in devices and gateways is leverage for efficiency and timeliness. Due to the intrinsic nature of the IoT: heterogeneous devices/resources, a tightly coupled perception-action cycle and widely distributed devices and processing, application development in the Fog can be challenging. To address these challenges, we propose a Distributed Dataflow (DDF) programming model for the IoT that utilises computing infrastructures across the Fog and the Cloud. We evaluate our proposal by implementing a DDF framework based on Node-RED (Distributed Node-RED or D-NR), a visual programming tool that uses a flow-based model for building IoT applications. Via demonstrations, we show that our approach eases the development process and can be used to build a variety of IoT applications that work efficiently in the Fog.**

***Keywords*—Distributed dataflow, Internet of Things, Programming models, Node-RED.**

## I. Introduction

While there has been significant research interest in the Internet of Things (IoT) over the past decade, much of this has focused on systems aspects of the IoT, including networking [1], systems [2], middleware [3] and Cloud support for the IoT [4]. To date, there has been less work looking at application development, which remains a complicated process due to the intrinsic nature of the IoT: heterogeneous devices/resources, various perception-action cycles and widely distributed devices and computing resources [5] [6] [7]. A promising approach to IoT application development has been to virtualize things using Cloud services, offering a well defined abstraction of devices and a common programming framework [8] [9] for developers to ease the development process. However, this approach is not suitable for a significant part of the IoT application spectrum which requires processing closer to the device, and a tighter coupling between events and actions. For example, in a simple scenario consisting of a virtual switch and light bulb, the system should behave like a real switch, with similar latency and responsiveness. By introducing a centralized Cloud based intermediary, we might introduce unnecessary latency and greater potential for network failure. Further, in many cases, pushing raw data directly into the Cloud is not desirable or necessary for many IoT applications. To reduce communications costs and storage needs, applications may require raw data filtering, aggregation and analysis before data is uploaded in the Cloud [10]. For instance, in STIS [2], a variant of EPC (Electronic Product Code) Information System [11], all raw events are collected from sensors/RFID readers and processed solely on the Cloud, resulting in unnecessary data redundancy and network overhead. By processing this data locally, costs can be lowered and the overall responsiveness of this system can be improved.

In an affort to address some of the shortcomings of the Cloud computing for the IoT, researchers have proposed Fog computing [12] [10] [13]. Unlike Cloud-based virtualization, the Fog computing paradigm leverages the computing, storage and network resources within and at the edge of the network to augment the capabilities of the Cloud. These processing elements, running on a variety of devices such as network gateways, IoT edge devices etc. provide a mechanism to move processing closer to the network edge. By distributing computing resources closer to users and things, the Fog computing model can be a better choice for building applications for the IoT.

However, while it is clear that the Fog computing model offers a number of benefits for typical IoT applications (proximity to data, better latency etc), Fog computing can be significantly more complex from the IoT application developer's perspective. Essentially we have moved from a model whereby IoT devices connect to a central Cloud service and all applications use a well defined IoT API offered by this centralised Cloud service as a way to manage and control remote devices, to one where computation and other resources are distributed across a complex network of servers, gateways and devices - in some cases even across different domains of control. While there are a number of research projects [14] [15] [16] [17] that have addressed this challenge, none of them provides a generic development model that can be widely applied to different application domains. Moreover, the distribution of application logic that involves resource coordination of heterogeneous devices and between the Fog and the Cloud has not been fully addressed.

In this paper, we examine the development of IoT applications that span both the Fog and the Cloud, identifying some important characteristics - and their associated requirements - that most IoT applications possess. These include device heterogeneity, support for different Perception-Action (PA) cycles, mobility and scalability requirements. Toward addressing these requirements we propose a Distributed Dataflow (DDF) programming model that provides an efficient means to develop IoT applications and coordinate resources distributed

978-1-4673-8058-4/15/$31.00 ©2015 IEEE

across hosts in a Fog deployment.

We validate our approach by implementing a DDF framework based on an open-source flow based run time and visual programming tool called Node-RED. Node-RED (NR), an open source project by IBM, uses a dataflow programming model for building IoT applications and services but has been designed as a run-time for individual devices. Our extensions to NR, called Distributed Node-RED (D-NR) extends the NR run-time to help developers to leverage resources between devices and the Cloud to build Fog based IoT applications. A preliminary implementation has been reported in [18], where we introduced a small change to the vanilla NR to support DDF deployment. In this paper, we report on a more extensive analysis of Distributed Dataflow as an approach to IoT programming and in particular its ability to support the Fog computing model. We also present a more complete design and implementation of D-NR and our experience using the system to build some typical IoT applications.

The paper is written as follows. Section II discusses the characteristics and requirements of IoT applications with regard the Fog computing model. Section III introduces our proposed DDF programming model for IoT applications development in the Fog. Our proof of concept implementation of DDF and experiences in building IoT applications are described in section IV and V. In Section VI we examine a number of existing systems that propose implementations of Fog computing or in-network processing frameworks, comparing them to our work. Section VII discusses some issues and lessons learned and lastly section VIII concludes our work.

