A Model for Distributed Service Level Agreement Negotiation in Internet of Things

Fan Li(✉), Andrei Palade(✉), and Siobhán Clarke(✉)

Trinity College Dublin, College Green, Dublin, Ireland
{fali,paladea,Siobhan.Clarke}@scss.tcd.ie

Abstract. Internet of Things (IoT) services can provide a comprehensive competitive edge compared to traditional services by leveraging the physical devices' capabilities through a demand-driven approach to provide a near real-time state of the world. Service provision in such a dynamic and large-scale environment needs to cope with intermittent availability of service providers, and may require negotiation to agree on Quality of Service (QoS) of a particular service. Existing negotiation approaches for IoT require a centralised perspective of the environment, which may not be practical given the scale and autonomy of service providers that rely on sensors deployed various environments to deliver their services. We propose a negotiation mechanism that uses distributed service brokers to dynamically negotiate with multiple IoT service providers on behalf of service consumers. The framework uses a hierarchical architecture to cluster service information and to manage the message flows during the negotiation process. Simulation results demonstrate the feasibility and efficiency of our proposal.

Keywords: Internet of Things · Distributed SLA negotiation · Negotiation protocol

1 Introduction

The IoT envisions that a large number of physical, potentially mobile devices, connected over the Internet, may provide a near real-time state of the world. The capabilities of each device can be abstracted through a well-defined interface and provided as a service [24]. Compared to traditional (cloud-based) services, service provisioning in such a dynamic and large-scale environment needs to cope with the intermittent availability of service providers, and may have flexible service quality demands and pricing options [8]. For example, compared to a weather forecasting application, a fire detection application has more stringent QoS demands on a smoke detection service and a temperature service. If a pay-as-you-go model is used, the same services can be delivered to different users with different service properties or QoS, by reconfiguring the services [5,14].

Mission-critical applications in transportation, health care, and emergency response services, require certain QoS guarantees to be provided to successfully

https://doi.org/10.1007/978-3-030-33702-5_7

deliver their services to stakeholders [23]. Traditionally, such applications have relied on a Service Level Agreement (SLA), which is a contract-like concept that formalizes the obligations and the guarantees of involved parties in the context of a particular service provisioning [13]. To tailor a service based on user's demand, and resolve possible conflicts between a service provider and consumer, a dynamic negotiation process is required where both parties express their own demands and preferences to arrive at a consensus before the actual service delivery. The output of this process is used to generate an SLA [21].

Compared to traditional (cloud-based) services, research on SLA negotiation in the IoT is still in the preliminary stage [17,19]. Because of the scale and the intermittent availability of geographically distributed service providers, the existing proposals do not address the negotiation problems that emerge because of frequent disconnections between service providers and insufficient awareness of local context such as service location [16]. The IoT is dynamic nature in terms of service providers' availability and mobility, unpredictable device status, and unstable wireless network conditions. Also, the data transmissions between devices and cloud, and the spontaneous interactions amongst devices may produce an enormous number of messages or events, which may further cause network congestion and reduce event processing capability. A lightweight negotiation protocol that considers the communication problems in a dynamic environment is needed for run-time IoT service negotiation. In our previous work, we assume a middleware is deployed on a set of edge devices, which uses a decentralized negotiation protocol to negotiate with candidate service providers on behalf of consumers [11]. The services are registered in the gateways that receive the registration requests, and the negotiation requests are forwarded by gateways to their neighbours until the request can be solved, or the maximum hop is reached. The simulation result shows that the purely decentralized architecture is not efficient enough to address large-scale and dynamic issues. Also, this experiment does not consider users' spatial requirements.

In this paper, we propose IoT-Negotiate, a negotiation model that enables distributed service brokers connected through an overlay network to manage the service information and control message flows during the negotiation process. The model uses a hierarchical topology to address the communication challenges in the environment, performs location-based data distribution and replication to enable an efficient message forwarding, and conducts distributed SLA negotiation with candidate service providers.

The remainder of this paper is organised as follows. Section 2 summarises the related work. Section 3 introduces the IoT-Negotiate mechanism. Section 4 describes the hierarchical overlay network of IoT-Negotiate and introduces the overlay network creation algorithm. Section 5 presents the service distribution mechanism. Section 6 illustrates the distributed SLA negotiation process. Section 7 details the experimental setup and evaluation results and Sect. 8 concludes the paper with a discussion about future research directions.

