2017 IEEE 7th International Symposium on Cloud and Service Computing

A Dynamic Module Deployment Framework for M2M Platforms

Bing-Liang Chen, Shih-Chun Huang, Yu-Cing Luo, Yeh-Ching Chung and Jerry Chou
Department of Computer Science, National Tsing Hua University, Taiwan
Email: {cbl920809,sjhuang,ycluo}@sslab.nthu.edu.tw
{ychung, jchou}@cs.nthu.edu.tw

***Abstract*—IoT applications are built on top of M2M platforms which facilitate the communication infrastructure among devices and to the clouds. Because of increasing M2M communication traffic and limited edge network bandwidth, it has become a crucial problem of M2M platform to prevent network congestion and service delay. A general approach is to deploy IoT service modules in M2M platform, so that data can be pre-processed and reduced before transmitting over the networks. Moreover, the service modules often need to be deployed dynamically at various locations of M2M platform to accommodate the mobility of devices moving across access networks, and the on-demand service requirement from users. However, existing M2M platforms have limited support to deployment dynamically and automatically. Therefore, the objective of our work is to build a dynamic module deployment framework in M2M platform to manage and optimize module deployment automatically according to user service requirements. We achieved the goal by implementing a solution that integrates a OSGi-based Application Framework(Kura), with a M2M platform(OM2M). By exploiting the resource reuse method in OSGi specification, we were able to reduce the module deployment time by $50\% \sim 52\%$. Finally, a computation efficient and near-optimal algorithm was proposed to optimize the the module placement decision in our framework.**

***Keywords*—Internet-of-Things, Machine-to-Machine, Module Deployment, Placement Optimization**

I. Introduction

Internet of Things (IoT) is a fast emerging technology. The number of IoT devices deployed in 2016 is 30% more compared to 2015, and the number is expected to increase to 20.8 billion in 2020 [1]. These IoT applications, such as smart cities and smart homes, are mostly built on a network systems connecting devices with cloud services, so that intelligent decision and action can be made by using the powerful servers in the cloud to analyze the big data collected from devices. However, due to the growing number of heterogeneous devices and limited network bandwidth at the edge or local networks, the huge amount of incoming data steams generated by IoT devices can easily lead to severe network congestion and surging servers loads. Hence, minimizing network traffic of IoT applications is a critical research issue.

In recent years, many efforts have been made from network architecture to programming model for the coordination of IoT applications. Such as, the Web of Things (WoT) [2] approach is a programming model that allows real-world objects to be part of World Wide Web. In other words, accessing to the real-world object is like accessing to the website. Using the RESTful APIs and link directed to any object on the Internet, we can access the service on the object. This way simplifies the data exchange between a large number of objects and service consolidation in each object. The WoT approach starts with the application layer of the network architecture to hide the underlying network complexity. When developers create a new application, they only need to consider how to compose a number of web services as so-called mash-up. In telecommunication, Machine to Machine (M2M) [3] is an upcoming application in the next-generation communication. Hence, M2M platforms are proposed and developed to address the challenge of communication problems across heterogeneous IoT devices. Although WoT [4] provides an programming model for developing IoT applications and M2M platform supplies a network framework for communicating IoT devices, they have not paid much attention on the issue of managing the IoT network traffic.

Traditionally, we rely on upgrading the capacity of network and server to handle the growing network traffic from IoT devices. But due to the bursty traffic behavior of IoT applications, optimizing for system peak load could significantly lower the average resource utilization. Hence, recently, dynamic service deployment [5] has drawn increasing interests from the research community as a more promising solution. The idea is to dynamically deploy service module in the M2M platform by user requests. These modules are deployed close to the edge of networks and shared among users, so that the overall robustness of the application is improved, and network traffic loading is reduced. Many challenges remain for this recently proposed idea. First, because of the limited resource of the devices, and diverse service functionality, a software architecture is needed to manage these service module, and coordinate them across devices. Second, module deployment occurs at runtime in an on-demand manner, so the deployment time has to be short, and critical to service quality. Finally, because modules can be shared among users, and users can be scattered around the network, it is not trivial where to deploy the requested modules under resource and service quality constraints.

In this paper, we firstly propose a novel M2M platform based on OM2M/Kura that can dynamically push service module (i.e., program) to the device nodes in the M2M platform. The low-end node (e.g. edge node and middle node) is applied to support remote deployment, and the high-end node (e.g. infrastructure node) is equipped with more computing capability and storage space to management all IoT applications in the M2M platform. As a result, the complicated application can be processed locally rather than push all data

978-1-5386-5862-8/17 $31.00 © 2017 IEEE
DOI 10.1109/SC2.2017.37

to the server for processing for reducing the network traffic and the load of data centers.