## II. IoT Applications Development in the Fog and the Cloud

While application development in the IoT has not been researched extensively there has been some work on addressing certain IoT application requirements. In many cases, work on Wireless Sensor Networks (WSN) were applied to IoT application development. To design an effective application framework for the IoT, key characteristics of the IoT must be considered including PA cycle [5], device heterogeneity [6], device mobility, interaction pattern [7], scalability and so on. However, the analysis of these issues is based mostly on existing WSN systems and IoT systems that consist of only physical devices, not a mix of devices and Fog, Cloud infrastructures. For example, in [7], heterogeneity is limited at the difference between sensor and actuator, or in [5] [7], the interaction model is just between devices themselves.

Fog computing extends these device-centric approaches to IoT development by introducing support for edge processing, network processing and a integration with Cloud computing. To date, there has been limited research work on the details of Fog computing, and in particular how to efficiently implement this model to support IoT application development. In [12], the authors identified key differences between the Fog and the Cloud and new challenges in the Fog. This work is rather general and not focused on developing IoT applications. From our experience and based on existing works, we derive four distinctive requirements that Fog-based IoT platforms should be able to support: the need to support different PA cycles, scalability, device heterogeneity, and mobility.

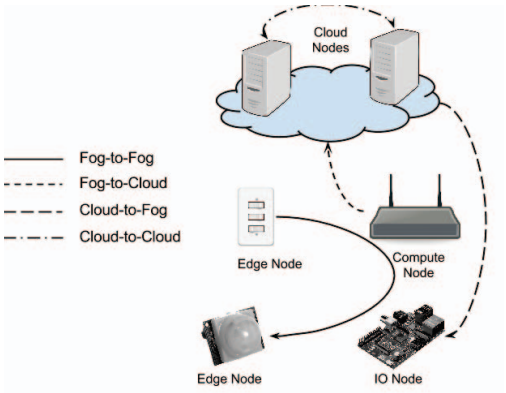


Fig. 1. Perception-Action cycle

### A. Perception-action cycle support

The PA cycle is a natural characteristic of IoT applications whereby participants in the Fog and Cloud infrastructure need to fulfill certain application logic and timeliness requirement for interaction. This is aligned with different interaction models between things and other web services. We identified four interaction models of the PA cycle (Fig. 1). 1) Interaction is between devices (things) in a local network, which satisfies immediate cycle action. 2) Interaction is initiated from things in a local network to the Cloud, which fulfills non-time-sensitive action result. 3) Interaction is initiated from the Cloud to things in a local network, which represents a semi-immediate action result. 4) Interaction is between IoT related web services in the Cloud.

In Fog computing, unlike Cloud computing or device-centric scenarios, computation resources are present both in the Cloud and on the network edge, making it possible for IoT applications that require immediate or semi-immediate action results to be satisfied more efficiently.

### B. Scalability

Scalability is an important characteristic of the IoT since the number of connected things is growing rapidly. Many research efforts rely on Cloud computing to support the analysis of IoT-generated data and application logic that involve trillions of things [19]. However, given the large amount of data generated by IoT devices, implementing IoT applications wholly in the Cloud is neither efficient nor feasible. In some Fog computing scenarios an application will need a method to pre-process raw data on the device or in a local network before uploading aggregated results to the Cloud. Furthermore, when the number of connected things grows, selecting or querying for things becomes a crucial task of an IoT app. Because of this, it can be difficult to maintain an up to date centralized directory of things when they are constantly changing their state and availability. In many cases, participating devices must be able to execute the IoT applications autonomously, independent of connectivity to a centralized Cloud hosted service.

### C. Heterogeneity

Device heterogeneity is another intrinsic characteristic of any IoT system [20]. An IoT platform must support different

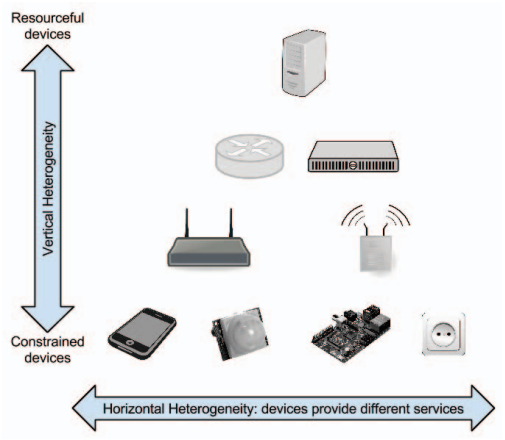


Fig. 2. Device Heterogeneity

levels of device heterogeneity and abstract the device complexity to some extent. From the Fog computing perspective, device heterogeneity in the IoT refers to the differences in not only protocols or device services [21] but also computing resources. We refer to the former differences as horizontal heterogeneity and the later as vertical heterogeneity in accordance to a hierarchical Fog architecture as shown in Fig. 2.

