HINC – Harmonizing Diverse Resource Information Across IoT, Network Functions and Clouds

Duc-Hung Le  
Distributed Systems Group  
TU Wien, Austria  
d.le@dsg.tuwien.ac.at

Nanjangud Narendra  
Ericsson Research  
Bangalore, India  
nanjangud.narendra@ericsson.com

Hong-Linh Truong*  
Distributed Systems Group  
TU Wien, Austria  
truong@dsg.tuwien.ac.at

Abstract—Effective resource management in IoT systems must represent IoT resources, edge-to-cloud network capabilities, and cloud resources at a high-level, while being able to link to diverse low-level types of IoT devices, network functions, and cloud computing infrastructures. Hence resource management in such a context demands a highly distributed and extensible approach, which allows us to integrate and provision IoT, network functions, and cloud resources from various providers. In this paper, we address this crucial research issue. We first present a high-level information model for virtualized IoT, network functions and cloud resource modeling, which also incorporates software-defined gateways, network slicing and data centers. This model is used to glue various low-level resource models from different types of infrastructures in a distributed manner to capture sets of resources spanning across different sub-networks. We then develop a set of utilities and a middleware to support the integration of information about distributed resources from various sources. We present a proof of concept prototype with various experiments to illustrate how various tasks in IoT cloud systems can be simplified as well as to evaluate the performance of our framework.

I. INTRODUCTION

State-of-the-art (and future) Internet of Things (IoT) devices (comprising sensors and actuators, and gateways, routers and switches to manage the same) are typically provisioned and configured within networks that possess high-speed communication capabilities. Such networks would be large in size, comprising thousands of IoT devices, which, through the networks, communicate with cloud services in data centers. Furthermore, today’s virtualization techniques can be applied at different levels, from the gateways integrating sensor and actuator data and control flows, to the networks connecting the gateways and data centers, and the cloud services in the data centers [1], [2]. Such virtualization techniques allow us to create end-to-end slices of IoT, network functions and cloud resources for a single application on the fly.

For applications and management tasks atop such IoT and their corresponding connected network functions, and cloud services, one of the crucial research questions is how to manage these IoT, network, and cloud resources. We need to deal with numerous heterogeneous devices distributed at different edge sites, which obviously employ different technologies and models to manage their IoT resource information. Thus, resource management must deal with heterogeneity and extensibility of underlying IoT, network and cloud resource models. Second, given that virtualization techniques, applications and management tasks expect to treat their managed resources in a view of single end-to-end slices [3], they need a high-level view harmonizing diverse resource information, instead of dealing with low-level information from different providers for different types of infrastructures and resources.

Since centralized management would be too unwieldy for dealing with these issues, a distributed approach is called for, by interfacing with multiple IoT sites, network function services, and clouds. However, a distributed management approach raises the key issue of how to develop and maintain resource models for the various resource providers, while maintaining relationships among various resources that span across sub-networks. This problem becomes especially acute in the presence of network slicing [4], which enables the network functions to be dynamically sliced via a process of “mix and match” of the underlying network components.

To that end, this paper makes the following contributions: (i) a distributed information model for IoT, network functions and cloud resources that maintains resource relationships, in particular, connecting resources across sub-networks; and (ii) a framework named HINC (Harmonizing IoT, Network functions and Clouds) for resource management, through which users can implement management actions, such as, reconfiguring IoT devices, defining/redefining relationships among IoT resources, and implementing network slicing. Throughout this paper, we demonstrate HINC via a proof of concept prototype. HINC aims to be easy-to-use for different users, and to support large-scale cloud-based IoT systems.

The rest of this paper is organized as follows. The next section presents the background and our approach. Section III describes our distributed information model. The HINC framework is described in Section IV, while our experiments are presented in Section V. Related work is discussed in Section VI, and Section VII concludes the paper and outlines future work.

II. BACKGROUND, RUNNING EXAMPLE AND APPROACH

A. Background

From the resource management perspective, the typical system in our study comprises three key types of components:
IoT elements, network management & virtualization services, and cloud services. Fig. 1 presents a schematic diagram of the same. We focus on a general model of blending IoT resources at the edge with cloud services in (centralized) data centers [1]. Things are either network-connected physical objects that sense their immediate environment and transmit data, e.g., sensors that determine the temperature, or physical objects interfacing with network-connected sensors monitoring the objects. This data transmission occurs at a particular frequency programmed in the sensor. The data transmission is sent to a gateway over a (local) network. Things also either act as actuators or interface with actuators controlling the Things. Thus, Things can be controlled via remote actions. Gateways are internet connected objects or services that integrate data from sensors, interface with actuators to control Things and support certain types of execution as in cloudlets [5]. With new concepts of virtualization at the edge, gateways might be referred as software-defined gateways [1], [6]. They then process, collate and perform other operations on the data before transmitting it to a cloud data center for eventual storage. In the cloud, the data is then processed and stored.

