Ann. Telecommun. (2017) 72:105–115
DOI 10.1007/s12243-016-0524-9

Service placement for latency reduction in the internet of things

Karima Velasquez[1] · David Perez Abreu[1] · Marilia Curado[1] · Edmundo Monteiro[1]

Received: 30 October 2015 / Accepted: 26 May 2016 / Published online: 9 June 2016

Abstract New services and applications become part of our daily activities as we evolve into new solutions supported by cutting-edge paradigms, like the Internet of Things and Smart Cities. In order to properly achieve the benefits theoretically provided by these models, new kinds of services must be designed. These new services have special requirements, as well as the users that access to them. One of these requirements is low latency levels, since a delayed reply could render to chaos for applications such as eHealth and public safety. The communication infrastructure must cope with these challenges by offering innovative solutions. One of these solutions is a smart service placement system that facilitates the location of services in the proper position according to specific needs. On this paper, a service placement architecture for the Internet of Things is proposed, with especial emphasis in its main module, the *Service Orchestrator*, for which implementation details are provided, including a model for the service placement task. Furthermore, technologies to implement the modules from the architecture are suggested. This proposal, as well as its validation, is framed within the scope of the SusCity project.

Keywords Latency · Fog · IoT · Service placement

✉ Karima Velasquez
kcastro@dei.uc.pt

[1] Department of Informatics Engineering, University of Coimbra, Polo II, Pinhal de Marrocos, 3030-290, Coimbra, Portugal

1 Introduction

As citizens rely more each day on the Informationand Communication Technologies (ICT) for their daily activities, cities move toward a new paradigm known as Smart Cities. A Smart City [1] is a place where conventional networks and services are transformed to make them more efficient, by using the ICT and the Internet of Things (IoT) to obtain benefits not only for the citizens but also for economic reasons. The ultimate goal of this paradigm is to provide the environment for better use of resources. Some of the services commonly comprised within a smart city include smarter urban transport networks, water supply and disposal facilities, and more efficient ways to heat and light buildings; additionally, also include the involvement of the city administration to define the guidelines needed to achieve this self-sustainability.

Many organizations are working to make this idea a reality, with several cities already considered as "Smart Cities" [2]. The SusCity project [3] is one of these initiatives. With the cooperation of several universities (including University of Coimbra, University of Minho, Lisbon Instituto Superior Técnico, and Massachusetts Institute of Technology), leading industries (including IBM and EDP - Energias de Portugal) and government offices (including CML - Câmara Municipal del Lisboa and CCDR - Comissão de Coordenação e Desenvolvimento Regional), the SusCity project aims at the identification of possible solutions to achieve self-sustainable environments in the city of Lisbon, Portugal. The idea behind the project is to collect information from different sources and feed it in a dashboard that, using estimation models, can predict the consequences of

applying some measures in particular locations. The results can in turn be analyzed by government members and even by citizens to make a decision about the most appropriate solution according to their needs, with solutions focused on smart mobility, buildings, and energy grids.

However, in order to successfully deliver the data from the information collectors to the dashboard, the ICT must provide efficient support for the different types of data and requirements from the diverse set of services available. The communication infrastructure must support data, for instance online and offline, aggregated and raw. Additionally, the services also have particular requirements to guarantee the proper working of the system. One of these requirements lies on the need of providing quick responses, for what is necessary to come up with smart mechanisms to mitigate the latency for the services, particularly for time-sensitive applications (e.g., eHealth, augmented reality) which are common on this type of environments.

One possibility is to take advantage of the proposed scenario by bringing the service closer to the IoT environment, to the edge of the network (also known as the *Fog*). The Fog is an extension of the Cloud Computing paradigm, where all the services are migrated to the edge of the network, creating a highly virtualized platform that offers storage, compute, and networking services between end users and Cloud data centers [4] [5], hence locating the services just one wireless hop away from the users. This solution is not as simple as it sounds, since the service could be wrongfully placed in a congested location, or even farther from the users (just to mention some possibilities), which would lead to a greater latency. Thus, efficient service placement mechanisms are needed in order to take advantage of the current conditions of the network while minimizing the service latency.

To reach this goal, a modular architecture was designed to perform intelligent service placement tasks in the ICT for the SusCity project, with the objective of reducing the service latency. The architecture includes an information collection module that learns from the current condition of the network to influence the placement decision.

This paper presents the architecture designed, with a special emphasis on its main module, the *Service Orchestrator*, including some details for its implementation, as well as a description of its interactions with the rest of the modules and indications on the technologies and protocols to use. Furthermore, possible applications for the architecture, not only within the context of the SusCity project but also for more general services, are suggested. The main contributions of this work are the following:

- The architecture. An architecture for the placement of IoT services that incorporates both the cloud and its edges, while adapting the solution to changing conditions in the system, aimed at the reduction of the latency.
- The orchestrator algorithm. An algorithm for the Service Orchestrator module of the architecture, that explains the main operation of the architecture, from the gathering of information regarding the network and system status, to the service placement and migration.
- The service placement model. A model using Integer Linear Programming to achieve the optimization goals of the architecture.

The paper is organized as follows. Section 2 presents a revision of related work and Section 3 introduces the proposed architecture and the description of its modules and interactions, providing some implementation details. Section 4 explains the applications that are defined in the scope of the SusCity project and introduces application scenarios for the architecture, while also emphasizing on the contributions of this proposal. Final remarks and conclusions are presented in Section 5.