To illustrate this, we classify Fog devices into three types of logical nodes based on their computing resources: Edge, IO (Input/Output), and Compute nodes. Edge nodes produce sensing data and consume actuation messages. IO nodes are capable of brokering communications with Edge nodes but generally possess limited computing resources. Compute nodes have some computing resources to offer. Edge nodes are typically not reconfigurable/programmable, while IO and Compute nodes are more dynamic and usually have some kinds of programmable runtime. A combination of these logical nodes can be implemented in one or separately in different physical devices. The decision on how to assign these logical node classifications to physical devices depends on the system designer and the capability of the device. For instance, a smart gateway with networking and computing resources can act as a Compute and an IO node to manage other physical devices, some acting as Edge nodes, others as IO nodes and Compute nodes.

An IoT application should be able to execute over a group of devices with various computing, sensing/actuating and communication resources to fully leverage the capability of different devices. Accordingly, an IoT system needs to provide a way to define the constraints that regulate where and how application logic should be deployed, so as to efficiently exploit computation resources that are widely distributed and scattered over the edge devices.

### D. Mobility

The requirement for device mobility comes from the natural portability of many physical things and has been identified by many researchers as a key requirement to address when building IoT platforms. However, to date it has mainly been viewed as a service availability problem, in that IoT applications may not be able to consistently request data or send control commands to things [22] [7]. From the Fog computing perspective, not only is service availability affected by a things mobility but also the availability of computational resources nearby. It is not only possible for Edge nodes to move around, Compute and IO nodes may also be mobile.

If Cloud computing cannot be used to provide computation resource for IoT applications (for reasons discussed above), the Fog implementation needs a mechanism to move computation resources in the Fog along with the thing [10] (similar to VM migration in the Cloud or the use of mobile agents), or have a mechanism for duplicating and distribute computation resources to the possible destinations of mobile things (similar to load balancing in the Cloud) [14].

We have identified core requirements for developing IoT applications and discussed these requirements in the context of Fog and Cloud computing combination. These requirements highlight the need for a programming model for the IoT that can leverage computing resources across the Fog and the Cloud for the development of IoT applications. In the next section, we propose Distributed Dataflow (DDF), a programming model for developing IoT applications in the Fog, toward meeting the requirements outlined above.

## III. Distributed Dataflow: a Programming model for the Fog

Dataflow [23] is a well-known programming model that has been applied for developing WSN applications in several works [16] [6] [24]. In the dataflow programming model, application logic is expressed as a directed graph (flow) where each node can have inputs, outputs and independent processing units. There are nodes that only produce outputs and ones that only consume inputs, which usually represent the start and the end of the flow. The nodes processing units process the inputs and produce outputs for downstream nodes. The processing unit of a node executes independently and does not affect the execution of other nodes. Thus, the nodes are highly reusable and portable. Dataflow programming models define their own language grammar for constructing the flow. Many modern systems provide a graphical interface for constructing the flow, with the aim to make it easier to move between design and implementation and reduce development time. Although the dataflow programming model originated as a programming model for parallel execution of tasks using multiprocessors, it has been widely adopted in distributed systems as a coordination language for developing distributed applications [23]. In this paper we refer to dataflow model mainly as a coordination language for developing IoT Applications.

A DDF is a dataflow program where the flow is deployed on multiple physical devices rather than one. Each physical device may be responsible for the execution of one or more nodes in the flow, forming *sub flows*. Some inter-node data transfer may happen between devices. Thus, DDF requires mechanisms for data transfer between physical devices to support communication between nodes on different devices. Unlike homogeneous WSNs, heterogeneous devices in an IoT application often need a flexible communications capability between physical devices so that inter-node data transfer can be made independently of the underlying communication protocols. It also requires a way to dynamically deploy nodes in the

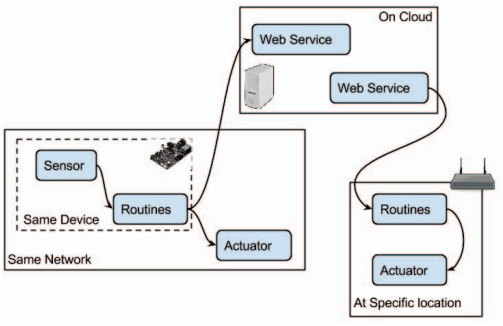


Fig. 3. Distributed Dataflow model

flow onto participating devices and servers. Fig. 3 illustrates our definition of DDF.

For the IoT, the dataflow programming model offers a significant advantage by raising the abstraction level of the underlying IoT systems to ease the developers task without sacrificing much flexibility. This is because once the underlying hardware, protocols and functionality of IoT systems are abstracted as nodes in a flow, much of the design of the application logic is simplified to manipulating node connections and processing generated data. When the developer needs more flexibility or functionality than the current nodes offer, new special-purpose nodes can be developed and deployed, or nodes that support embedded script languages can be used to leverage features of the underlying system, implement new protocols or functionality that does not exist in the current system.

In this way, the development of an IoT application can be split between two classes of developers. Node developers can ensure that the nodes needed to communicate with specific things and their required protocols are available, while IoT application developers can focus on wiring up the flow and creating scripting nodes to connect things and services in the Fog. While this separation of node-level development has been proposed in other works [25] [26], the abstraction of node in dataflow programming model using only inputs and outputs greatly simplifies the application development process. For example, in [25] the authors proposed the notion of a reusable thing driver that abstracts things functionality and hides its complexity. But there, the application developer is still required to learn the drivers complex programming interfaces. Meanwhile in [26], the authors had to trade off flexibility for simplicity when abstracting things functionality into web service interfaces.