2 Related Work

Existing literature on SLA negotiation is limited, especially for dynamic, large-scale environments such as IoT. As one of the key area of building a negotiation component [26], the negotiation protocol has been discussed in different cloud projects. For instance, a set of messages were designed for QoS negotiation when delivering composite services [23]. However, this proposal moved the burden of negotiation from the end user to each atomic service. Karl Czajkowski *et. al.* [4] presented the Service Negotiation and Acquisition Protocol (SNAP) for negotiating access to different resources in a distributed system. However, it is too heavyweight and not flexible enough for automatic negotiation. Nabila *et. al.* [9] illustrated the generic alternating offers protocol proposed by Rubinstein, for bargaining between agents. Based on that, a set of extensions has been proposed to address different negotiation issues such as multilateral negotiation [2], or semantic-based approach [18]. FIPA Contract Net Interaction Protocol [6] (CNP) is another commonly-used negotiation protocol, which supports recursive negotiation to find a compromise [28]. Smith [22] described the semantics of exchanged information among the nodes in a distributed system under the assumption that each node can communicate with every other node. Misura et al. [16] proposed a cloud-based mediator platform where automatic negotiation is performed to find the conditions of data provision that are acceptable to application providers. Gaillard et al. [7] outlined a centralized SLA management component in WSN to guarantee the QoS parameters. However, this framework relies on human intervention to finish the negotiation process. Mingozzi et al. presented an SLA negotiation framework for M2M application [15]. However, the paper does not specify the detail of the negotiation mechanism, and the single request-reply interaction is insufficient for multi-round negotiation.

3 System Model

Consider a smart city environment where service providers deploy their services on resource-constrained, potentially mobile devices to capture data from the surrounding physical environment. Such service providers can provide a comprehensive competitive edge compared to traditional service provisioning techniques by leveraging the available services through a demand-driven approach to enable new applications for citizens such as real-time monitoring applications (e.g., traffic and real-time public transport services monitoring, particle concentration or noise pollution detection). Such services are generally developed using various approaches and technologies, and provide various QoS levels. SLA negotiation procedures may be required to achieve certain guarantees about the QoS of the application. Also, given the scale the environment, an automated procedure may be required as a manual approach may not be practical. To automate SLA negotiation for an urban-scale environment, we propose IoT-Negotiate, which is a distributed negotiation framework deployed on a set of devices deployed at the edge of the network. These devices can be mobile such a mobile handset or

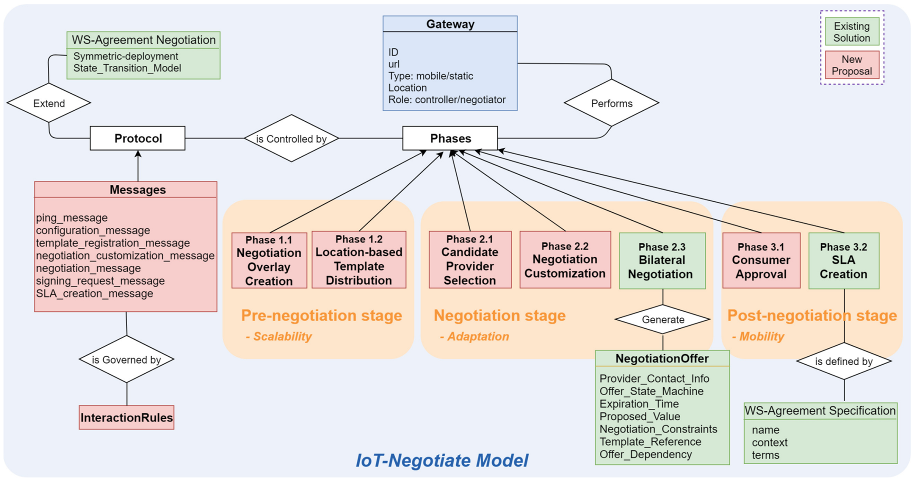


Fig. 1. IoT-negotiate ontology

fixed such as a workstation connected through WiFi or Ethernet. We refer to the devices as gateways. The negotiation procedure is performed by the fixed gateways because they are likely to have more stable and reliable network connections, whilst the mobile gateways are only used for forwarding messages.