Then, we further introduce a resource reuse method which uses the OSGi specifications [6] to reuse the M2M resource in our M2M platform implementation. M2M resource is a set of resource data model defined by OM2M platform. Using the resource reuse method can pre-loading the M2M resource that commonly used by all the service module. Therefore, the startup time of our dynamic module deployment can be significantly reduced by 75%.

Finally, we propose a module placement algorithm to control the deployment decision of our implemented framework. The goal of our placement problem is to maximized the number of satisfied requests subjects to the service quality requirement (i.e., network distance between users and service module) from users and resource capacity constraints on devices. We formulate the dynamic module placement problem as an ILP optimization problem, and propose a heuristic module placement algorithm, called MPA. By comparing to the optimal solution from using an ILP solver(CPLEX), we prove MPA is a computational efficient and near-optimal algorithm.

The remainder of this paper is organized as follows. In Section 2, we describe the standards for M2M and the Internet of Things, including ESTI SmartM2M and oneM2M, and discuss related works about deployment platform. The architecture and implementation of our dynamic module deployment M2M platform is detailed in Section 3. The dynamic module placement problem formulation and algorithms are given in the Section 4. Section 5 is the evaluation results of our proposed M2M platform and algorithm. Finally, Section 6 is the conclusion of our work.

II. Related Works

To solve the high vertical fragmentation of current M2M markets and the lack of the standards, the ETSI initially released a set of specifications named the ETSI SmartM2M standards for a common M2M service platform [7]. In July 2012, a global M2M standard organization named oneM2M was formed and employed a simple horizontal platform architecture that fits within a three-layer model comprising applications, services and networks. The oneM2M standards is the same as the ETSI SmartM2M with initial focus on the Service Layer which resides between the Application Layer and the Network Layer. The Common Services Entity (CSE) represents an instantiation of a set of c̈ommon service functionsöf the M2M environments as shown in Fig. 1. The services provided by the Common Services Layer in the M2M system are referred to as Common Services Functions (CSFs), such as the Application and Service Layer Management, the Communication Management and Delivery Handling, the Data Management & Repository, and so on. The CSFs provide services to the Application Entities (AEs) via the Mca reference and to other CSEs via the Mcc reference point, and interact with the Underlying Network Service Entities (NSEs) via the Mcn reference point [8].

Several studies [9], [10], [11] present the design and implementation of platforms which can offload tasks from end devices to edge servers. In [9], Cloudlet placed a powerful machine near to users for reducing service latency. Users

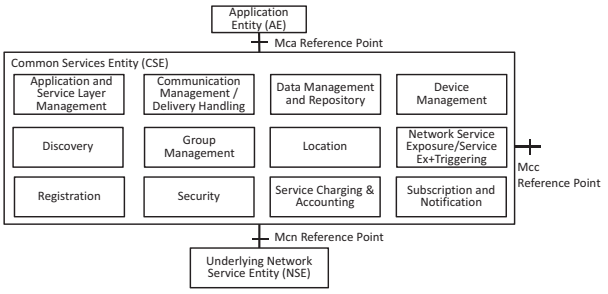


Fig. 1. Common Services Entity(CSE)

can push virtual machine images to the machine to perform their tasks. Unlike our work, Cloudlet does not consider the limited capacity on devices. ParaDrop [10] is a fog device implemented on end-user gateways. ParaDrop is a service deployment framework for utilizing residual capacity on network gateways, but it does not consider the dynamic module deployment problem. [11] proposed a module placement algorithm similar to us, but it didn't discuss the mechanism and implementation of dynamic module deployment.

III. Module Deployment Mechanism

This section describes the design and implementation of our dynamic module deployment framework. First, we briefly describe the existing M2M framework architecture, and explain how we extended it to support the functionality of dynamic module deployment. Then, we use a step-by-step workflow to detail the implementation of our dynamic module deployment mechanism. Finally, we minimize the module deployment time of our framework by implementing a resource reuse method through the OSGi specification.

A. Platform Architecture

Our work is based the OM2M platform [12], which is an oneM2M-compliant open-source M2M service platform. It supplies the software to enable the Common Service Entity (CSE). OM2M provides RESTful APIs to enhance interoperations with the CSEs. A modular architecture is proposed running on the top of an OSGi layer [13], making it highly extensible via plugins. Each plugin is a Common Services Function (CSF) that offers specific functionalities. A CSF can be remotely installed, started, stopped, updated, and uninstalled via the OSGi Equinox runtime. However, the operations for managing these CSE plugins have to be done manually on the running nodes.