2 Related work

Some works have already dealt with placement issues. Many works are focused on the placement of virtual machines (VM) [6–8] or even for virtual data centers (VDC) [9] and for intra-data center traffic [10]. While some ideas can be extrapolated, the concept of services differs from the placement of the whole VM, since it represents a finer granularity level of the same problem.

Furthermore, the solution to identify the best place to locate either VMs or services is usually addressed as a multi-objective optimization problem, since there are several factors to take into consideration. For instance, by trying to decrease costs, there could be consequences such as increasing the energy consumption. In these cases, the optimization process tends to be prioritized by objectives, according to the needs of the customers or the service provider.

For example, Zhang et al. [11] aim at optimizing hosting costs, while Huang et al. [12] focus at optimizing energy costs. Ooi et al. [13] target the optimization of the availability of the services when resources fail, and Ghaznavi et. al. [14] look for minimizing operational costs in providing virtual network function (VNF) as a service. Unlike these works, the main focus of this paper is on the reduction of the service latency, while other optimization goals are also considered with lower priority levels.

Another important issue that is not considered in some works [15–17] is the use of the Fog paradigm. While it might seem that the problem is basically the same (placement in the cloud or in the fog), including the Fog brings

additional benefits for final users, for instance lower latency levels; moreover, since the Fog represents a single-hop destiny for final users, it maximizes the availability of the services on disaster scenarios (e.g., earthquake, flooding).

The placement of cloudlets, small datacenters on the edge of the network, has also been studied by Jia et al. [18]; however, their work is focused on the assignment of users to the closest cloudlet, and did not address the problem of migration once the network conditions and users' location changed. The dynamism of an online solution is also not considered by Bienkowski et al. [19].

For IoT environments, it is important to collect the data from sensors, and send it promptly to the cloud services in order to be properly processed. The collection process must be completed in short amounts of time. Unlike the works previously described, this proposal is focused on the placement of services combining the advantages of both cloud and Fog environments, providing support for dynamic conditions by offering a network-aware inline solution with the objective of reducing the latency of the services provided.

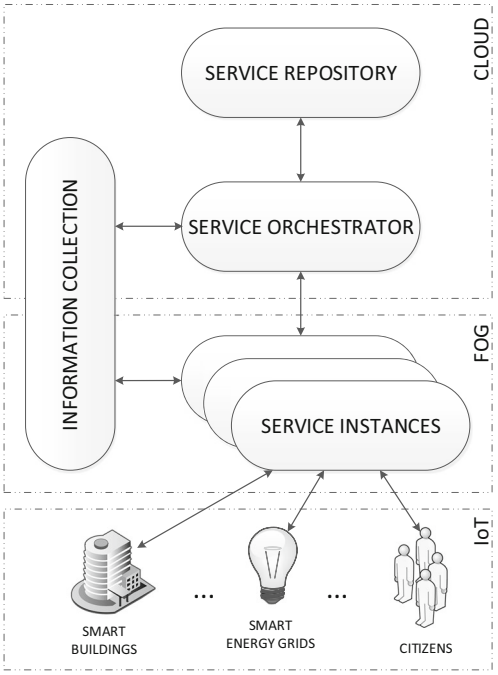

Fig. 1 Proposed Architecture

3 Proposed architecture

A modular architecture was designed in order to deal with the issues exposed in Section 1. The architecture will enable the use of smart service placement schemes that guarantee the reduction of the latency according to the current state of the network and the location of users and servers, while taking advantage of the fog scenario. The strategy behind the architecture is gathering information from the underlying network and using this information to place the services in the most convenient locations.

The architecture, depicted on Fig. 1, is composed by three main modules, plus the locations where the service instances are going to be placed. The following subsections describe each module and the interactions among them.

3.1 Service repository

The first module is the *Service Repository*. This is the place where the available services are stored. A service is used to build applications that provide a set of functionalities to users in the IoT environment. The services can be used by themselves, or can be combined to create more complex composite services and/or applications. Also, the Service Repository can be updated with new services that arose according to users' needs.

By physically locating this module in the Cloud, it will provide the architecture with additional freedom to move the instances to different locations according to the needs of the users. Besides the actual services, this module also stores metadata about the services (e.g., requirements for a specific content) that will influence the location of each service.

3.2 Information collection

The following module is the *Information Collection* module. This module will be in charge of gathering information from the network and about the interaction among the users and the services requested. This information, together with the metadata from the services, is going to influence the future locations of the service instances. To implement this module, an option is using the Application-Layer Traffic Optimization (ALTO) protocol [20, 21]. The ALTO protocol offers a mechanism to perform better-than-random peer selection in peer-to-peer (P2P) networks, by obtaining information about the underlying network that helps to create an optimal overlay network, for instance, grouping devices that are located closer to each other, thus reducing the response time in their communication. Among the pieces of information that can be obtained with the ALTO protocol are routing policies from autonomous systems (ASs), bandwidth capacity of the nodes, and hop count. Although it was originally intended for P2P networks, this protocol can be applied to any kind of network.

The ALTO protocol uses two main information elements, called *network map* and *cost map* [22]. The network map contains the description of groups of hosts without including information about the connection between those groups. Thus, the network map describes the topology of the

network. The cost map is associated to a network map and defines unidirectional connections between groups of hosts with an associated cost value for each one-way connection. The cost map also includes the definition of the metric type (e.g., hop count, routing cost) and the unit type (e.g., numerical, ordinal).