To enable the demand-driven service provisioning vision, the service providers are required to provide the service properties with default values in SLA templates and publish the templates to the market. An SLA template is defined as $\text{SLAT} = \langle At_{id}, A_c, At_t, N_i, T, C \rangle$, consisting of template id At_{id}, agreement context A_c, template temporality At_t, negotiation information N_i, terms T (e.g., location, QoS, price, etc.) and constraints C, which can be regarded as a blueprint to create a valid negotiation offer based on user-specific demands [10]. Figure 1 shows the ontology of our proposed IoT-Negotiate model.

The IoT-Negotiate model composed of three stages: pre-negotiation, negotiation, and post-negotiation. Seven types of messages are designed to support different phases in the three stages:

Definition 1. ***Ping message****, which is defined as: $Ping = <S_{id}, R_{id}, \text{“hello”}>$, consisting of the sender identifier S_{id}, receiver identifier R_{id} and a “hello” string.*

Definition 2. ***Configuration message****, which is defined as: $Cf_{msg} = <O_p, S_{id}, R_{id}, m, ttl, Route>$, consisting of an operation code O_p, sender identifier S_{id}, receiver identifier R_{id}, message content m, the maximum number of hops ttl, and a routing table $Route$.*

Definition 3. ***Template registration message****, which is defined as: $Tr_{msg} = <O_p, SP_{id}, S_t>$, consisting of an operation code O_p, service provider identifier SP_{id}, and a template S_t.*

Definition 4. ***Negotiation customize message****, which is defined as: $Nc_{msg} = <Ni_{id}, Nr_{id}, cnt, S_t>$, consisting of the negotiation initiator identifier Ni_{id}, negotiation responder identifier Nr_{id}, negotiation context cnt (e.g., negotiation protocol, SLA schema, deadline, etc.), and the referred template S_t.*

Definition 5. ***Negotiate message****, which is defined as: $Ng_{msg} = <S_{id}, R_{id}, O, O_p, m>$, consisting of the sender identifier S_{id}, receiver identifier R_{id}, negotiation offers O, operation code O_p and message content m.*

Definition 6. ***Signing request message****, which is defined as: $Sr_{msg} = <G_{id}, SC_{id}, O_a, Route>$, consisting of the gateway identifier G_{id}, the consumer identifier SC_{id}, a list of acceptable offers O_a, and a routing table Route.*

Definition 7. ***Mobile entity locating message****, which is defined as: $Ml_{msg} = <S_{id}, R_{id}, E_{id}, m, O_p>$, consisting of the sender identifier S_{id}, the receiver identifier R_{id}, the entity identifier E_{id}, message content m, and operation code O_p.*

Definition 8. ***SLA creation message****, which is defined as: $Sc_{msg} = <Ai_{id}, Nr_{id}, O_a>$, consisting of the agreement initiator identifier Ai_{id}, agreement responder identifier Ar_{id}, and an offer signed by the service user O_a. The response should contain the reference of new pending SLA instance.*

In the **pre-negotiation** stage, a logic hierarchical negotiation overlay network (HNON) is dynamically created by exchanging ping and configuration messages (Phase 1.1, Sect. 4). The HNON manages SLA templates and controls the message flow during the negotiation process. The SLATs submitted by service providers are distributed in HNON according to service locations using template registration messages. The location-based message forwarding mechanism (Phase 1.2, Sect. 5) is designed based on the assumption that service providers are more likely to appear or move around the areas that are close to the advertised service location. If a gateway detects a local stored SLAT has the potential to meet QoS requirements, the gateway has a bigger chance to directly connected to the service provider (i.e. candidate service provider) to start a bilateral negotiation.

The **negotiation stage** begins when a user submits the request through a negotiation message. The message contains an offer expected by the user, which specifies the requested service location, QoS requirements and negotiation constraints. The message is forwarded to the gateways that are close to the requested location over the HNON. The gateways compare the request with local stored SLATs according to a unified SLA ontology (e.g., the WIoT-SLA ontology [10]) to search for candidate services (Phase 2.1). Once a candidate service is detected, a negotiation customization message (Phase 2.2) is sent to the service provider to initialize the negotiation instance before the bilateral negotiation (Phase 2.3). After the negotiation, the entity locating message may be used to locate mobile consumers if they cannot be contacted at the moment (Sect. 6).