When examining the development of IoT applications across the Fog and the Cloud, we hypothesize that a dataflow programming model would be a good fit for the derived requirements in section II. This is because a dataflow program consists of a graph of nodes and the deployment of nodes into an IoT environment can be controlled dynamically and flexibly. A detailed justification of our DDF programming model for IoT applications development in the Fog is explained as follows, based on our outlined requirements.

### A. Perception-Action cycle support

In our discussion we showed that an IoT application can have different requirements on the PA cycle support, ranging from (semi-) immediate actuation after a sensing event is captured, to sense-only application logic. A DDF model can accomplish this by strategically deploying nodes of the flow to edge servers and physical devices. Nodes that are involved in an immediate actuation cycle can be deployed on closely located devices, such as those are on the same network. This keeps the communication delay between Compute and IO nodes to minimal so that immediate actuation result can be achieved.

### B. Heterogeneity

DDF model supports IoT applications with both horizontal and vertical heterogeneity requirements. By developing specialised nodes that represent different things' functionality, developers can build applications that incorporate various thing services by wiring the nodes together. Because the interaction between things is abstracted as dataflow between nodes supplied by developers who are experts in the various protocols and hardware connectivity, application developers need not focus on these aspects of the application, meeting horizontal heterogeneity requirements. Vertical heterogeneity requirements can be met by differentiating participated things based on their computing resource, while constraining the deployment of nodes in the flow so that they are deployed only on devices with appropriate resources. For example, some devices can be marked as Compute nodes; some nodes in the flow are constrained to run only on Compute nodes. This ensures computing resources in devices and gateways at the network edge are leveraged efficiently.

### C. Mobility

We propose to address mobility in the DDF by duplicating nodes in the flow and deploying them around the potential locations of things participating in the application. Generally a node in the flow needs to be deployed on more than one physical device; it is usually duplicated to run on a group of devices so that the system is flexible enough to handle the movement of things. For example, a developer can choose to deploy some nodes in the flow on all gateways in a building based on either network or physical location. This allows other devices that depend on such nodes to move freely around in the building. While it may be possible and more efficient to use a dynamic mobile code migration technique rather than code duplication, weve found this approach to be simpler and so far, suitable for small, simple application logic that does not require excessive computation resources.

### D. Scalability

A DDF application does not rely on a centralised management sysstem to coordinate participating devices. Rather, the participating devices themselves decide how to execute the application logic and communicate with other participating devices based on the flow's design. This helps the system to scale up since no intrinsic management service is required once the initial deployment completes. A DDF implementation in the Fog also provides an easier way to coordinate in-network

processing resources by dynamically wiring physical devices together. This reduces the need to transmit raw data generated by things to the Cloud, improving the overall efficiency of the system.

## IV. Proof of Concept Implementation

To evaluate how practical it is to implement the DDF programming model, we designed a dataflow application framework based on NR [27], a visual dataflow programming language and runtime developed by IBM. NR was developed using JavaScript language with Node.JS technology. It consists of a web-based visual dataflow editor with pre-built input, output as well as processing nodes. It allows developers to write applications by wiring these nodes together. The applications are then forwarded to NR's backend where actual executions of application logic are performed. NR itself was designed to develop applications that run on a single device such as a Raspberry Pi. In our preliminary work [18], we extended NR to support the design of dataflow programs that run across many devices and between devices and the Cloud, which we called Distributed Node-RED (D-NR).

Our IoT implementation consists of D-NR processes running across participating devices in local networks and servers in a Cloud infrastructure. One of these D-NR processes serves as a development server where developers design their application (flow) using the NRs visual dataflow editor. All participating D-NR processes subscribe to an MQTT topic that represents the status of the flow being developed. We will refer to this main dataflow as the *master flow* to differentiate it from the *sub flows*[1] that run on individual D-NR processes. Whenever a developer deploys an update to the *master flow*, all the participating devices and Cloud servers are notified so that they can pull the latest version of it. The participating D-NR instances then parse the *master flow* and based on a set of constraints, decide which nodes should to be deployed locally and which are to be replaced by a *placeholder* node. The *placeholder* node is used to connect *sub flows* from different devices together.

In this paper, we tackle the differences in device capability or device heterogeneity, not addressed in our previous work. Each participating D-NR instance hosted on a device describes its own set of capabilities in advance. In the current design, these capabilities include the devices computation resource, network bandwidth, available storage as well as other user-defined properties such as physical/logical location or device group. The devices computation resource is divided into 4 classes according to the three types of nodes discussed in section II: Edge, IO, and Compute device for Fog environment and Cloud device for Cloud environment. Network bandwidth and storage availability are based on the devices physical network interfaces capability and available disk storage. Physical/logical location or device group are user-defined properties that can be configured by system operators which can be used to identify/locate the device.

Based on this capability definition, constraints can be applied to the *master flow*. Developers can specify which device

[1]Note, our notion of *sub flows* is similar, but subtly different from the notion of sub-flows in vanilla NR. In vanilla NR, sub flows are reusable flow segments which still run on a single machine, while ours are segments of the master flow that run on distributed devices.