To facilatate remote software package management, we also use Kura in our framework implementation. Kura is a Java/OSGi-based smart application container [14] that enables remote management of IoT gateways. It also uses the OSGi specification to allow for remote management of the IoT applications installed in Kura including their deployment, upgrade and configuration management. Moreover, Kura provides a wide range of services for simplifying the process of writing the IoT application such as Data Services, Cloud Services, I/O Services and etc. The services provided by Kura as shown in Fig. 2.

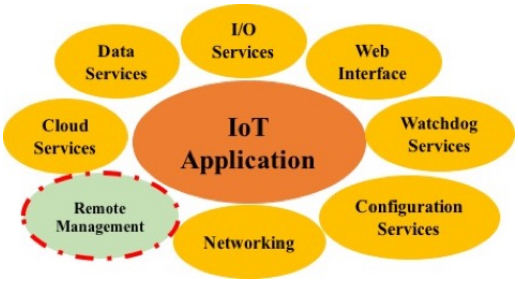


Fig. 2. Services provided by Kura platform

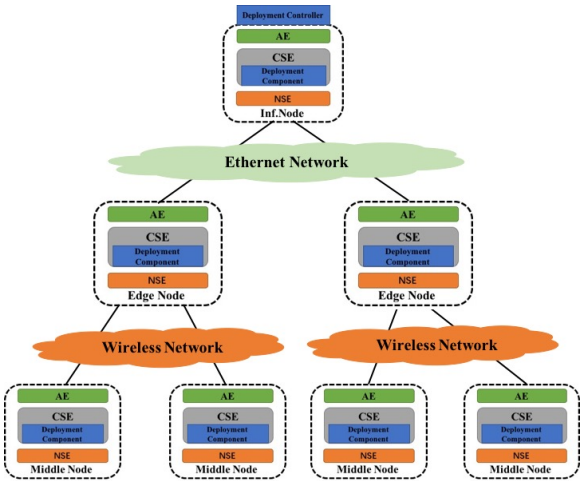


Fig. 3. The proposed M2M platform architecture

Fig. 3 shows the overall picture of our OM2M platform. Infrastructure node acts as the cloud platform or datacenter for IoT. Edge node is at the edge of telecom network, and it registers to the infrastructure node to manage the nodes which are under its resource tree. Applications, such as smart home, can be deployed on our edge node and not all data from the IoT applications needs to be sent to the cloud servers. Middle node refers to gateway or router, it registers to the edge node and manages the underneath heterogeneous devices. Application service nodes are the end-users who need IoT service to process or collect their data. We designed two different functional components to perform the deployment task. One is *deployment component*, which runs on every nodes in M2M platform to complete the task of module deployment. The other is *deployment controller*, which is at the top of M2M platform and makes the remote deployment decisions. The details of these components are given next.

1) Deployment Component: Deployment component is in charge of performing the deployment task on a node. We implemented this component based on the Remote Management Service in Kura, which enables remote IoT applications management including deployment management, upgrade management and remote access. In order to adapt this management service into our M2M platform, we modify the Remote Management Service into a Common Service Function (CSF) plugin package in the OM2M platform following the OSGi specification as showed in Fig. 4. Hence, the deployment component can be automatically started after the OM2M platform is launched. Then the deployment controller can remotely interact with the Remote Management Service of deployment component to perform dynamic module deployment.

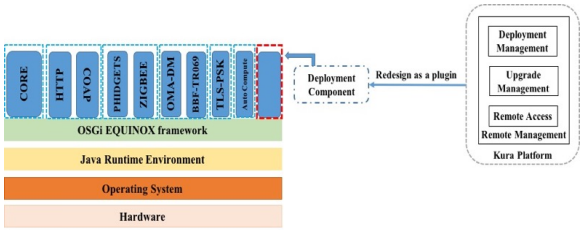


Fig. 4. The OM2M building blocks

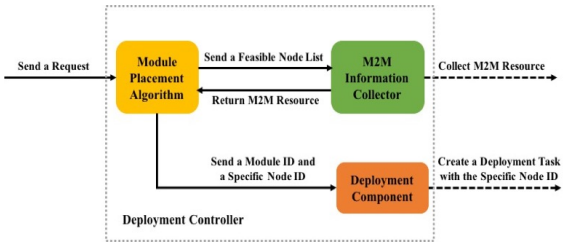


Fig. 5. The steps of module deployment flow in deployment controller

2) Deployment Controller: Deployment controller is at the top of the M2M framework. Its core function is to make placement decisions according to the current system states, and user service requests, and then interact with the deployment component to perform remote module deployment. Deployment controller is composed of a local deployment component, a Module Placement Algorithm (MPA), and an information collector as shown in Fig. 5. The information collector collects all the resource usage information, and service module requests in the system. After the controller receives service requests, it calls the Module Placement Algorithm to find a module placement plans that can satisfy the most number of service requests while minimizing the network traffic. Then, a set of deployment tasks are created to realize the placement plan. Each deployment task contains the require information for deployment components to perform a module deployment task on a remote node.