In the **post-negotiation stage**, the negotiation results from different brokers are aggregated and the most optimized solution is selected and sent to the

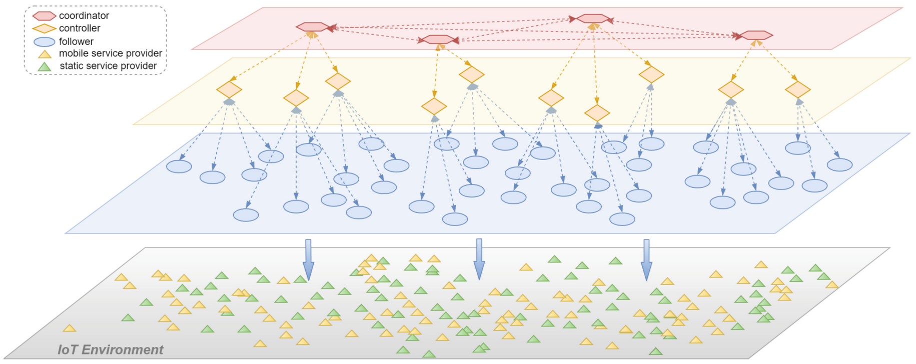


Fig. 2. Hierarchical negotiation overlay network

user through a signing request message (Phase 3.1). Once the negotiation result is approved by the user (i.e., the message is digitally signed), an SLA creation message is sent to the corresponding service provider to create a pending SLA (Phase 3.2).

4 Hierarchical Negotiation Overlay Network (HNON)

The HNON is a logistic negotiation overlay built upon the actual network topology. Figure 2 shows the three-layered structure of the proposed HNON. Each static gateway distributed in the environment is assigned to at least one of the following roles: follower, controller or coordinator. Each follower has a controller and each controller is associated with a coordinator. Followers compose the bottom layer of the overlay, which only work as brokers under the control of their controllers. Controllers compose the middle layer, which divides the environment into a set of sub-areas. Each controller can be regarded as a small data centre in the sub-area. It collects its followers' information and replicates local registered templates. The sub-area is referred to as the controller's range, which is roughly estimated by the maximum distance between the controller and its followers. The top layer is comprised of coordinators, which are directly connected with each other through the internet. A coordinator can be regarded as the access point of its controllers. In this hierarchical architecture, the follower layer guarantees a timely bilateral negotiation with service providers, the controller layer allocates negotiation tasks and improves the efficiency of template match-making process. The coordinator layer propagates messages over different sub-areas.

To create the HNON, the key requirement is to automatically assign different roles to gateways based on the network topology. To maximize the communication efficiency, we enforce that in each sub-area, the gateways which have the maximum number of wireless connections are assigned as controllers. Since the network topology is unknown for each gateway initially, the static gateways first broadcast ping messages through WiFi to identify their neighbouring gateways. Based on the number of replies, each gateway acquires the connection information and exchange this information with neighbours. As can be seen in Algorithm 1, each gateway initially caches itself as the controller and sends the controller's information through a configuration message (O_p is set to CIM) to search for a gateway that has the maximum number of connections in the nearby area (Line 1–6 in Algorithm 1(a)). Then, it waits for messages from other gateways and updates the cache if the controller specified in the received message has more connections than the cached one, or a shorter route has been detected for the same controller. If the cache is updated and the TTL message has not been reached, the controller's information is further propagated to neighbours (Line 2–17 in Algorithm 1(b)). Here the message's TTL controls the message propagation range, which is also the range of each sub-area. When the message propagation time is due, it sends a verification to the controller through a Cf_{msg} whose O_p is set to CVM (Line 7–10 in Algorithm 1(a)). The controller saves the follower's information and updates its range (Line 18–21 in Algorithm 1(b)). If the controller can not access to the Internet, it multicasts a Cf_{msg} (O_p is set to RIM) to neighbours to search for an internet-connected gateway in the sub-area, and sends a verification to it when the time is due (Line 13–22 in Algorithm 1(a)). Once a gateway is allocated as a new coordinator, it saves controllers' information and collects other coordinators' information by broadcasting a Cf_{msg} (O_p is set to FRM) through the Internet (Line 23–26 in Algorithm 1(a), Line 22–35 in Algorithm 1(b)).