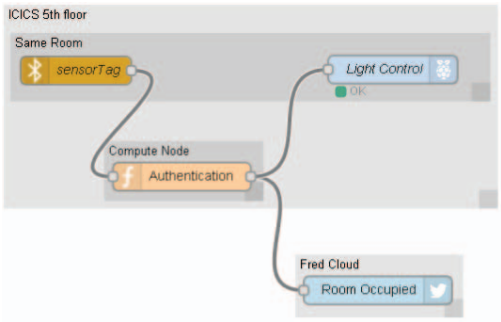


Fig. 4. Develop Fog-based IoT application with D-NR

a *sub flow* should be deployed on by applying constraints on the devices capabilities and properties. In our previous work, we defined only one simple constraint: a deviceId. This deviceId explicitly matches a node in the *master flow* with a device or class of devices it should be deployed to. Since there is only one constraint, assigning distributed sub flows to devices is very straightforward. Developers just need to specify the device constraint by drawing rectangles over the *master flow* to set the desired device on which the nodes should be deployed. In our current design, the constraints are more complex and now include computation resources, network bandwidth, available storage, and user-defined properties. These new constraints continue to act as input into which D-NR instance a node should be deployed on. They are designed based on the new requirements identified on the previous sections. For example, in order to fulfill a time-sensitivity requirement, we should be able to constrain the deployment so that some specific nodes need to be run on the same network to minimise the communication delay between them. While working on the DDF system we have realized that the number and range of constraints can be numerous. With this design we've tried to find the right balance between flexibility and usability in the initial set of constraints we support.

## V. Application Development Experience

To evaluate our proposed DDF programming model we built a typical IoT application in the Smart Environment domain [28]. Specifically, our application consists of a TI SensorTag, two Raspberry Pis (RPi), a PC server in our local network and an Amazon EC2 instance. The RPi, our PC server and the EC2 instance all have our D-NR installed to execute their subset of our applications dataflow. The SensorTag has two buttons, which can control the lights in every room it enters. One RPi has a Bluetooth Low Energy (BLE) dongle that can communicate with the SensorTag, the other RPi is used to control the lights using its GPIO pins. These RPis can be combined as one in real deployment, however to demonstrate the DDF model, we decided to keep them separated.

There is an Authentication node that authenticates the SensorTag based on its MAC address so that only a specific SensorTag can control the lights. The application also updates some web services whenever a control message is sent in order to announce that the room has been occupied. Our application scenario requires immediate actuation of the lights in the room after a control message is sent (a SensorTag button is clicked). Thus, the Authentication node must be deployed in the local

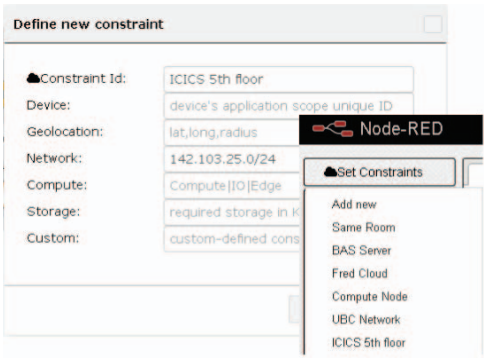


Fig. 5. Creating and Setting Constraints for Nodes

TABLE I. EXISTING IOT FOG AND DATAFOW SYSTEMS

| Works | PA cycle | Hetero-geneity | Mo-bility | Scala-bility |
|---|---|---|---|---|
| Mobile Fog | on same device | vertical | yes | large |
| MA-based MapReduce | perception only | no | no | large |
| Flask, ATaG | local network | horizontal | no | medium |
| glue.thing | via Cloud | horizontal | no | small |
| T-Res, D-LITe | local network | horizontal | no | small |

network (our PC server) while the web services interaction can run on our EC2 instance.

As seen in Fig. 4 and 5, we used constraints in our D-NR implementation to specify how our application logic should be deployed. The SensorTag and Light Control nodes are required to run within the same room so that the SensorTag only controls lights in the room it has entered. This means the movement of the SensorTag is accommodated by duplicating these nodes across devices in our building. This is done by the participating devices themselves whenever a change in the master flow is made. That is, they will pull the master flow, parse it's topology and decide which nodes to deploy automatically instead of having the application manage all the devices. Therefore, our application can execute across a large number of devices when the scale of deployment goes up. Another constraint was applied to the Authentication node, which requires it to run on a Compute Node in our local network. This ensures the result of the Authentication process can be propagated immediately to the Light Control, fulfilling the time sensitive requirement. Lastly, a Twitter node is employed to broadcast the occupancy status of the room. This node is required to run on the EC2 instance (FRED Cloud[2]). Clearly, our D-NR implementation can support different requirements of PA cycle and different device capabilities/services.

## VI. RELATED WORK

In this section we survey some existing frameworks that leverage the Fog computing and dataflow models. These systems are compared to our requirements analysis in Table I and are discussed below.