B. Module Deployment Flow in M2M Platform

This section introduces the details of module deployment flow between deployment controller and deployment component as showed in Fig. 6. First, the deployment component in deployment controller creates a deployment task based on the module ID and target node ID decided by the Module Placement Algorithm (MPA). Then, the deployment component sends a request using the RESTful API to find the target node location (In this paper, the node's location information refers to the ip address which is registered to the M2M platform) through M2M platform. After a node in M2M platform receives this request, it checks whether its ID is the same as the ID in the request. If the ID is the same, the node responses this request with its location information. Otherwise, the node forwards this request to the nodes under its resource tree. This searching process repeats until the target node responses to the controller with its location information. After receiving

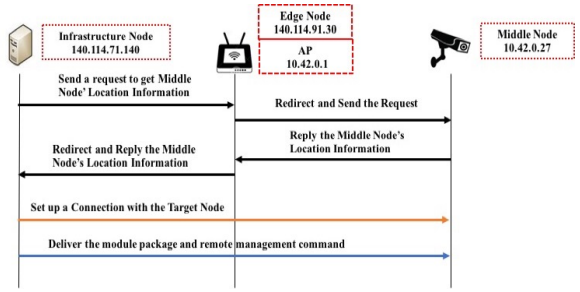


Fig. 6. The module deployment flow in M2M platform

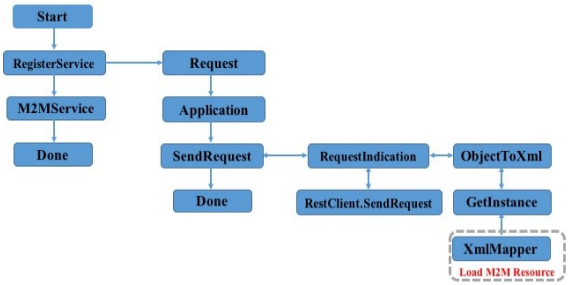


Fig. 7. The flow of deployment module startup

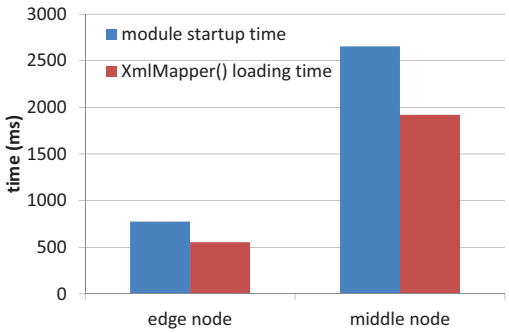


Fig. 8. The overhead analysis of deployment module startup time

the response from the target node, the deployment component in the deployment controller setups a end-to-end connection tunnel to the deployment component on the target node. Finally, module package and remote management command are sent to the target node for deployment.

C. Resource Reuse Method

After completing our proto-type system implementation, we made an effort to profile the system performance and further optimize the performance of our implementation.

Fig. 7 depicts the step-by-step code path of the module startup process in our framework. After detailed profiling the time spent in each step on both edge node and middle node, we found that most of the module deployment time was spent on loading a OSGi resource, called XmlMapper function. As shown in Fig. 8, the deployment module startup time is around 775.2 milliseconds and 2654.2 milliseconds in edge node and middle node, respectively. But more than 75% of the time was caused by the load time of XmlMapper function.

XmlMapper is a function used to covert M2M resource XML representation to M2M resource Java Object and vice verse. It has to be loaded by almost every Comon Service Functions. Therefore, we decided to implement a resource reuse method in our framework to make XmlMapper become a common OSGi resource. So it can be loaded when the OM2M node is initialized, and shared by all the dynamic deployed modules. As shown by our real testbed evaluation, the implemented resource reuse method can significantly reduce the startup time by 75%, and the total deployment time by 50%. Also, this method can reduce the network transfer package size of deployment software, and the memory usage on the deployment nodes. Therefore, this technique is critical to the performance and resource usage in our framework implementation.

IV. Module Placement Problem and Solution

After explaining the mechanism and implementation of our M2M system framework, we describe our module placement problem and algorithm in this section. Module placement problem is critical to the performance of our dynamic module deployment framework, because the placement decision can determine the service quality seen by the users, and the resource consumption of service module on the deployed nodes. Hence, this section first define and formulate the placement problem addressed in this paper, and then propose an efficient and near-optimal heuristic placement algorithm to make timely decision at runtime.