There are different types of networks between Things and gateways, gateways and gateways, and gateways and cloud data centers. The types of networks can vary widely, ranging from fixed to changing infrastructure, and fixed to changing Things. For our paper, we assume that the underlying physical infrastructure is fixed, although it can be separated into sub-networks via network slicing. We also assume that the Things are dynamic, comprising stationary and mobile Things; examples of the latter being users’ smartphones or wearables. For most IoT-based applications, this is a reasonable assumption, since physical infrastructures for IoT usually does not change frequently, whereas their usage could vary.

B. Impact of Virtualization on IoT Resource Management

The key technology that is driving IoT adoption is virtualization at various levels: at the gateway, the network, and the cloud center. While virtualization at cloud centers is well-known, research on software-defined gateways and network function virtualization is just starting. Software-defined gateways not only provide high-level abstractions to deal with complex underlying sensors and actuators but also enable virtualized environments for executing applications inside gateways [6]. Software-defined gateways would be one of the core resources that could be “sliced” and provisioned for different applications. Another core resource is defined in the network functions virtualization (NFV) [7] piece of Fig. 1. Today’s networks rely on software to configure the network, leading to network virtualization. From the IoT point of view, network virtualization offers the opportunity of dynamically rerouting transmitted data from Things to appropriate cloud storage via varying network paths. Network virtualization also offers the possibility of network slicing [4], i.e., (virtually) separating the network into different sub-networks, with each sub-network responsible for transmitting certain types of data. Network slicing also enables network operators to provision cloud resources.

C. Motivating Example

To illustrate our ideas, we use an example of emergency disaster response involving multiple victims, and requiring the coordination of multiple agencies. Examples could be a fire in a crowded movie theater, or a multi-car crash on the highway. Such a scenario would require the following:

- Emergency response services would need to be composed on the fly and involve a combination of the following (but not limited to): provisioning ambulances/fire engines, alerting doctors, alerting hospitals, and identifying optimal routes for emergency vehicles.
- Victims would be identified and their conditions monitored either by wearables.
- The information from the victims, along with information sent by emergency responders, would inform the appropriate treatment protocols to be administered to the victims on the fly during their transit to the hospital.
- Additionally, traffic sensors could point to the most optimal route that emergency vehicles can take.

Such a scenario would require all the components from the physical resource layer depicted in Fig. 1 for, e.g., in a short period of time. For executing such an example, of special interest here would be the dynamic provisioning of a virtual resource layer (like in the middle layer of Fig. 1), including a dedicated sub-network for this set of emergency responders, so that we can coordinate different operations among responders, networked hospitals and doctors. Hence, several components, such as shown in the management layer of Fig. 1, must be able to manage the resource information to support the provisioning of resources and the execution of applications atop these resources.

D. Approach

In our approach, we let different sub-networks (IoT, NFV, and clouds) to utilize their own models but they support the distributed resource information through monitoring plug-ins.
and information transformers to map link diverse types of information. In brief, our approach consists of the following key aspects:

- We permit the modeling of diverse types of (stationary and mobile) Things (i.e., resources) in a distributed IoT network, while representing the relationships among the resources as a first class entity in our model.
- Our model is distributed and decentralized, i.e., it permits the representations of parts of the IoT network, which can then be composed together by suitably modeling inter-resource relationships across the parts.
- We integrate all the various aforementioned sensors/actuators and cloud services consistently, by leveraging resource relationships, so that a comprehensive information model for the IoT network can be generated.

III. DISTRIBUTED RESOURCE INFORMATION MODEL

A. Existing resource models and APIs

Several resource models and accessing APIs are for IoT, network functions, and clouds, for example, shown in Table I. We make an assumption that these providers have their own models and APIs, and we harmonize information from the diverse type of services and service providers. Many APIs are based on REST, as also depicted in Table I. Therefore, we will focus on existing providers with REST API in our work.