Another option to get information about the network is using the Simple Network Management Protocol (SNMP), that enables the exchange of management information between network devices [23]. The SNMP model comprises four key elements: (1) the management station, (2) the agent, (3) the Management Information Base (MIB), and (4) the network management protocol. The management station is the core of the system, in charge of network configuration and troubleshooting. The agents could be hosts, routers, and bridges, and are responsible for monitoring the network conditions, collecting information that is sent to the management station. With this information, the topology of the network and its condition can be obtained. The MIB serves as a collection of objects that represents the knowledge shared, and the management protocol defines the exchange of information.

However, the mechanism employed by SNMP has been reported to have limitations [23]. Thus, ALTO represents a better option for the *Information Collection* module.

Moreover, it is important to consider the design of a standard format for the description of the network and cost maps, possibly using ontologies [24] to guarantee a unified language that can be correctly interpreted by all the modules involved.

3.3 Service orchestrator

The next module is called *Service Orchestrator*, which is the core of the service placement architecture. This module implements the strategies that will make the decisions about the current location for every service instance. By using the data provided by the Information Collection module and the metadata from the services, it will be able to request the corresponding service instance from the Service Repository and ultimately locate it in the appropriated location. This model gives the freedom to even combine some service instances in order to create more complex services. More technical details about this module are described in the following subsections.

3.3.1 Algorithms for the service orchestrator

There are two possibilities to start the tasks of the Service Orchestrator, and they differ in how the first placement is done. The first one is doing the first placement randomly, and the second one is using a prediction about the state of the network. In both cases, subsequent placements are done by taking into account the information about the status of the network.

Algorithm 1 shows the functionality for the Service Orchestrator using the first approach. In this case, once the Service Orchestrator starts, it places the services in random locations, to later adapt to the conditions of the network, which will be informed by the Information Collection module.

The algorithm receives as input the locations of the servers (content delivery networks—CDN servers, see Section 3.4), where the service instances are going to be deployed. Since this algorithm corresponds with a random start, the Service Orchestrator requests the instances of the services (line 3) and begins to place them randomly in the servers (line 4). It is also noticeable that the orchestrator calculates the number of instances of the service to deploy according to the data provided by the Information Collection module. For instance, the amount of user's requests by location. Then, on line 6, a timer is used to allow some interactions that will feed the Information Collection module with data to create new maps. The definition of this timer is an important issue to take in consideration, since a timer too small can generate unwanted oscillations in the system, but a timer too large could derive in the system not adapting to changes fast enough. The final tuning of the value of this timer will depend on experimental results. Afterward, the Service Orchestrator requests the network and cost maps from the Information Collection module (line 7) and with this data performs an informed placement of services (*instances*) in the CDN servers (*locations*) for subsequent iterations of the algorithm. The strategy refers to the particular scheme used for the service placement task (e.g., greedy heuristics). This allows the Service Orchestrator to apply different strategies according to the needs of the system, defined by the administrator.

Algorithm 1 reflects the behavior of the Service Orchestrator using the second approach. In this case, the Information Collection module is capable of creating a map without current information of the underlying network concerning the services that are going to be deployed, but using estimations from previous knowledge (e.g., traffic conditions) and from some data loaded by the system administrator, hence creating a "prediction" of the behavior of the network and of future interactions of the users. Loaded with these maps and with metadata from the Service Repository, the Service Orchestrator performs the placement of the services.

Once again, the algorithm receives the locations as input. On line 4 the Service Orchestrator requests the network and cost maps from the Information Collection module, that on the first iteration will consist on a prediction based on the conditions of the network and some input from the system administrator (e.g., priority of the services). This information, combined with the metadata from the services, will

Algorithm 1 Service orchestrator - random start

```
Input : locations
1:  Start
2:      procedure RANDOMSTART(locations)
3:          instances ← requestInstance(service)
4:          servicePlacement(random, instances, locations, metrics)
5:          while (TRUE) do
6:              wait(time)
7:              maps ← requestMaps()
8:              instances ← requestInstance(service)
9:              servicePlacement(strategy, instances, locations, maps)
10:         end while
11:     end procedure
12: End
```

allow the Service Orchestrator to determine the amount of service instances to deploy as well as to perform a "better-than-random" placement of the services (line 6). Then, a timer is used to allow the interactions of the users with the Service Instances, which will generate new metrics for the Information Collection module, that in turn will create updated network and cost maps for the Service Orchestrator.

3.3.2 Service placement model

The core task of the Service Orchestrator corresponds to the service placement function, which represents an NP-hard problem. Different alternatives have been evaluated. Some possibilities include using an evolutionary approach [25], a solution based on greedy heuristics [26], or applying the solutions to well-known problems (e.g., facility location problem [27]).

Another possibility is using Integer Linear Programming (ILP) [12, 28–30], since it represents a well-known mechanism to deal with NP-hard optimization problems, in which the variables are restricted to integer values [31]. For the Service Orchestrator, the variables to use can be modeled as integer, for instance, the set of nodes, edges, applications and services that compose them, and requests. A model using ILP for the *servicePlacement* task (see Algorithm 1 line 9) has been designed. The model, which borrows ideas from Moens et al. [28], incorporates the awareness of the service requests, thus bringing more dynamism; furthermore, the optimization objectives were rebuilt and enhanced to tailor them to the problem identified in this research. The variables used in the model are summarized in Table 1. The optimization objectives were defined as follows:

- Minimize the hop count between users' location and serving nodes (Service Instances). This will allow to place the service instances closer to users, hence reducing the latency.
- Minimize the hop count between communication nodes. In case there are services communicating among each others, they should be placed close together in order to decrease even more the latency.
- Minimize the number of service migrations. In favor of keeping the system stable and reduce the oscillation of services.