A. Problem Formulation

A M2M system like the one shown in Fig. 3 can be denoted as a graph with N nodes (i.e., either edge or middle nodes) and E edges connected between nodes. The computing capacity on each node can be varied, so we use C_n to denote the capacity of a node $n \in N$.

The placement problem formulated in this paper is to decide the deployment location of a set of modules M in order to satisfy a set of service requests R arriving at a given time interval. Each request $r \in R$ asks for some service module $m_r \in M$. A request is satisfied if and only if its requested module is successfully deployed in the M2M system at a feasible location.

To reflect the placement constraints in real systems, a module $m \in M$ can only be placed on a node n if the following three requirements are matched. We briefly describe these constraints, and their associated variables (1) The capacity constraint, that is a module can only be placed on a node with sufficient residual capacity. In other words, there can only be limited number of module deployed on each node. Hence we introduce U_m to be the resource usage of module m, and α_n to be the resource capacity already used on node n. (2) The feasibility constraint, that is a module can only be placed on certain nodes that are specified by the user request. Hence we introduce N_r to represent the set of feasible nodes for request r. (3) Finally is the service quality constraint, which is measured by the network distance between the users and service deployed node. The network distance is likely to cause higher request response time delay. Therefore, we introduce

L_m to denote the maximum network distance that can be tolerant for the user request of module m, and we use $d_{r,n}$ to denote the network distance (i.e., number of hops or latency delay) between from request r's user location to a node n.

The goal of our placement problem is to maximize the number of satisfied requests under the three constraints mentioned above. Let $x_{n,m} \in \{0,1\}$ be the boolean variable for our placement decision. If $x_{n,m} = 1$, then module m is placed on node n. Otherwise, module m is not placed on node n. Then, our problem can be formulated by the following equations.

$$\max \sum_{r \in R} y_r \tag{1}$$

$$st : y_r = \min(\sum_{n \in N_r} x_{n,m_r}, 1), \forall r \in R \tag{2}$$

$$\sum_{m \in M} x_{n,m} \times U_m + \alpha_n \leq C_n, \forall n \in N, \forall m \in M \tag{3}$$

$$x_{n,m_r} \times d_{r,n} \leq L_m, \forall n \in N, \forall r \in R \tag{4}$$

$$\sum_{n \in N_r} x_{n,m_r} \leq 1, \forall r \in R \tag{5}$$

$$x_{n,m} \in \{0,1\}, \forall n \in N, \forall m \in M \tag{6}$$

The objective function in Eq. (1) maximizes the number of satisfied requests y_r. Eq. (2) set y_r to 1 if the module of request r is placed on one of the nodes. Otherwise, y_r is set to 0. Eq. (3) guards the capacity constrains of every node $n \in N$. Eq. (4) guards the service quality constrain of each request $r \in R$. Eq. (5) guards the feasibility constraint, so that a module m_r of the request r can only be deployed on a node n, where $n \in N_r$. Eq. (6) represents our placement decision. The resulting formulation can be solved by existing ILP solvers, such as CPLEX [15], GLPK [16], or CBC [17]. Doing so, however, may be computationally intensive, as the problem is a variation of the NP-hard knapsack problem. Hence, we developed a heuristic algorithm as presented in the next subsection.

B. Module Placement Algorithm (MPA)

The design principles of our proposed Module Placement Algorithm (MPA) are described as follows. The MPA algorithm needs to make two decisions: (i) which request should be considered first and (ii) which node should be used for deployment first. More specifically, the MPA algorithm considers the request with shorter network distance (i.e., less latency delay). If the network distance of the requests are the same, the MPA algorithm chooses the request which consumes less resource capacity. After sorting the requests, the MPA algorithm iteratively selects the node for deploying the modules. Firstly, The MPA algorithm selects the node which is the closest to the end user who sends this request. If the distance between the feasible nodes and the end user who sends this request are the same, the MPA algorithm selects the feasible node which has the most capacity. It stops once all the requests are fulfilled or all resources are used. Fig. 9 shows the pseudocode of the MPA algorithm.