B. Information Model

Our information model is depicted in Fig. 2. At the heart of our model are the Virtual IoT Resource elements, which are used to capture information about software-defined gateways and their capabilities, and the Virtual Network Function Resource elements, which are used to capture edge-to-edge and edge-to-cloud network function virtualization capabilities. These core elements are linked to the Physical Resources, such as things, sensors and actuators, and to the Cloud Services, such as virtual machines, data services and data analytics.

In our model, the physical resources do not directly connect to cloud but via software-defined gateways. In such a view, the virtual resources become an intermediate layer to hide the complexity of things. We explain the key elements in our information model in the following:

1) Virtual IoT Resource: To raise the abstraction level, we view heterogeneous types of physical resources as provided via different domain models. One or more physical resources are amalgamated into a virtual IoT resource (interchangeably referred to as simply virtual resource), and provide sets of Capabilities. By this sense, a virtual resource is a container of a set of physical resources with common characteristics. An example of a virtual resource could be the set of traffic sensors in a suburb, which could determine the average traffic flow. Hence, a virtual resource could comprise particular DataPoints which enable it to calculate average traffic flow from the various readings from the sensors. The ControlPoints capture information about how to control the resources, such as to increase the data rate of the sensors. The capability contains references to external APIs, which actually implement and enforce interactions with the resources.

Such a sensor virtualization approach is also in line with good practices for building cloud-based IoT systems by leveraging software-defined machines concepts [1]. The capability of each virtual resource can be expressed in terms of the data it can handle, the services it exposes to the external world and the environment it provides for applications.

The model supports heterogeneity of physical resources, e.g., a data point can tell the body temperature regardless of the types of underlying device. Moreover, the capabilities link with physical resources via resourceID to maintain important information, e.g., the location of the device. This model can deal with the dynamicity of underlying physical resources by hiding unnecessary details from external applications.

2) Network Functions: Virtual resources are also connected to each other via inter-resource relationships. Any pair of virtual resources connected to each other can belong to the same sub-network or different sub-networks. Leveraging the concept of network slicing, we manage a sub-network as Network Function Service which is provisioned by a Network Provider via a set of Virtual Network Functions. Additionally, each network could comprise multiple Cloud Services belonging to a Data Center. Each cloud service would implement various functions, such as load balancing and message queueing (the discussion about them is outside the scope of this paper).

3) Resource Relationships: Since relationships among resources are crucial for our information model, we present a detailed treatment here. We identify the following:

- Containment: A resource $R_i$ is contained in another resource $R_j$ if $R_i$’s functionality is required to complete the description of that of $R_j$. We have separately called out the containment of several physical resources within a virtual resource. However, even virtual resources could be contained within each other, e.g., traffic flow virtual sensor for a suburb contained within a traffic flow virtual sensor for a city.

- Composition: Resources $R_i$ and $R_j$ are said to be composable if the data generated by both resources can be combined in a semantically meaningful way. What is “semantically meaningful” is to be defined by the user. One example of two composable resources could be two virtual sensors, each modeling traffic flow for adjacent suburbs in a city.

- Dependency: is a special type of relationship among resources $R_i$ and $R_j$, which could be neither in a containment nor composition relationship. This relationship is user-defined, and signifies that data from $R_i$ would be incomplete without data from $R_j$. For example, a dependency between traffic flow data and accident data signifies that analyzing traffic flow would be incomplete without the effect of accidents on traffic flow.

These relationships can be further qualified depending on whether the two resources in question belong to the same sub-network or not. Hence, a resource relationship $\text{REL}(R_i, R_j)$ is defined as $\text{REL}(R_i, R_j) = (R_i, R_j, Type, N_i, N_j, Att)$,
TABLE I