5 Location-Based Template Distribution

As we mentioned in Sect. 3, the location-based template distribution mechanism is performed when a service provider advertises its service by sending the SLA template to a nearby gateway through a Tr_{msg} (O_p is set to ADV). The template is forwarded over the HNON and stored in the gateways that are within or close to the service location. Figure 3 shows the template distribution process, the message is first forwarded to a coordinator. The coordinator computes the distance between the service location and its controllers' locations, and forwards the message to controllers whose range is within the service coverage (O_p is set to $CREG$). Controllers compute the distance between service location and their followers' locations, cache the follower that has minimum distance and reply the distance to its coordinator. The coordinator caches the controller that has the minimum distance, adds the distance (d_f) to the message content and sends the message to other coordinators (O_p is set to TRG). If the coordinator does not have any controllers whose range is within the service coverage, the d_f is set as the minimum distance between the service location and its controllers' locations. Other coordinators perform the same process and reply the minimum distance if it is not greater than the d_f specified in the received message. When time is due, the originating coordinator forwards the template to the cached controller if there is no reply (O_p is set to $FREG$), or forwards the template to the coordinator that replies minimum d_f (O_p is set to $RREG$). Then the template is further forwarded to the cached follower so that it can be saved in the

Algorithm 1: HNON Creation Algorithm (a) - Message Sender

```
1  Cache itself as controller;
2  Create $Cf_{msg}$ ($Cf_{msg}.m$ ←cached controller info, $Cf_{msg}.O_p$←CIM);
3  Set Timer T;
4  Send $Cf_{msg}$ to neighbours;
5  /* Waiting for responses */
6  if receives a reply: Check if the cached controller needs to be updated;
7  if T expires and the cached controller is not itself:
8      Create $Cf_{msg}$ ($Cf_{msg}.m$ ←self info, $Cf_{msg}.O_p$ ←CVM);
9      Mark itself as a follower, send $Cf_{msg}$ to cached controller;
10     if receives the ACK: Save controller's identifier and routing info;
11 if T expires and the cached controller is itself:
12     Mark itself as a controller;
13 if current gateway is a controller and no Internet connection:
14     Create $Cf_{msg}$ ($Cf_{msg}.m$ ←self info, $Cf_{msg}.O_p$ ←RIM);
15     Set Timer T;
16     Send $Cf_{msg}$ to neighbours;
17     /* Waiting for responses */
18     if receives a reply: Check if the cached coordinator needs to be updated;
19     if T expires:
20         Create $Cf_{msg}$ ($Cf_{msg}.m$ ←self info, $Cf_{msg}.O_p$ ←RVM);
21         Send $Cf_{msg}$ to cached coordinator;
22         if receives the ACK: Save coordinator's identifier and routing info;
23 if current gateway is a controller and have Internet connection:
24     Create $Cf_{msg}$ ($Cf_{msg}.m$ ←self info, $Cf_{msg}.O_p$ ←FRM);
25     Mark itself as a coordinator, broadcast $Cf_{msg}$;
26     if receives a reply: Add the responder to coordinator list;
```

gateway that closest to the service location (O_p is set to REG). All the registered templates are also replicated in corresponding controllers. If a service provider is mobile and specifies a flexible service location, the SLA template is stored in the gateway that receives the request. The provider re-submits the request when moving more than a pre-defined distance. All the registered templates are periodically checked by gateways to remove the ones that are out of date or the providers are unreachable. Also, a provider can change their offerings after registration by submitting a new template with the same identifier but different creation timestamp. The updated template is submitted to a nearby gateway through a Tr_{msg} (O_p is set to UPD). Similarly, this message is forwarded to a coordinator and multicasted to all coordinators. Each coordinator forwards the message to its controllers to detect if the originating template is stored in their local areas and update it if so. If the service location is changed in the updated template, the same template distribution process will be performed to search for the closest gateway, and the old template will be deleted.