Line 1 decides the order of requests. The for-loop starts from line 2 iterates through all sorted requests and the for-loop

Algorithm 1 Module Placement Algorithm

```
1:  sort requests r ∈ R on L_m in the asc. order ; use U_m (also in the asc.
    order) to break ties
2:  for r ∈ R do  //iterate with the sorted requests
3:      for n ∈ N do  //iterate with the nodes
4:          if Eq. (1d) is satisfied then
5:              N_r ←N_r ∪ {n}
6:          end if
7:      end for
8:      sort nodes n ∈ N_r on d_{r,n} in the asc. order ; use α_n (also in the asc.
    order) to break ties
9:      for n ∈ N_r do //iterate with the sorted feasible nodes
10:         if Eq. (1c) is satisfied then
11:             X_{n,m} ←1
12:             C_n ←C_n-X_{n,m} U_m
13:             α_n ← α_n+X_{n,m} U_m
14:         end if
15:     end for
16: end for
```

Fig. 9. The pseudocode of the MPA algorithm

starts from line 3 iterates through all nodes in M2M platform. Based on the given requests, line 4 checks the violation of the constraint in Eq. (4) and line 5 decides the set of feasible nodes. And line 8 decides the order of feasible nodes based on L_m and α_n. The for-loop starts from line 9 iterates through all sorted feasible nodes. Line 10 checks the violation of the constraint in Eq. (3) and line 7 assigns the value to the decision variable . In terms of complexity, the MPA algorithm results in polynomial time, which is proved in Lemma 1

Lemma 1(Time Complexity). The MPA algorithm terminates in polynomial time

Proof. In terms of complexity, creating the sorted units list in line 1 has a complexity of $O(|R| \log |R|)$. The for-loop in line 3$\tilde{7}$ goes through nodes $n \in N$, which leads to a complexity $O(|N|)$. Creating the sorted units list in line 8 has a complexity of $O(|N| \log |N|)$. The for-loop in lines 9$\tilde{1}$5 goes through feasible nodes $n \in N_r$, which leads to a complexity of $O(|N|)$. Since the for-loop in lines 2$\tilde{1}$6 has $O(|R|)$ iterations, the overall time complexity of the MPA algorithm is $\max\{O(|R| \log |R|), O(|R|(2|N| + |N| \log |N|))\}$. Therefore, MPA algorithm has a polynomial time complexity.

V. Experiments

In this section, we use a real testbed to evaluate the benefit of resource reuse method, and then use a simulation to evaluate the complexity and optimality of Module Placement Algorithm (MPA).

A. Resource reuse method evaluation

1) Real testbed environment: Here, we describe the real testbed environment for evaluating our proposed dynamic module deployment framework, and the resource reuse method described in Section III. As illustrated in Fig. 2, our experimental environment has one infrastructure node, two edge nodes and four middle nodes. The infrastructure node was running on a regular PC machine. The two edge nodes were running on Intel Mini Computers, and the four middle nodes were running on Raspberry Pi. The network between infrastructure node and edge node is Ethernet network, and the network between infrastructure node and middle node is wireless network. The Intel Mini Computer enables the Wi-Fi connection from

TABLE I. SPEC OF NODES IN M2M PLATFORM

	Inf.node	Edge node	Middle node	App service node
CPU	2 core	4 core	1 core	1 core
Ram	4 GB	4 GB	512 MB	512 MB
Disk	128 GB	1 TB	16 GB	16 GB

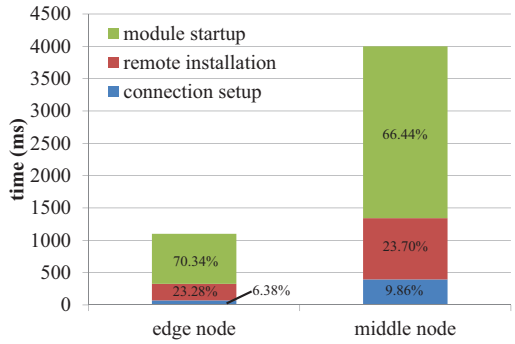


Fig. 10. The breakdown of module deployment time on edge node and middle node when resource reuse method is not applied

Raspberry Pi and the Raspberry Pi enables Wi-Fi or Bluetooth connection from devices. Finally, to generate service requests, we deployed a set of application service nodes connected to the middle node via either Wi-Fi or Bluetooth. All the spec of nodes is listed in Table I.

2) Results: The basic procedures of module deployment are composed of (i) Setup connection from the edge or middle node to the deployment controller on the infrastructure; (ii)Remote install module by transferring the module package from the module controller to the edge or middle node; (iii)Module startup on the edge or middle nodes as shown in Fig. 7. Fig. 10 shows the breakdown of module deployment time on edge node and middle node when resource reuse method is not applied. The edge nodes have more computing power than the middle nodes, so the time is shorter on the edge nodes than the middle node. But for both types of nodes, we can observe that majority of the time, close to 70%, is spent on module startup. As explained in Section III-C, this is mainly because of the time consuming operation for loading the XmlMapper function. On the other hand, the remote installation time is only about 23%, and the setup connection time is less than 10%.