Hong et al. proposed Mobile Fog (MF) [14] that allows the deployment of an IoT application across multiple devices in a hierarchical system architecture from the network edge to the Cloud. It provides an API for clients to send data vertically or horizontally between devices in a hierarchical network, as well as handlers that are used to process the data in the devices. MF relies on a dynamic node discovery process to associate devices together in a parent-child relationship where parent nodes lend their computation resources to process data received from child nodes. MFs hierarchical system architecture naturally allows IoT applications to aggregate and process data locally along the way from the edge network to the Cloud. In addition, it supports load balancing between parent nodes so that child nodes are associated with underloaded parents, improving the overall scalability of the system. Mobility is handled by the dynamic resource discovery process and duplication of application codes on multiple devices. Meeting the requirements for PA cycle is not fully taken into account by MF since a parent node can only act on the same device from which it receives the message. Because of this, PA cycle logic cannot span multiple devices (e.g. the case where logic that processes sensing data on one device actuates another device). Regarding device heterogeneity, the MF programming model differentiates nodes vertically based on the parent-child relationship and the differences in their computation resources. Furthermore, it does not support differences on services provided by edge devices (there is only one API to get sensing data from all types of devices). Supporting IoT application that operate on different types of edge devices is very difficult as the developer has to manually identify the data format of all edge devices.

Ichiro [17] proposed a Mobile Agent-based (MA) MapReduce, inspired by the MapReduce data analysis model. Instead of transferring all sensing data to a centralised server for processing using MapReduce, the authors performed MapReduce locally at the edge devices to minimise the network consumption. The system allows developers to write and deploy MapReduce procedures on heterogeneous devices. This work supports a typical class of sense-only IoT applications without any actuation so that various PA cycles are not mentioned. However, it proves the need of in-network processing or Fog computing for the IoT.

Distributed middleware frameworks can also be seen as implementations of Fog computing since they provide in-network processing capability that leverages computation resources of edge devices. Daniele et al. proposed T-Res [15], a framework that support in-network processing for WSN. T-Res was implemented in Contiki OS with a lightweight Python virtual machine. T-Res abstracts IoT applications into tasks that are deployed on embedded devices. Each task consists of four sub-resources: input source, output destination, processing function and a last output value. The IoT application is developed by configuring these tasks sub-resources so that data is transferred between devices and processed by devices in-network. Similarly to T-Res, Sylvain et al. proposed D-LITe[29], a distributed middleware for motes based on Finite State Transducers, which is also implemented in Contiki OS. An IoT application is abstracted into a set of rules that control the behaviour of the underlying mote based on the messages

[2]http://fred.sensetecnic.com

it received and its current state. The limitation of these works is that they cannot incorporate application logic that spans across local networks and the Cloud in a unified way. They also do not provide a way for the developer to automatically select the things that participate in the application so that manual configuration and deployment are required. This makes scalability a challenge for both systems. Moreover, in both systems computation resources are tightly coupled into the edge devices assuming the devices have the same capabilities. Finally, in neither case did either of these projects address mobility.

Flask [16] and ATaG [30] are two well-known data-driven programming models for WSN. In both works, authors abstracted the application logic to small processing units that manipulate and exchange data with each other, which then form a dataflow graph. Compared to our proposal, these works do not support vertical heterogeneity, that is, they did not exploit the difference in devices' computation resources for the application logic. That is, while ATaG has a notion of horizontal heterogeneity by assigning attributes to devices to differentiate them based on their sensing/actuating services, Flask operates on a set of homogeneous devices. Furthermore, since these two works were developed specifically for WSN, they do not consider different PA-cycles that was addressed in this paper, such as the interaction that involves the Cloud.

glue.thing [31] is another framework for Iot Applications development that has some similarities to our work in that it was also built on top of the Node-RED platform. In glue.thing, a master device controls and monitors all participating nodes in a local network. Whenever a device joins the network, its special nodes are added into the flow editor of the master device and, made available to the developer. In another implementation, glue.thing deploy the master device on the Cloud and manually register devices to the Cloud so that service composition can be made in the Cloud. However this scheme is less scalable because the glue.thing Composer works with devices individually rather than classes of devices. Our work is significantly different in that we exploit computation resource in the Fog to provide in-network runtime environment.

Compare to DDF, none of these works fully supports the rich set of PA–cycles as proposed and fulfilled in our work. We also put forward a new dimension to the heterogeneity definition of IoT systems and showed that none of these works addresses both vertical and horizontal heterogeneity as ours does. Mobility of things is not handled in most work while MF and DDF share a similar approach that uses code duplication. Scalability remains a difficult problem for developing IoT Applications that only a few of the discussed work can manage large scale deployment. MF achieves this by deploying the same application codes on multiple devices while DDF requests the participated devices pull the codes autonomously.

## VII. Discussion

To program the IoT with D-NR, we have developed some nodes ourselves, which are used to connect to our Web of Thing Toolkit (WoTKit) platform [32]. The development of nodes for Node-RED and D-NR was straightforward; we need only to provide mechanisms for the input and output nodes to authenticate against the WoTKit service, and ways to move control messages and sensor data between D-NR and the WoTKit platform. These new nodes were written in JavaScript making development an easy task. We also inherited a substantial number of nodes that are already made available to the Node-RED platform so that the development of IoT flows with D-NR is largely an exercise of wiring existing nodes together.