<table>
<thead>
<tr>
<th>Provider</th>
<th>Category</th>
<th>API(s)</th>
<th>Information models</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIWare Orion</td>
<td>IoT</td>
<td>RESTful (NGSI10), one-time query or subscription</td>
<td>High level attributes on data and context</td>
</tr>
<tr>
<td>FIWare IDAS</td>
<td>IoT</td>
<td>RESTful for read/write custom models and assets</td>
<td>Low level resource model catalogs</td>
</tr>
<tr>
<td>IoTivity</td>
<td>IoT</td>
<td>REST-like OIC protocol, support C++, Java and JavaScript</td>
<td>Multiple OIC model</td>
</tr>
<tr>
<td>OpenHAB</td>
<td>IoT</td>
<td>RESTful for query and control IoT resources</td>
<td>Low level resource model catalogs</td>
</tr>
<tr>
<td>OpenDayLight</td>
<td>Network functions</td>
<td>Dynamic REST generated from Yang model (model-driven)</td>
<td>Low level resource model catalogs</td>
</tr>
<tr>
<td>OpenLatent</td>
<td>Network functions</td>
<td>RESTful for network service description</td>
<td>FESI MANO v1.1 data model</td>
</tr>
<tr>
<td>OpenStack</td>
<td>Cloud</td>
<td>RESTful, multiple languages via SDK, OCCI, CIMI</td>
<td>OpenStack model, OCCI, CIMI</td>
</tr>
</tbody>
</table>

Fig. 2. Main elements of the distributed resource information model

where Type refers to relationship type, $N_i$ and $N_j$ refer to the respective sub-networks (which could be the same or different) and Att is a set of attributes elucidating the relationship.

C. Resource Information Integration

Our resource information model is designed to be extensible since it needs to cater to multiple underlying IoT devices, network functions, and clouds, and especially since the types of IoT devices becoming available nowadays are increasing rapidly and also provided by diverse providers. Fig. 3 depicts how our management system would integrate specific resource providers into our resource information model.

Fig. 3. A flow of integrating specific resource management services

IoT, network functions, and cloud infrastructures expose APIs that can be accessed and queried. For IoT, we focus at the gateway level, where many devices are integrated and managed by specific information providers. The Resource Driver explores the available resources by interacting with specific resources providers, e.g., to retrieve the list of sensors/actuators on a device via discovery functions. The low-level information from specific providers is then processed to construct its virtual resources and save to our local repository. This model is then merged with the existing distributed resource information model using the information validator (which validates and performs error correction, if needed) and the capability extractor. In particular, the capability extractor works on mapping the information from the IoT devices into the format used to store the resource information model in the repository. For this, existing IoT vocabulary databases, e.g., IOTDB², can be used.

To give an example of how resources information can be integrated, we examine the following two models. First, the model of OpenIoT in Listing 1 is structured with metadata of physical devices (e.g. gateways) and a list of assets representing logical resources (e.g., sensors connected to the devices). Second, Listing 2 shows information from the OpenHAB framework running on the gateway, which integrates multiple resources, and provides data as per its domain model for each type of resource, e.g. the location.

Listing 1. Excerpt of a body temperature sensor from OpenIoT service

```json
"data": {
"DeviceProps": {
"commandURL": "http://.../services/OpenIoT/assets/..",
"lastIP": "195.97.103.225",
"commands": true
},
"asset": {
"name": "00:3b:B6:BodyTemperature",
"description": "asset model protocol" },
"model": "SENSOR_TEMP",
"registrationTime": "2015-04-16T15:39:58Z",
"status": "Active",
"sensorMetaData": {
"sens": {
"dataType": "BodyTemperature", "unit": "Celsius",
"rate": "10"
}}}
```

Listing 2. Excerpt of data of a via OpenHAB provider

```json
{ "type": "LocationItem", "link": "http://localhost:8080/rest/items/DemoLocation" }```

²https://iotdb.org
We can now extract the relevant information from both models and construct the virtual resource (see Listing 3). Assuming the geolocation sensor is linked and shows the location of the patient, the gathered information is extensible to attach multiple schemas to the capability, e.g. to know the location related to the data point or the status of the sensor. This extension is important to capture diverse types of information without sticking to a fixed model, also to support complex queries.

Listing 3. Virtual IoT resource information
```
"SoftwareDefinedGateway": {
"uuid": "5ba60...", "name": "gateway1",
"capabilities": [],
"capabilityType": "DataPoint",
"dataType": "BodyTemperature", "measurementUnit": "Celsius",
"resourceID": "00:3b:B6:BodyTemperature",
"extra": []
},
"sensor": {
"provider": "OpenIoT",
"status": "active"
},
"capabilityType": "ControlPoint",
"name": "changeRate",
"resourceID": "00:3b:B6:BodyTemperature",
"description": "change sensor rate",
"reference": "http://.../services/OpenIoT/assets/..",
},
}
```

Similarly, we gather the network and cloud information from network and cloud providers. Listing 4 shows an excerpt of network service information compatible with the ETSI MANO data model [8], and a cloud service. Such end-to-end information supports interoperability among different subsystems, which is crucial for our emergency response running example.

Listing 4. Excerpt of Network and cloud service information
```
"NetworkFunctionService": {
"name": "v4perf-server",
"vdu": { "vmInst": "ubuntu1", "provider": "openstack" },
"type": "v2perforder",
"networkFunction": { "vnf": { "vnInfo": "vn1", "vnfPort": "vnf1", "vnfId": "vn1", "vnfPortId": "vnf1" },
"vnf": { "vnfPort": "vnf1", "vnfId": "vn1" },
"vnInfo": { "vnfId": "vn1", "vnfPort": "vnf1", "vnf": "vn1" } }
},