Algorithm 2: HNON Creation Algorithm (b) - Message Receiver

```
1  /* listening configuration messages */
2  if $Cf_{msg}.O_p$ =CIM:
3      resvCon ← $Cf_{msg}$.getMessageContent().getControllerConnections();
4      resvRoute ← $Cf_{msg}$.getRoute();
5      if resvCon > cached controller's connections:
6          Update cache with received controller's info;
7          Set state into active;
8      else if $Cf_{msg}$ specifies a shorter route for a same controller:
9          Update cache with resvRoute;
10         Set state into active;
11     else: Set state into inactive;
12     if state is active and TTL is not reached :
13         Add self identifier to resvRoute;
14         Create $Cf_{msg}$ ($Cf_{msg}.m$ ←cached controller info, $Cf_{msg}.O_p$←CIM,
           resvRoute);
15         Send $Cf_{msg}$ to neighbours;
16         Set state into inactive;
17     else: Set state into inactive;
18 if $Cf_{msg}.O_p$ =CVM:
19     Mark itself as a controller, update range;
20     Mark sender as a follower, save follower's identifier, location and route;
21     Send back ACK;
22 if $Cf_{msg}.O_p$ =RIM:
23     if current gateway has internet connection:
24         Reply with self identifier and routing info;
25     else if TTL is not reached:
26         add self identifier to message's routing table;
27         forward the message to neighbours;
28 if $Cf_{msg}.O_p$ =RVM:
29     Mark itself as a coordinator, mark sender as a controller ;
30     Save controller's identifier, location, range and route;
31     Send back ACK;
32 if $Cf_{msg}.O_p$ =FRM:
33     if current gateway is a coordinator:
34         Save sender's identifier;
35         Reply with self identifier;
```

6 Distributed SLA Negotiation

The SLA negotiation process is mainly composed of three phases: (i) request forwarding and template match-making; (ii) negotiation customization with the candidate service providers; (iii) distributed bilateral negotiation and mobile consumer locating.