Algorithm 2 Service orchestrator - prediction start

```
Input : locations
1:  Start
2:      procedure PREDICTIONSTART(locations)
3:          while (TRUE) do
4:              maps ← requestMaps()
5:              instances ← requestInstance(service)
6:              servicePlacement(strategy, instances, locations, maps)
7:              wait(time)
8:          end while
9:      end procedure
10: End
```

Table 1 Notation

Symbol	Description
N	Set of nodes on which the service can be executed
E	Set of edges that connect the nodes
A	Set of applications (each corresponds to a set of services)
R	Set of services
R_a	Set of requests for an application
L	Set of locations where requests are generated
$I_{a,s}$	Instance matrix. Equals 1 if service s is part of application a, and 0 otherwise
$H_{n1,n2}$	Hops matrix. Indicates the amount of hops between $n1$ and $n2$
$P_{s,n}^{a,r}$	Placement matrix. Equals 1 if service s is executed on node n for the r^{th} request of application a. 0 otherwise
$F_{s1,s2}^{a,r}(n_1, n_2)$	Flow matrix. Contains the amount of bandwidth (Mbps) belonging to the r^{th} request for application a, used for communication between services
$Q_{a,l}$	Request matrix. Contains the amount of requests belonging to a originated in location l
$U_{s,n}$	Execution matrix. Equals 1 when an instance of service s is executed on node n
U^{t-1}	Execution matrix from the previous invocation of the placement algorithm

Equation 1 corresponds with the objective of minimizing the hop count between users and serving nodes. The idea is, with the help of the Information Collection module, to keep track of the amount of requests for each service generated on each location. That information (gathered on the request matrix Q) is used to minimize the hop count matrix.

$$min \sum_{n,l \in N} H_{n,l} \times \sum_{a \in A} \sum_{l \in L} Q_{a,l} \tag{1}$$

Equation 2 describes the objective of minimizing the hop count between communicating nodes, thus reducing the communication latency and eventually the service latency. The flow matrix accounts the bandwidth consumption between communicating services. By bringing closer together the service instances with the most bandwidth consumption among them, this objective is accomplished.

$$min \sum_{n1,n2 \in N} H_{n1,n2} \times \sum_{a \in A} \sum_{r \in R} \sum_{s1,s2 \in S} F_{s1,s2}^{a,r}(n_1, n_2) \tag{2}$$

Finally, Equation 3 implements the last objective of minimizing the amount of service migrations. Given that the algorithm is executed iteratively, the execution matrix from the previous iteration can be used to compare with the current one, thus reducing the number of service migrations in the system.

$$min \sum_{s \in S} \sum_{n \in N} |U_{s,n} - U_{s,n}^{i-1}| \tag{3}$$

The model takes in consideration applications that are composed of more than one service, which corresponds with realistic IoT environments. Moreover, the iterations of the algorithm allows to migrate services when the conditions of the system change.

3.4 Service instances

For the *Service Instances*, the architecture combines paradigms such that the location of the service is optimized according to its service time. One of these paradigms is Fog computing. Since the fog is located at the edge of the network, some benefits can be provided, such as location awareness, low latency, support for mobility, and predominance of wireless access [32]. The fog paradigm is ideal to work with other technologies, like content delivery networks (CDN). A CDN is a group of servers, with replicas of the content and the services from original servers, that allows to offload the work from those original servers [33]. Once a request is placed, the CDN locates a server closer to the user (geographically, topologically, or by any parameter). By placing the non-original servers at the edge of the network (in the fog), the latency can be reduced. For the proposed architecture, the locations where the Service Instances are going to be placed, refer to CDN servers positioned in the fog.

The Service Orchestrator and the Service Repository, along with some components (for instance, the server) of the Information Collection module could be deployed in the Cloud, together with the centralized component of the monitoring system in charge of collecting all the information and creating the predictions to be used. On the contrary, the Service Instances and the rest of the components of the Information Collection (e.g., the clients) should be located at the Fog, closer to the users of the services, to take full

advantages of the mobility support, location awareness, and low latency, offered by the Fog.

The Service Instances are updated and multiplied according to the data provided by the Information Collection module, such as users demands and locations, in order to prevent bottle necks in high demand periods, or waste of resources on low demand periods.

3.5 Interactions among the modules

The interactions between the modules are described using sequence diagrams. As stated in Subsection 3.3.1, two approaches have been used for the start-up process: (1) randomly placing the services for the first time, and (2) using estimations of the network behavior to locate the service instance on the first attempt.

Figure 2 shows the first approach. After some interactions with the users, the metrics are gathered by the Information Collection module, and the network and cost maps are built (depicted by the activity block in this module). Then, the maps are requested by the Service Orchestrator, that uses them to calculate the next best location for the service instances.

Figure 3 shows the second approach. In this case, the Information Collection module calculates the network and cost maps based on the prediction about the network behavior. These maps are then delivered to the Service Orchestrator upon request, which in turn uses them to calculate the optimal locations for the service instances. After the services are placed, the interactions with the users begin, thus generating new metrics that will feed the Information Collection module to generate new network and cost maps updated with the current information from the network.