"CloudService": {
"type": "storage", "provider": "openstack", "attributes": {
"capacity": "1 TB"
}
}
```

IV. ARCHITECTURE AND PROTOTYPE

A. Architecture

We introduce the architecture of our distributed IoT resource management system, HINC (Harmonizing IoT, Network functions, and Clouds) in Fig. 4. The key part of our architecture is the Local Management Service; it is deployed on a gateway or a network station that interfaces with the Information Providers to manage the resources. Multiple drivers enable unified access to providers, such as querying or controlling resources. For querying resource information, the output of the driver is passed to information translators to export the capabilities as per Section III-B. In our architecture, Information Providers are external services who expose the resources and capabilities via sets of APIs. We do not interface directly with particular resources, but via existing provider APIs.

In the cloud, the Global Resource Service coordinates with multiple Local Resource Services, and also maintains the inter-resource relationships; clients (human or automated software) interact with the Global Resource Managers to access resource-related information. Also in the cloud, the Relationship Manager models and manages the inter-resource relationships. Each such (global) query is further translated into local queries at each gateway, with each local query applicable to the IoT resources connected to the gateway in question. Each Local Management Service will also implement actions on the IoT resources, through its interfaces to specific providers/third-party APIs, in response to instructions from the Global Management Service.

B. Distributed Resource Information Requests and Responses

We are working towards a novel protocol of information-centric end-to-end management of resources [3]. The first step is to identify the resource by a semantic naming scheme (attribute-base) which enables different routing techniques in future implementations. The naming scheme is supported by our information model (Section 2). In this paper, we elaborate the routing capability of the AMQP. Our purpose is to route the queries from Global Management Service to Local Management Services based on the information semantics, as follows:

- The Local Management Service queries providers that it interfaces to and subscribes to a set of topic accordingly. E.g. for a provider providing traffic data in districts 1, the set of topics can be TrafficData.*, TrafficData.District1.*.
The Global Management Service, when receiving a query from a user, will generate the topics based on the query, and push a message to such topics. The responses from any Local Management Service are gathered via a temporary topic related to the query. This naming is also used to identify all types of resources in network functions and cloud services.

C. Prototype

We are currently implementing our HINC framework using Java as per our distributed architecture from Section IV. We are currently implementing multiple plug-ins or adapters to interact with multiple providers (See Table I). The communication between Local and Global Management Services is implementing using RabbitMQ, a message-oriented middleware. Global Management Services provides well-defined REST interfaces, and Local Management Services run as lightweight daemons. We use OrientDB to implement the repository.

V. Experiments

A. Experimental testbed

Due to the difficulties to have a real testbed with full hardware and software for IoT, network functions and cloud, we use Amazon EC2, CloudAMQP and a set of physical machines with various software to emulate a realistic testbed. Fig. 5 shows the deployment of the testbed. The in-lab part is deployed on a server (8 CPU-i7, 3.60GHz, 32GB RAM) with: (1) the edge with emulated sensors and gateways deployed on 100 docker containers in total, and (2) the network functions and cloud resources are emulated by other docker containers. In our in-lab testbed, we use Weave (https://www.weave.works/) to emulate the routers. At the cloud part, we deploy the HAProxy (http://haproxy.org/) Load Balancer and an event processing service (internally developed web application on top of Apache Tomcat (http://tomcat.apache.org/)) to process the sensor data, detect particular events and save the collected data into a Cassandra cluster.

In the testbed, we deploy one Local Management Service on every emulated gateway. We use an emulated provider that can generate many sensor’s metadata. Each Local Management Service interacts with one provider to update information to the database. When receiving a query, information is read directly from the database. From the resource information we extract the following control capabilities: create/remove router; attach/detach a gateway with a router; create network route; reconfigure the gateway in terms of data points (change data transmission rate of a sensor), control points (start/stop IoT devices) and connectivity (change the gateway protocol). These capabilities are not enforced by HINC but by corresponding specific providers for us to emulate resource controls.

B. Reducing complexity in accessing and control resources

1) Scenario: Our system aims to support programmers and software-agents who need a better way to program and control the IoT resources. An example is we need to (i) query and change capabilities of a set of sensors, (ii) create a slice of network functions to sustain the data produced by sensors due to the change in (i), and (iii) increase cloud storage to process the sensor data. To overcome the complexity of low level information in the above-mentioned scenario, we provide easy-to-use APIs for developers and applications.

2) Examples: Listing 5 shows an example of Java code to increase the sample rate of all temperature data from those body temperature sensors with sample rate greater than 5 seconds (such as the one represented in Listing 1), to exactly 5 seconds. At the client, users build a template of data point, which can include extended models, e.g. the SensorProps to capture the sensor rate. Users then need to define the communication channel by using the QueryManager, from which they can create and send a query and to obtain a list of data points. To send a control back to the resources, a control point (which is available and associated with the data point) wraps the change rate operation and transfers it to underlying management services to execute. Similarly, the code shows how the user can query network and cloud services. The programming objects here hide the complex query and communication details from the user.

Listing 5. Controlling a slice of sensors, network functions and cloud services