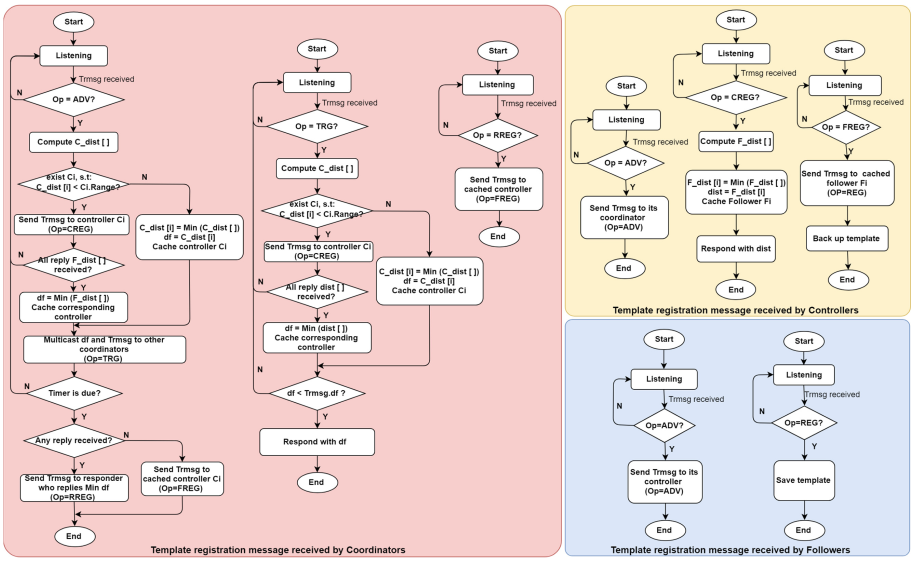


Fig. 3. Processing of template registration messages

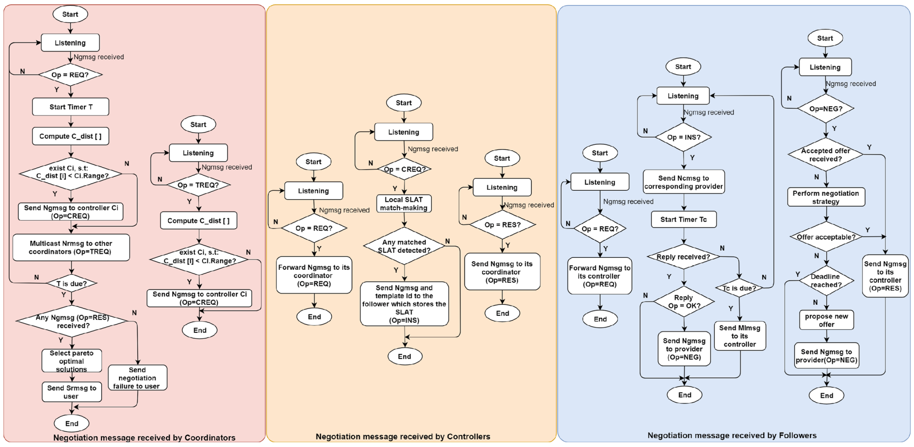


Fig. 4. Processing of negotiation messages

6.1 Negotiation Request Forwarding

Figure 4 shows the location-based request forwarding mechanism: a user submits a request through a negotiation message (O_p is set to REQ). The message is first forwarded to the coordinator layer (O_p is set to $TREQ$), then propagated to the controllers whose range covers the requested location (O_p is set to $CREQ$). Controllers match the request with local backup templates to search for the candidate services that have the potential to satisfy all the QoS requirements and forward the providers' information to the followers that actually store the templates (O_p is set to *INS*) to initialize the negotiation instance. Based on the negotiation information provided in the template, the follower sends a customization message to the provider's negotiation interface to test provider's availability and customize the negotiation context (i.e. negotiation protocol, SLA schema, template). Then the follower generates an initial offer according to the constraints specified in the request and the template. The bilateral negotiation phase starts when the follower sends the offer to the service provider through a negotiation message (O_p is set to *NEG*). During the phase, follower negotiates with the service provider by exchanging offers in an orderly way [25]. Each time a new offer is proposed by the service provider, the follower makes decisions (i.e., accept/reject the received offer, or propose a new offer) according to a predefined negotiation strategy[1]. The bilateral negotiation stops when the deadline is reached, or an offer is accepted. During the negotiation customization or consumer approval phase, it is possible that the WiFi-connected mobile negotiating parties (i.e., service providers or users) move to another place and lose the original network connection. The mobile entity locating message is created and sent to the local controller to detect the negotiation parties (O_p is set to *INQ*). The local controller propagates the message to its followers (O_p is set to *FINQ*) and each follower sends a ping message to the entity to test the connection. If any follower receives an ACK, it forwards the message content to the entity. Otherwise, the local controller forwards the message to its coordinator(O_p is set to *RINQ*), the coordinator propagates the message to its other controllers (O_p is set to *CINQ*) to detect the entity in different managed sub-areas. If the consumer cannot be connected in any sub-area, the coordinator forwards the message to other coordinators (O_p is set to *NINQ*) to start an exhaustive searching over the whole network. This design guarantees that the entity locating process is firstly performed in the local sub-area, then the nearby sub-areas, and then the whole network.

7 Evaluation

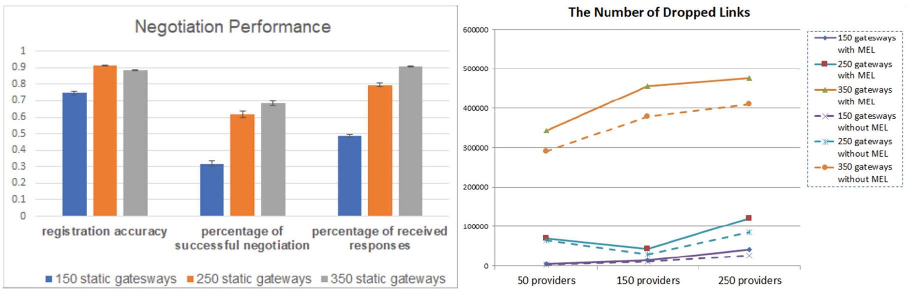
To test the performance of IoT-Negotiate model, we implemented it using *Simonstrator* [20], which is a peer to peer simulator for distributed mobile applications. The environment is configured as Dublin city center where a set of static gateways are randomly distributed and connected through WiFi. 50% gateways have Internet connections. Service providers (mobile or static) and consumers (mobile) are initialized randomly in the environment. The consumers and autonomous service providers are connected to the network by WiFi while web service providers connect with the Ethernet. Based on an existing OWLS-SLR dataset [1] and the IoT services examples proposed in related literature [3,27], we create 436

[1] The discussion of negotiation strategy is out of this paper's scope, more details about the evaluation of received offers and the corresponding decision-making model can be found in [12].

service prototypes which specify the service name, domain information, functional and non-functional properties (i.e., location and QoS parameters). Based on the prototypes, we generated different SLA templates for each provider by randomly assigning values to negotiable QoS parameters based on a predefined variation range. The variation range guarantees that the conflict between the service provider and a consumer is resolvable. In other words, it guarantees a successful negotiation if the services provider receives the request. In this experiment, the maximum hop of messages is set to 8 and the negotiation timeout is set to 2 min. Considering the possible network congestion, gateways use the UDP send-and-reply mode to send messages, the maximum time to wait for the reply is 2 s. Since the autonomous service providers are likely to be online and offline at any time, the *Churn* model provided by *Simonstrator* to model the connectivity of peers is adopted to simulate the availability of hosts, and the mobile entities follow a random movement pattern with a pre-defined moving speed varying from 16.7 m/s to 27.8 m/s (i.e., car speed).