The biggest challenge we encountered when developing and deploying IoT applications to the Fog using D-NR is to coordinate communication between nodes on different devices. Currently we employ an intermediate message broker (based on MQTT) as the communication channel between nodes on different devices. Using a broker with publish/subscribe mechanisms ensures loose coupling between participating servers, gateways and devices, and removes the need for components to maintain the IP addresses and host names themselves. However we have found that some application logic requires certain services and devices to have knowledge of each other. For example, in some scenarios, an actuator in room A should only receive command messages from another person or thing in the same room. In another scenario, we may want to deploy the same flows to many different devices located in different places, but we do not want the messages to be exchanged between these places.

In our previous D-NR paper, we used MQTT topics to coordinate communication between nodes comprised of the source node and destination node ids in a distributed flow partition. If we deploy a *sub flow* to multiple devices, all devices listening on that topic receive messages published by the source *sub flow*. This is not always desirable. To address this, we added constraint information to the topic between *sub flows* that nodes on different devices will publish and subscribe to. This is done by hashing all the constraints being applied on a node to make a short, unique topic ID that such nodes will publish and subscribe to. However, using a message broker also has the disadvantage of requiring all communication to go through an intermediary, which can affect the timeliness of a PA cycle. We are currently working on leveraging existing service discovery mechanisms and direct communications protocol[3] to collect devices on a local network.

## VIII. Conclusion

In this paper we have suggested that the Fog approach is a suitable architecture for IoT applications and proposed a Distributed Dataflow programming model as a basis for Fog-based IoT applications. We discussed the core requirements that Fog-based IoT applications need to meet and identified a number of issues with existing approaches to Fog based application development. We have shown how our DDF programming model provides an easy way to design and develop IoT applications by combining application constraints and device capabilities to help drive the dynamic deployment of application logic. Finally, we discussed our implementation of the DDF programming model by extending the Node-RED platform which we used to implement and demonstrate several typical IoT applications. This small scale evaluation has shown

[3]This approach has been employed in another Node-RED fork [33] and we have initiated discussions with the authors about collaborating on this.

the viability of our approach and the many advantages of the DDF programming model and its practicability in real world IoT scenarios. Although we have demonstrated a basic DDF implementation, there are several open issues, for example the need to include a distributed discovery and communications infrastructure suitable for our platform to facilitate the communication between devices that span multiple networks and domains. We plan to address this and other issues in future work.

## ACKNOWLEDGMENT

This work has been partially supported by NSERC and Sense Tecnic Systems Inc.

## REFERENCES

[1] IETF, "Ipv6 over networks of resource-constrained nodes (6lo)," https://datatracker.ietf.org/wg/6lo/documents/.

[2] N. K. Giang, S. Kim, D. Kim, M. Jung, and W. Kastner, "Extending the EPCIS with Building Automation Systems: A New Information System for the Internet of Things," in *2014 8th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, 2014, pp. 364–369.

[3] S. H. Kim and D. Kim, "Multi-tenancy support with organization management in the Cloud of Things," in *Proceedings - IEEE 10th International Conference on Services Computing, SCC 2013*, vol. 6, no. 1, 2013, pp. 232–239.

[4] S. Nastic, S. Sehic, D.-h. Le, H.-l. Truong, and S. Dustdar, "Provisioning Software-defined IoT Cloud Systems," in *The 2nd International Conference on Future Internet of Things and Cloud (FiCloud-2014)*, 2014.

[5] P. Patel, A. Kattepur, D. Cassou, and G. Bouloukakis, "Evaluating the Ease of Application Development for the Internet of Things," Tech. Rep., 2013.

[6] A. Awan, S. Jagannathan, and A. Grama, "Macroprogramming Heterogeneous Sensor Networks using Cosmos," in *Proceedings of the European Conference on Computer Systems (EuroSys)*, vol. 41, no. 3, 2007, p. 159.

[7] L. Mottola and G. P. Picco, "Programming wireless sensor networks," *ACM Computing Surveys*, vol. 43, no. 3, pp. 1–51, 2011.

[8] S. Nastic, S. Sehic, M. Vogler, H.-L. Truong, and S. Dustdar, "PatRICIA – A Novel Programming Model for IoT Applications on Cloud Platforms," in *Service-Oriented Computing and Applications (SOCA), 2013 IEEE 6th International Conference on*. Ieee, Dec. 2013, pp. 53–60.

[9] R. Lea and M. Blackstock, "City hub: A cloud-based iot platform for smart cities," in *Cloud Computing Technology and Science (CloudCom), 2014 IEEE 6th International Conference on*, Dec 2014, pp. 799–804.

[10] M. Yannuzzi, R. Milito, R. Serral-Gracia, D. Montero, and M. Nemirovsky, "Key ingredients in an IoT recipe : Fog Computing , Cloud Computing , and more Fog Computing," in *Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), 2014 IEEE 19th International Workshop on*, 2014, pp. 325–329.

[11] GS1, "Epcglobal network," http://www.gs1.org/.