```java
// specify the sensors whose sample rate needs to be changed
DataPoint template = new DataPoint('BodyTemperature');
SensorProps sensorProps = new SensorProps();
template.getExtra().add(new PhysicalResource(sensorProps));
QueryManager queryMng = new QueryManager('ampq://10.99..');
List datapoints = queryMng.queryDataPoints(template);

// observe the resource and send back the control
for (DataPoint dp : datapoints) {
  DataPointObserver obs = new DataPointObserver(dp) {
    public void onChange(DataPoint newVal)
      SensorProps props = newVal.getExtra('SensorProps');
      if (props.getRate() > 5) {
        props.setRate(5);
        queryMng.sendControl(control);
      }
  }
  queryMng.sendControl(control);
}
```

3The source code, further examples, and evaluation results are accessible at http://SINCConcept.github.io/HINC/.

Fig. 5. Experimental Testbed

Listing 5. Controlling a slice of sensors, network functions and cloud services

```
// create the template of the query
NetworkFunctionService nfsTemp = new NetworkFunctionService();
nfsTemp.getQuality().setBandwidth("16 GB/s");
CloudService cloudTemplate = new CloudService('storage');
```
C. Experiments on scalable resource management

1) Query time by the number of gateways: In this experiment, we increased the number of gateways in our deployment and recorded the response time of several queries. The Global Management Service broadcasts a query to gather information from all the gateways. The number of responses obtained by the Global Management Service equals the number of gateways. We observe two deployment: (1) the in-lab where all the services are deployed internally, and (2) the multiple sites where services are at different location.

Fig.6 shows the latest response time of a query in different in-lab deployments. We see that, overall, the response time increases as the number of gateways increases. By using distributed communication, we can see the processing times at Local Management Service increases slightly irrespective of number of gateways. However, increasing the number of gateways does increase the processing time at the Global Management Service. In contrast, Fig.9 shows the multiple sites deployments, where transmission time is major and increases according to the number of gateways. We also see the processing time at Global Management Service is unpredictable because the VM on Amazon EC2 is a free tier with which resources are shared. Global service processing time is lesser than the in-lab experiment because the network latency reduces the continuous workload on Global Management Service.

When the Global Management Service sends a query to multiple Local Management Services, the responses are gathered via multiple messages. Fig.7 and Fig.10 show the response time of the first and the last message which arrived and are processed at the Global Management Service. The difference between the two bars shows that the total query time increases appreciably when many messages are queued up for processing. This is due to the Global Management Service gathering a large amount of data into a single location, due to its centralized nature. Hence, a centralized resource management would not be scalable.

2) Query time by the number of underlying resources: In this experiment, we increased the number of sensors at 100 and increase the number of sensors connected to each gateway. Fig.11 shows that response time increases substantially as the number of sensors increases in each gateway. Compared with the previous experiments, we see that the transmission time increases more to transfer more data items. This again demonstrates the non-scalability of centralized IoT resource management.

3) Query time from different deployment: In this experiment, we setup two sites of IoT resources with Local Management Service in India and in Austria (Fig.5). These services communicate via a RabbitMQ deployed on the cloud in Ireland. As a result, Fig.8 shows that the query time depends on the location of these services. E.g. both Global and Local service in India got the highest response time due to the longest communication route, while both services in Austria got the shortest route and the fastest response time.

The above experiments show that our distributed approach does ensure scalable resource management even as the scale of the IoT system – expressed in the amount of data transmitted, number of sensors or number of gateways – increases.

VI. RELATED WORK