In both cases, the network and cost maps will continuously get updated by the collection of new metrics, as was depicted by Algorithms 1 and 2. This will influence future service placement decisions. Additionally, the service instances will have some metadata that will help in the decision making process of the Service Orchestrator. For the interaction of the modules, it is also important to take in consideration the use of application programming interfaces (API), such as the one defined by Sköldström et al. [34].

4 Applications and discussion

To further clarify how the proposed architecture is going to support the placement of the required services described, some examples are illustrated; in addition, some discussion about the novelty of the contribution is also provided.

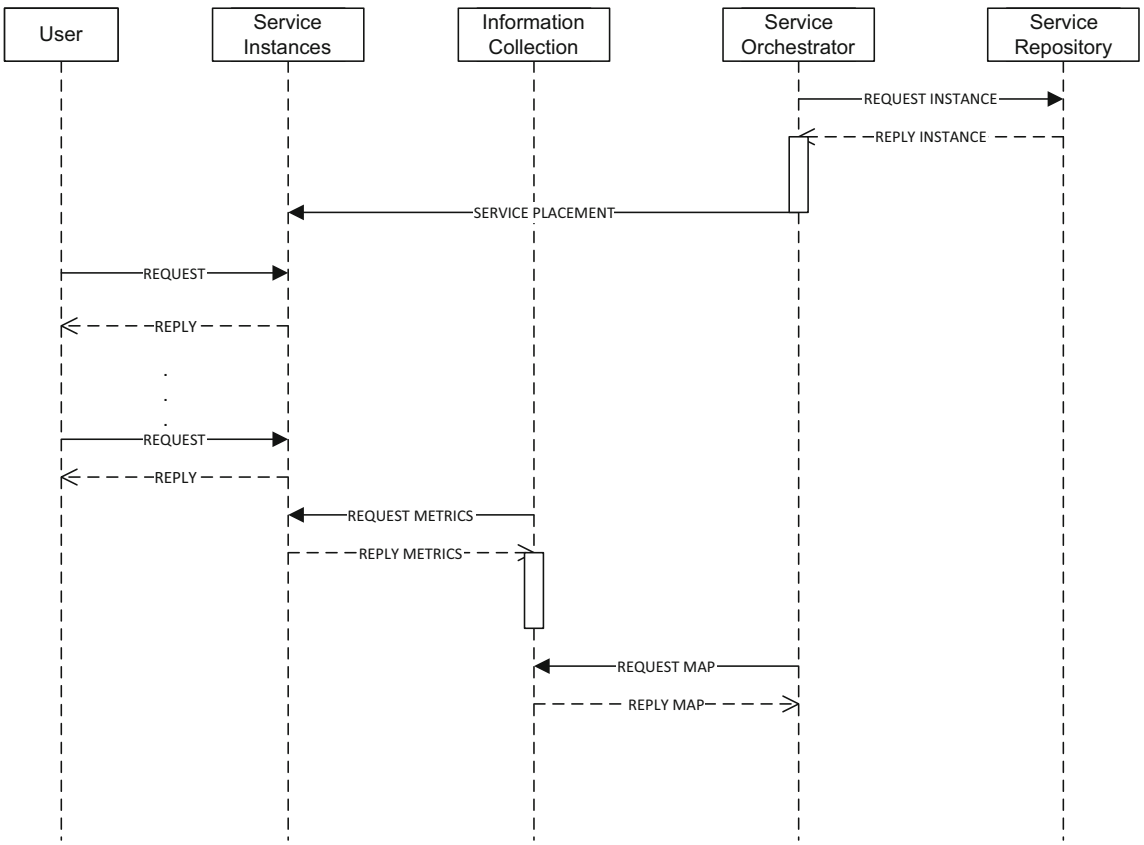

Fig. 2 Sequence diagram—random start

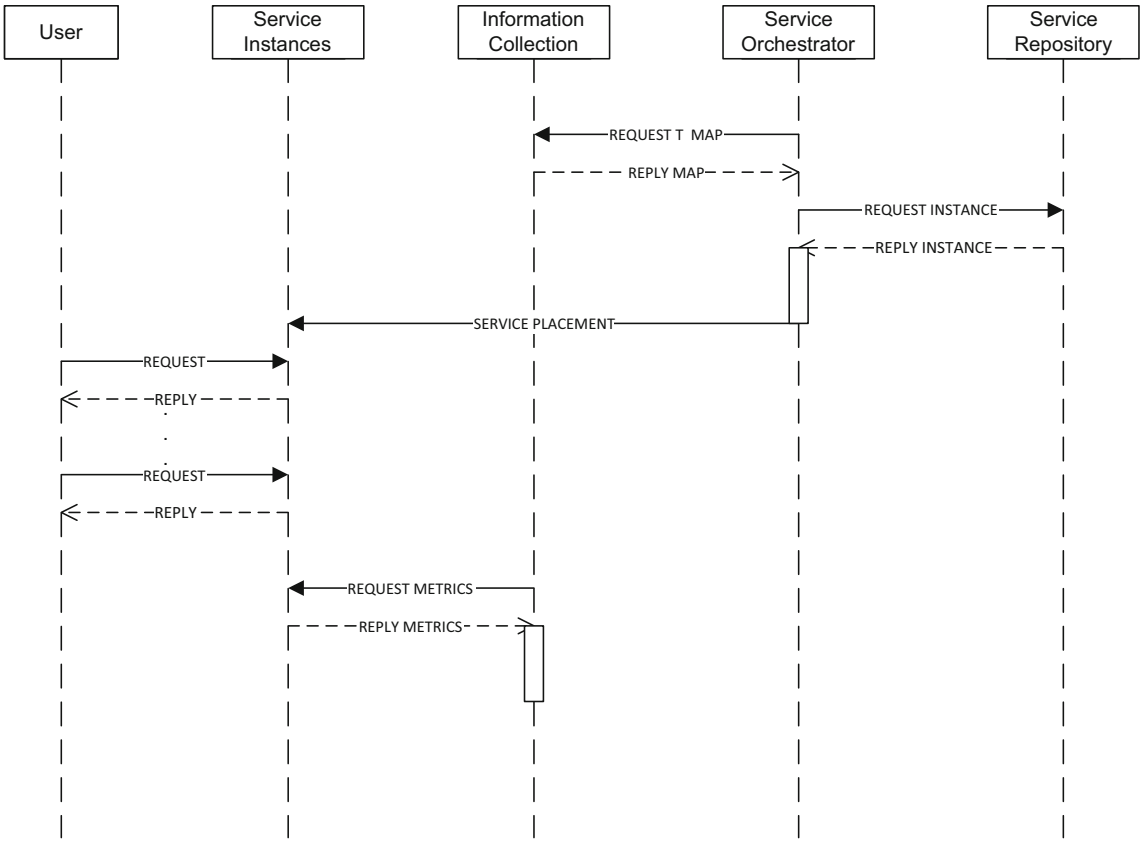

Fig. 3 Sequence diagram—prediction start

4.1 Application scenarios

The SusCity project has defined three main types of applications: (1) smart buildings, (2) smart energy grids, and (3) smart mobility solutions. In the case of smart buildings, the services will include processes to monitor, predict, and control performance on buildings. Among the indicators that can be used to monitor the buildings are energy consumption, occupant behavior, and indoor environment quality. For smart grids, it has been considered to include solar power forecasting and electric vehicles charging optimization. Finally, for smart mobility, proposed solutions include on-road monitoring of private and public vehicles, and also collecting information from pedestrian's smartphones, to generate urban mobility models. To get the measurements, different sensors are going to be deployed.

One of the services that are going to be needed is the data collection service. In the case of the smart mobility solutions, special sensors (e.g., global positioning system, GPS) are going to be placed on vehicles, as well as using smartphones for pedestrians, to monitor for instance the origin and destination of individual trips. According to their movement, they could get farther away from the collecting service, thus delaying this process. In this case, a migration of the service could be useful to place the service in locations closer to the data sensing devices. Service instances would be running on CDN servers that will generate metrics, like popularity of the services by measuring the amount of requests. These metrics then will be gathered by the Information Collection module, which will generate the corresponding updated network and cost maps and provide them to the Service Orchestrator upon demand. With these maps, the Service Orchestrator can make the decision of migrating the service to a more convenient location.

Figure 4 describes this example. Data gathered by smartphones is collected by Service Instance "C", while data gathered by GPS sensors is collected by Service Instance "A". Since the GPS sensors are located in moving vehicles, it is possible that these sensors move away from "A" toward another service instance. In the example depicted in Fig. 4, the cars move closer to Service Instance "B", thus, the Information Collection module would register the update on the location of the requests for this service, and forward this information to the Service Orchestrator which would then migrate the service from Service Instance "A" to "B".