[12] L. M. Vaquero and L. Rodero-Merino, "Finding your Way in the Fog," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 5, pp. 27–32, 2014.

[13] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog Computing and Its Role in the Internet of Things Characterization of Fog Computing," in *the first edition of the MCC workshop on Mobile cloud computing (MCC '12)*, 2012, pp. 13–15.

[14] K. Hong, D. Lillethun, B. Ottenwälder, and B. Koldehofe, "Mobile Fog : A Programming Model for Large Scale Applications on the Internet of Things," in *the second ACM SIGCOMM workshop on Mobile cloud computing (MCC '13)*, 2013, pp. 15–20.

[15] D. Alessandrelli, M. Petraccay, and P. Pagano, "T-Res: Enabling reconfigurable in-network processing in IoT-based WSNs," in *Proceedings - IEEE International Conference on Distributed Computing in Sensor Systems, DCoSS 2013*, 2013, pp. 337–344.

[16] G. Mainland, M. Welsh, and G. Morrisett, "Flask: A language for data-driven sensor network programs," *MA, Tech. Rep. TR-13-06*, 2006.

[17] I. Satoh, "A Framework for Data Processing at the Edges," *Database and Expert Systems Applications*, vol. 8056, pp. 304–318, 2013.

[18] M. Blackstock and R. Lea, "Toward a Distributed Data Flow Platform for the Web of Things," in *Proceedings of the 5th International Workshop on Web of Things (WoT '14)*, 2014.

[19] F. Li, M. Voegler, M. Claessens, and S. Dustdar, "Efficient and scalable IoT service delivery on cloud," in *IEEE International Conference on Cloud Computing, CLOUD*, 2013, pp. 740–747.

[20] I. Chatzigiannakis, G. Mylonas, and S. Nikoletseas, "50 ways to build your application: A survey of middleware and systems for wireless sensor networks," in *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, 2007, pp. 466–473.

[21] T. Teixeira, S. Hachem, V. Issarny, and Nikolaos Georgantas, "Service oriented middleware for the Internet of Things," in *Proceedings of the 4th European Conference on Towards a Service-Based Internet*, vol. 6994, no. 257178, 2013, pp. 220–229.

[22] C. Beckel, H. Serfas, E. Zeeb, G. Moritz, F. Golatowski, and D. Timmermann, "Requirements for smart home applications and realization with WS4D-PipesBox," in *Emerging Technologies & Factory Automation (ETFA), 2011 IEEE 16th Conference on*. Ieee, Sep. 2011, pp. 1–8.

[23] W. M. Johnston, J. R. P. Hanna, and R. J. Millar, "Advances in dataflow programming languages," *ACM Computing Surveys*, vol. 36, no. 1, pp. 1–34, 2004.

[24] J. Zhang, Z. Li, O. Sandoval, N. Xin, Y. Ren, R. a. Martin, B. Iannucci, M. Griss, S. Rosenberg, J. Cao, and A. Rowe, "Supporting Personizable Virtual Internet of Things," in *Ubiquitous Intelligence and Computing, 2013 IEEE 10th International Conference on and 10th International Conference on Autonomic and Trusted Computing (UIC/ATC)*. Ieee, Dec. 2013, pp. 329–336.

[25] J. Im, S. Kim, and D. Kim, "IoT Mashup as a Service: Cloud-Based Mashup Service for the Internet of Things," in *Services Computing (SCC), 2013 IEEE International Conference on*. Ieee, Jun. 2013, pp. 462–469.

[26] M. Kovatsch, S. Mayer, and B. Ostermaier, "Moving Application Logic from the Firmware to the Cloud: Towards the Thin Server Architecture for the Internet of Things," in *2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS '12)*. Ieee, Jul. 2012, pp. 751–756.

[27] IBM, "Node-red," http://nodered.org/.

[28] A. Whitmore, A. Agarwal, and L. Da Xu, "The Internet of Things-A survey of topics and trends," *Information Systems Frontiers*, no. March 2014, pp. 1–14, 2014.

[29] S. Cherrier, Y. M. Ghamri-Doudane, S. Lohier, and G. Roussel, "D-LITe: Distributed Logic for Internet of Things Services," in *Internet of Things (iThings/CPSCom), 2011 International Conference on and 4th International Conference on Cyber, Physical and Social Computing*. Ieee, Oct. 2011, pp. 16–24.

[30] A. Bakshi, V. K. Prasanna, J. Reich, and D. Larner, "The abstract task graph: A methodology for architecture-independent programming of networked sensor systems," in *Proceedings of the 2005 workshop on Endtoend senseandrespond systems applications and services*, no. Eesr 05, 2005, pp. 19–24.

[31] R. Kleinfeld, "glue . things a Mashup Platform for wiring the Internet of Things with the Internet of Services," in *Proceedings of the 5th International Workshop on Web of Things (WoT '14)*, 2014.

[32] M. Blackstock and R. Lea, "IoT mashups with the WoTKit," in *Proceedings of 2012 International Conference on the Internet of Things, IOT 2012*, 2012, pp. 159–166.

[33] Z. Chao, "mobile collaborative framework," https://github.com/ilc-opensource/mcf.