After applying our resource reuse method to prevent XmlMapper function from being loaded repeatedly at runtime, we were able to reduce the module package size, and module startup time as shown in Fig. 11. Similar reduction results were observed for both edge and middle nodes. The module package size was reduced by 14.3%, which contributes to roughly 8% reduction in remote connection time, and less network bandwidth consumption. More importantly, the resource reuse method significantly reduce the module startup time by almost 75%, and reduce the total deployment time by more than 50%.

B. Module Placement Algorithm (MPA) Evaluation

1) Simulation setup: We use simulation to evaluate the performance of the Module Placement Algorithm (MPA). In the simulation, we designed three kinds of requests R which corresponds three kind of modules M. Each module

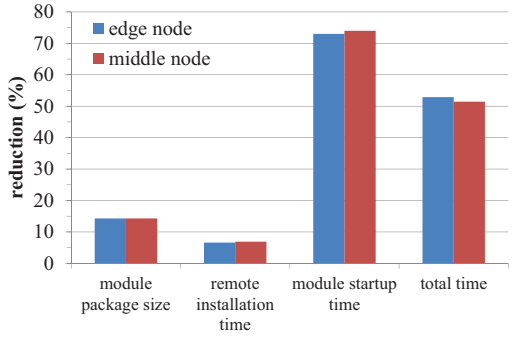


Fig. 11. The reduction of deployment time and package size after resource reuse method is applied

TABLE II. SPEC OF MODULES

	Module 1	Module 2	Module 3
Usage(U_m)	3	2	1
Latency(L_m)	1	2	3

M has its own usage U_m and latency L_m. All the specs of modules are listed in Table II. The resource capacity, C_n, of infrastructure nodes, edge nodes, and middle nodes were 6, 4, 2, respectively. By default, we consider the system has 50 nodes, either edge or middle nodes, and varied numbers of requests, $|R| = \{10, 15, 20\}$, are randomly generated for requesting one of three kinds of modules.

2) Results: The goal of MPA algorithm is to maximize the number of satisfied requests under the limited resource capacity of nodes, and the maximum latency requirement of service module. First, we show the MPA algorithm can achieve near optimal results by comparing to the results from solving the problem formulation in Section IV-A using the CPLEX LP solver. As shown in Fig. 12, although the MPA algorithm has fewer number of satisfied requests than the CPLEX solver. But the differences are less than 11% under varied number of requests.

In Fig. 13, we further show the MPA algorithm has polynomial computation time regarding to the number of requests and nodes in the system. In Fig. 13(a), we fixed the number of requests to 15, and varied the number of nodes from 10 to 100. In Fig. 13(b), we fixed the number of nodes to 50, and varied the number of requests from 10 to 20.Since MPA is a heuristic algorithm, in both evaluation cases, the computation time grows sub-linear to the number of nodes and requests in the system. Therefore, we can conclude from our evaluations that MPA is an fast efficient algorithm that can achieve near optimal solution, and can be applied as an online algorithm in our dynamic module deployment framework.

VI. CONCLUSION

In this paper, we implemented a novel M2M platform which can dynamically deploy modules on a M2M platform. We further designed a deployment controller which is at top of the M2M platform to create the deployment task with the specific node, and also designed a deployment component on nodes to perform the deployment task. Furthermore, we introduced a resource reuse method using the OSGi specification to provide IoT service to devices as soon as possible. As

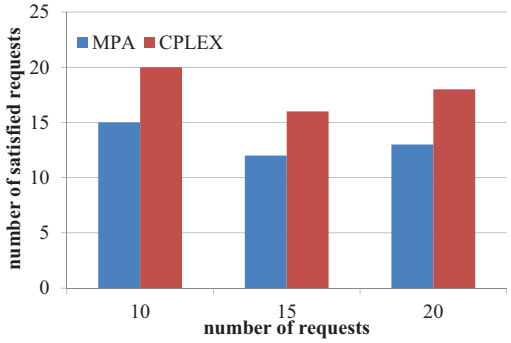
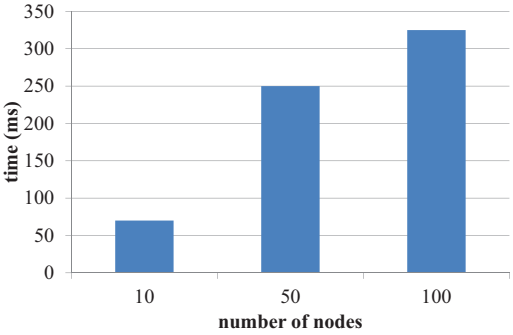
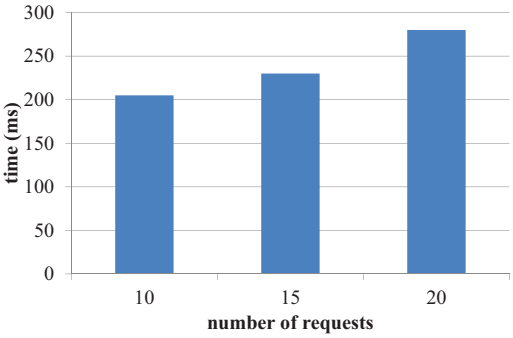