IoT resource modeling approaches: Several studies have introduced models to manage IoT resources. In [9], relationships between physical things, sensors, actuators and devices are described. Oteafy et al. [10] specify the basic model for IoT objects and the way that resources can be shared by multiple applications. Zhang et al. [11] model the IoT objects as resources comprising mapping with services to provide the necessary management functions. Benazzouz et al. [12] propose in project ClouT the information model for IoT with resource, service and device blocks. Zhang and Meng [13] propose a multi-dimensional IoT resource model comprising functionality, spatial, temporal and usage priority. All these approaches do provide rather detailed models for IoT devices, but none of them consider a distributed approach accommodating the underlying topology as in our paper. Ranjan et. al [14] propose a resource overlay on top of cloud and IoT infrastructure in order to facilitate better resource management. Lopez et. al [15] consider the complex issue of resource management in large IoT networks, and propose a clustering approach to subdivide the network for ease of management. We view [15] as our complementary work, since their approach can be employed by users for network configuration, prior to implementing our resource modeling and provisioning approaches.

Information models for network virtualization: The expected emergence of 5G [4] has raised interest in applying network slicing for optimal network utilization of IoT applications. This has given rise to several information models for network virtualization. As part of the EU project NOVI, van der Ham et. al [16] present a set of information models for modeling virtualized network resources and services. Some notable models for managing network are NetJSON (http://netjson.org/rfc.html) and YANG (http://www.netconfcentral.org/yang_docs). We view the works presented in [16], [17] to be complementary to our work, and we will be leveraging them to enhance our future work through the incorporation of resource provisioning using virtualized network resources.

IoT resource management frameworks: The OpenIoT [18] project has developed a middleware for extracting information from sensors without knowing what exact sensors are used. Several solutions for resource discovery, management and provisioning [19], [20] have been developed, but they do not address how IoT resource management can be implemented on a large scale, especially along with network virtualization.
In this paper, we have addressed the crucial research issue of managing diverse resources in IoT systems at large scale, and we present a highly extensible management framework that we call HINC, for harmonizing IoT, networks and clouds. We have shown that our framework is extensible enough to accommodate information from various IoT resources with different data models. The distributed nature of our approach also facilitates scalable IoT resource management. Although this work is at an early stage, the result of this work would foster further research into areas as such as end-to-end IoT resource provisioning and configuration. We have also illustrated our work via a realistic running example in the emergency response domain as well as the performance and scalability of HINC via an initial set of experiments on a proof of concept prototype.

We will scale up our prototype and testing it on larger real-life usage scenarios. We will also be running more detailed experiments to evaluate how our approach can handle dynamism in the underlying IoT infrastructure.

VII. CONCLUSIONS AND FUTURE WORK

In this paper, we have addressed the crucial research issue of managing diverse resources in IoT systems at large scale, and we present a highly extensible management framework that we call HINC, for harmonizing IoT, networks and clouds. We have shown that our framework is extensible enough to accommodate information from various IoT resources with different data models. The distributed nature of our approach also facilitates scalable IoT resource management. Although this work is at an early stage, the result of this work would foster further research into areas as such as end-to-end IoT resource provisioning and configuration. We have also illustrated our work via a realistic running example in the emergency response domain as well as the performance and scalability of HINC via an initial set of experiments on a proof of concept prototype.

We will scale up our prototype and testing it on larger real-life usage scenarios. We will also be running more detailed experiments to evaluate how our approach can handle dynamism in the underlying IoT infrastructure.

REFERENCES


Fig. 6. Query time by number of gateways, in-lab

Fig. 7. Gateway’s response time variability, in-lab

Fig. 8. Query time from several distributed sites

Fig. 9. Query time by number of sensors, sites

Fig. 10. Gateway’s response time variability, sites

Fig. 11. Query time by number of sensors, sites