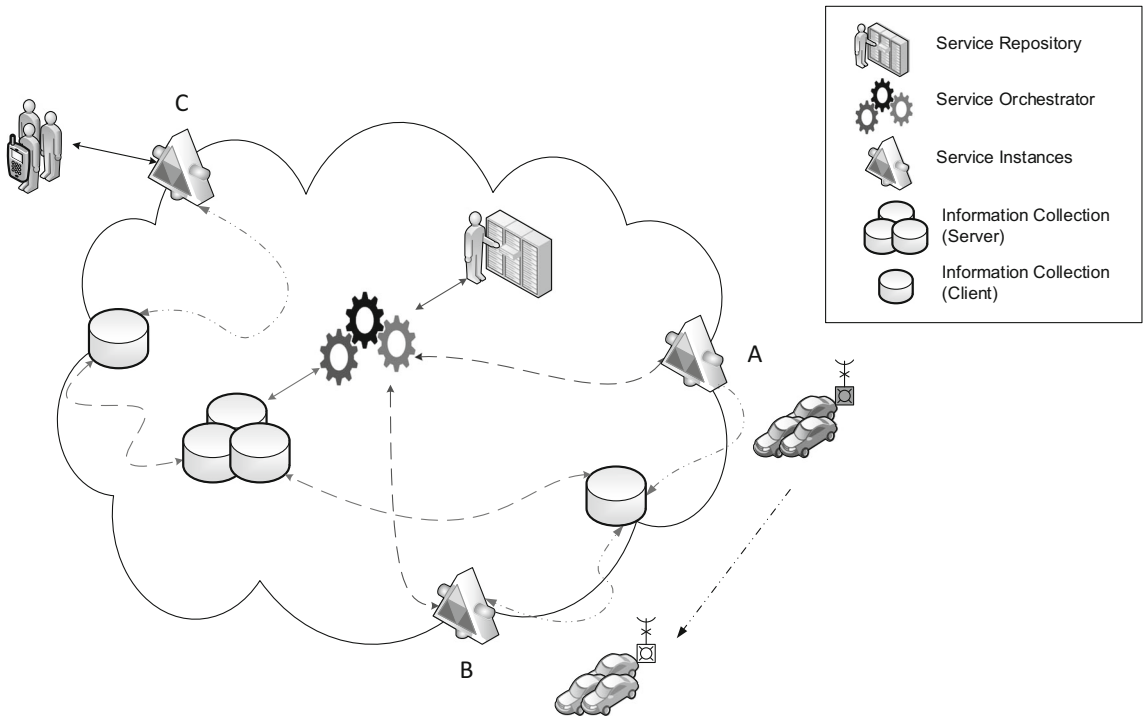

Fig. 4 A data collection service scenario

On a more advanced stage, it is possible to consider other automatized solutions. For instance, imagine a city where a festival is taking place, in which a big amount of users are concentrated in specific geographic areas. A possibility for a service could be offering customized directions to points of interest to moving users (e.g., restaurants, concerts, and special offers). Thus, by monitoring the places where the requests for this service are focused, the Service Orchestrator could migrate the service to a more appropriate location and hence could provide a faster response.

It is important to mention that although this architecture was designed for the SusCity project, it is generic enough to fit in different scenarios requiring to perform service placement tasks. For instance, consider a football match. An interesting service is providing instant replays and relevant information for the match (e.g., red and yellow cards, player changes) directly to spectator's smartphones. The service would be available for the duration of the match, and according to the number of people (simultaneous active connections) in the stadium, several instances could be deployed to guarantee the quality of the service. The decision of the amount of instances and their locations would be made by the service placement system automatically. The first instance could be made available in a fixed manner, while the additional instances will be set up by the Service Orchestrator according to the data gathered by the Information Collection module. The service could even be extended to specific areas (e.g., squares, malls) where crowds are formed spontaneously.

4.2 Discussion

The architecture proposed is aimed at the achievement of a smart service location with emphasis on reducing the response time for the services. Unlike existing solutions discussed on Section 2, this architecture takes in consideration two important factors:

1. Making an informed decision by learning the current condition of the underlying network in a standard manner.
2. Taking advantage of the fog paradigm.

The architecture has a key module, the *Information Collection* that allows to know the changes on the network status, as well as the interactions between the users and the system. This vital information allows the system to be aware and adapt to the changing conditions of its surroundings, hence providing support for more realistic environments. Furthermore, the incorporation of novel technologies such as the fog computing paradigm and the ALTO protocol infuse the solution with up-to-date mechanisms that improve the final results.

Additionally, it is important to mention that multi-objective optimization principles are being taken into

consideration, since not only the latency must be minimized but also other types of resources (e.g., storage, bandwidth, content required, energy consumption) are being contemplated in the solution.

5 Conclusions

The transition to the Smart City paradigm leads to the rise of new applications and services. The communication infrastructure must deal with the challenges this new scenario imposes. One of these challenges is the latency, since the communication infrastructure becomes more crowded leading to an increased delay; furthermore, some applications (e.g., eHealth, public safety control) have more strict time constraints that have to be met. One possible solution for this issue is the use of smart service placement mechanisms that bring the Cloud located services toward the users at the edge of the network.

In this paper, an architecture for service placement was presented. The goal of the architecture is to locate services in convenient servers at the fog level and continuously migrate these services according to the changing conditions of the network and status of the users, by learning information from the communication environment. The modules for the architecture were characterized as well as the interactions among them. Also, some ideas for the behavior of the Service Orchestrator were discussed as well as some details of its implementation using ILP.

As future work, we plan to continue this path by refining the set of variables and optimization objectives used on the service placement task, and working on the validation of the proposed models within the SusCity project framework.

Acknowledgments This work is financed by national funds via FCT - Foundation for Science and Technology within the scope of project "MITP TB / CS / 0026/2013 - SusCity".