Fig. 12. The MPA algorithm has similar results to the optimal (i.e., CPLEX) algorithm in terms of the number of satisfied requests



(a) The MPA algorithm computation time as the number of nodes in the system increases



(b) The algorithm computation time as the number of requests in the system increases

Fig. 13. The MPA algorithm has polynomial computation time complexity

shown by our evaluation on the real testbed, the proposed method can reduce module package size by 14.3%, the module startup time by almost 75%, and the total deployment time by more than 50%. Finally, we studied the module placement decision by formulating an optimization problem to maximize number of satisfied requests, and proposing an efficient Module Placement Algorithm (MPA). Our evaluations show that the MPA algorithm has polynomial computation time regarding to the number of requests and nodes in the system, and achieves near optimal results comparing the CPLEX LP solver.

This paper is a starting point to use M2M platform for dynamically deploying modules on heterogeneous devices. However, with the incredible growing speed of the number of deployment modules in M2M platform, there are still many open challenges of the proposed platform. For instance, due to the limitation of OSGi specification, when a large number of modules are deployed in the same node in M2M framework, the quality of service provided by the module will be affected.

ACKNOWLEDGEMENT

This study was conducted under the Advanced Communication Technology Research and Laboratory Development project of the Institute for Information Industry, which is subsidized by the Ministry of Economic Affairs of the Republic of China.

REFERENCES

[1] Gartner. Gartner press release. http://www.gartner.com/newsroom/id/3165317, 2015.

[2] Vlad Stirbu. Towards a restful plug and play experience in the web of things. In *Semantic computing, 2008 IEEE international conference on*, pages 512–517. IEEE, 2008.

[3] Tarik Taleb and Andreas Kunz. Machine type communications in 3gpp networks: potential, challenges, and solutions. *IEEE Communications Magazine*, 50(3), 2012.

[4] Dominique Guinard, Vlad Trifa, and Erik Wilde. A resource oriented architecture for the web of things. In *Internet of Things (IOT), 2010*, pages 1–8. IEEE, 2010.

[5] Michael Vogler, Johannes M Schleicher, Christian Inzinger, and Schahram Dustdar. Diane-dynamic iot application deployment. In *Mobile Services (MS), 2015 IEEE International Conference on*, pages 298–305. IEEE, 2015.

[6] OSGi Specifications. https://www.osgi.org/developer/specification.

[7] M Ben Alaya, Yassine Banouar, Thierry Monteil, Christophe Chassot, and Khalil Drira. Om2m: Extensible etsi-compliant m2m service platform with self-configuration capability. *Procedia Computer Science*, 32:1079–1086, 2014.

[8] oneM2M TS-0001. ”functional architecture”. v2.10.0,August,2016.

[9] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Caceres, and Nigel Davies. The case for vm-based cloudlets in mobile computing. *IEEE pervasive Computing*, 8(4), 2009.

[10] Dale Willis, Arkodeb Dasgupta, and Suman Banerjee. Paradrop: a multi-tenant platform to dynamically install third party services on wireless gateways. In *Proceedings of the 9th ACM workshop on Mobility in the evolving internet architecture*, pages 43–48. ACM, 2014.

[11] Hua-Jun Hong, Pei-Hsuan Tsai, and Cheng-Hsin Hsu. Dynamic module deployment in a fog computing platform. In *Network Operations and Management Symposium (APNOMS), 2016 18th Asia-Pacific*, pages 1–6. IEEE, 2016.

[12] OM2M. Open source platform for m2m communication. http://www.eclipse.org/om2m/.

[13] Chao-Lin Wu, Chun-Feng Liao, and Li-Chen Fu. Service-oriented smart-home architecture based on osgi and mobile-agent technology. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 37(2):193–205, 2007.

[14] Kura. Open source java/osgi-based framework for iot gateway. http://www.eclipse.org/kura/.

[15] IBM CPLEX optimizer. http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/.

[16] GNU Project. Glpk (gnu linear programming kit). https://www.gnu.org/software/glpk/.

[17] Cbc (coin-or branch and cut). https://projects.coin-or.org/Cbc.