References

1. Smart Cities. Digital Agenda for Europe - A Europe 2020 Initiative, 2016, Last visited: 10-05-2016. [Online]. Available: https://ec.europa.eu/digital-agenda/en/smart-cities
2. European Smart Cities, 2016, Last visited: 10-05-2016. [Online]. Available: http://www.smart-cities.eu
3. FCT - SusCity Project, 2016, Last visited: 10-05-2016. [Online]. Available: http://groups.ist.utl.pt/suscity-project/home
4. Vaquero LM, Rodero-Merino L (2014) Finding your way in the fog: towards a comprehensive definition of fog computing. SIGCOMM Comput Commun Rev 44(5):27–32
5. Technology Radar, Cisco, December 2014, Whitepaper sponsored by Cisco
6. Piao JT, Yan J (2010) A network-aware virtual machine placement and migration approach in cloud computing. In: Proceedings of the 9th International Conference on Grid and Cloud Computing, GCC 2010, pp 87–92
7. Camati R, Calsavara A, Lima Jr L (2014) Solving the virtual machine placement problem as a multiple multidimensional knapsack problem. In: Proceedings of the 13th International Conference on Networks, ICN 2014, pp 253–260
8. Ren Y, Suzuki J, Vasilakos A, Omura S, Oba K (2014) Cielo: An evolutionary game theoretic framework for virtual machine placement in clouds. In: Proceedings of the 2014 International Conference on Future Internet of Things and Cloud, ser. FICLOUD'14, pp 1–8
9. Amokrane A, Zhani MF, Langar R, Boutaba R, Pujolle G (2013) Greenhead: virtual data center embedding across distributed infrastructures. IEEE Trans Cloud Comput 1(1):36–49
10. Pu Q, Ananthanarayanan G, Bodik P, Kandula S, Akella A, Bahl P, Stoica I (2015) Low latency geo-distributed data analytics. SIGCOMM Comput Commun Rev 45(4):421–434
11. Zhang Q, Zhu Q, Zhani M, Boutaba R, Hellerstein J (2013) Dynamic service placement in geographically distributed clouds. IEEE J Select Areas Commun 31(12):762–772
12. Huang Z, Lin K-J, Yu S-Y, Hsu J. Y-J (2014) Co-locating services in IoT systems to minimize the communication energy cost. J Innov Digit Ecosyst 1(1–2):47–57
13. Ooi BY, Chan HY, Cheah Y. N (2010) Dynamic service placement and redundancy to ensure service availability during resource failures, vol 2, pp 715–720
14. Ghaznavi M, Khan A, Shahriar N, Alsubhi K, Ahmed R, Boutaba R (2015) Elastic virtual network function placement. In: IEEE 4th International Conference on Cloud Networking (CloudNet), pp 255–260
15. Steiner M, Gaglianello BG, Gurbani V, Hilt V, Roome W, Scharf M, Voith T (2012) Network-aware service placement in a distributed cloud environment. SIGCOMM Comput Commun Rev 42(4):73–74
16. Hao W, Thuraisingham B, Yen I (2009) Dynamic service and data migration in the clouds. In: Proceedings of the International Computer Software and Applications Conference, pp 134–139
17. Spinnewyn B, Braem B, Latré S (2015) Fault-tolerant application placement in heterogeneous cloud environments. In: 11th International Conference on Network and Service Management (CNSM), pp 192–200
18. Jia M, Cao J, Liang W (2015) Optimal cloudlet placement and user to cloudlet allocation in wireless metropolitan area networks. IEEE Trans Cloud Comput 99:1–1
19. Bienkowski M, Feldmann A, Jurca D, Kellerer W, Schaffrath G, Schmid S, Widmer J (2010) Competitive analysis for service migration in vnets. In: Proceedings of the second ACM SIGCOMM workshop on Virtualized infrastructure systems and architectures, pp 17–24
20. Alimi R, Penno R, Yang Y, Kiesel S, Previdi S, Roome W, Shalunov S, Woundy R (2014) Appl-Layer Traffic Optim (ALTO) Protocol 9:RFC7285
21. Seedorf J, Burger E (2009) Appl-Layer Traffic Optim (ALTO) Probl Statement 10:RFC5693
22. Faigl Z, Szabó Z, Schulcz R (2014) Application-layer traffic optimization in software-defined mobile networks: a proof-of-concept implementation. In: Proceedings of the 16th International Telecommunications Network Strategy and Planning Symposium (Networks), pp 1–6
23. Zhou YT, Deng ML, Ji FZ, He XG, Tang QJ (2015) Discovery algorithm for network topology based on SNMP. In: International Conference on Automation, Mechanical Control and Computational Engineering. Atlantis Press

24. Wang W, De S, Toenjes R, Reetz E, Moessner K (2012) A comprehensive ontology for knowledge representation in the internet of things. In: IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)
25. Gao Y, Guan H, Qi Z, Hou Y, Liu L (2013) A Multi-objective ant colony system algorithm for virtual machine placement in cloud computing. J Comput Syst Sci 79(8):1230–1242
26. Biran O, Corradi A, Fanelli M, Foschini L, Nus A, Raz D, Silvera E (2012) A stable network-aware VM placement for cloud systems. In: Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. IEEE Computer Society
27. Boloori Arabani A, Farahani RZ (2012) Facility location dynamics: an overview of classifications and applications. Comput Indust Eng 62(1):408–420
28. Moens H, Hanssens B, Dhoedt B, Turck FD (2014) Hierarchical network-aware placement of service oriented applications in clouds. In: 2014 IEEE Network Operations and Management Symposium (NOMS), pp 1–8
29. Lukovszki T, Rost M, Schmid S (2016) It's a match!: near-optimal and incremental middlebox deployment. SIGCOMM Comput Commun Rev 46(1):30–36
30. Sung J, Kim M, Lim K, Rhee J-K (2013) Efficient cache placement strategy for wireless content delivery networks. In: Proceedings of the International Conference on ICT Convergence, pp 238–239
31. Stephen TLM, Bradley P, Hax ArnoldoC Integer programming. In: Applied Mathematical Programming. Addison-Wesley Publishing Company, 1977, ch. 9, pp. 272–319
32. Bonomi F, Milito R, Zhu J, Addepalli S (2012) Fog computing and its role in the internet of things. In: Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, ser. MCC '12. ACM, pp 13–16
33. Krishnamurthy B, Wills C, Zhang Y (2001) On the use and performance of content distribution networks. In: Proceedings of the First ACM SIGCOMM Workshop on Internet Measurement Workshop - IMW '01, p 169
34. Sköldström P, Sonkoly B, Gulyás A, Németh F, Kind M, Westphal F-J, John W, Garay J, Jacob E, Jocha D, Elek J, Szabó R, Tavernier W, Agapiou G, Manzalini A, Rost M, Sarrar N, Schmid S (2014) Towards unified programmability of cloud and carrier infrastructure. In: Proceedings of the 2014 Third European Workshop on Software Defined Networks, ser. EWSDN'14. IEEE Computer Society, pp 55–60