Figure 5(a) shows the simulation results when the number of static gateways increases from 50 to 250 while the number of service providers is set to 150 (i.e., 50 mobile service providers and 100 static service providers). 100 consumers periodically submit requests to nearby gateways within 100 min. The negotiation result is evaluated using three metrics: template registration accuracy, percentage of successful negotiation, and the percentage of signing request messages received by the user. The template registration accuracy measures if the template has been correctly registered into the proper gateways. If the distance between the service location and the registering gateway is within 500 m, we regard it as a correct registration. The percentage of successful negotiations measures the ability to achieve a successful negotiation when potential solutions are existing in the environment. Since we adopt a negotiation strategy that guarantees a successful negotiation when consumers and providers have overlapped negotiation space, this metric measures if a candidate service provider can be successfully detected and contacted with. The percentage of signing request messages received by the user measures the efficiency of the mobile entity locating mechanism when returning the negotiation result back to the user.

The simulation result shows that the registration accuracy is around 75% when there are only 50 gateways deployed in the environment. The incorrect registrations are mainly caused by two reasons: (i) some controllers cannot find any coordinators to propagate messages. When we increase the percentage of internet-connected gateway from 50% to 90%, the registration accuracy increases to 97.2%. This is also approved by the result that the registration accuracy improves as the number of deployed gateways increasing. (ii) the SLA templates are more likely to be forwarded to the controllers that have large ranges, but the gateway finally registers the template may not be the closest gateway. This implies that the SLA templates may be stored in incorrect places if the gateways are not evenly distributed. The success rate is not as high as we expected: the percentage of successful negotiations is only about 31.7% when there are 150 gateways deployed in the environment. The main reason that causes the



(a) Negotiation Performance (b) The number of dropped links

Fig. 5. Simulation result

negotiation failures is negotiation timeout, which includes following situations: (i) mobile entities can not connect to any gateway during the negotiation process, or the area they are moving around does not have any internet-connected gateway;(ii) the candidate service is not registered in the right place since the controller which receives the registration request can not find a coordinator to propagate the request. When we increase the percentage of internet-connected gateway from 50% to 90%, the success rate increases from 31.7% to 72%. If we assume a majority of gateways are internet-connected, this result is acceptable. However, if most of the gateways are WiFi-connected, increasing the number of gateways can reduce the type of failures to some extent but can not eliminate the negative impact because the mobile entity locating mechanism highly relies on the interactions between gateways of different layers. Any lost response may cause a large number of simultaneous interactions when the number of gateways is large, which further increases the risk of losing packages. Figure 5(b) shows the comparison of the number of dropped links with and without mobile entity locating process (MEL). This result implies that the current mobile entity locating mechanism is not lightweight enough to locate entities that move fast in the environment. A mechanism that can trace and manage mobile negotiating entities may be more efficient to address the communication problem.

However, compared to our previous work, by adopting the hierarchical negotiation overlay network, we have decreased the maximum negotiation time from 10 min to 2 min, and the users' spatial requirements are considered as well.

8 Conclusion and Future Work

In the IoT environment, there is potential for a range of different types of devices to provide their functionalities as services and tailor the services' properties based on user-specific requirements. However, the demand-driven IoT service provisioning requires an SLA negotiation between consumers and service providers. In a distributed large-scale environment like the IoT, a middleware that can automatically negotiate with candidate service providers on behalf of users is needed. This paper proposes a distributed SLA negotiation model in the IoT environment, which uses a hierarchical negotiation overlay network to cluster service information and to manage the message flows during the negotiation process. Although the simulation results demonstrate the feasibility and efficiency of the negotiation model, it still shows some limitation in term of addressing mobility problems. In future work, we plan to optimize the negotiation model by designing a more lightweight mobile entity locating mechanism. This might be improving the hierarchical negotiation overlay network so that it can trace and predict the location of mobile negotiation parties in real-time without introducing much wireless communication.

Acknowledgment. This work was funded by Science Foundation Ireland (SFI) under grant 13/IA/1885.

References

1. OWLS-SLR - Datasets. http://lpis.csd.auth.gr/systems/OWLS-SLR/datasets.html
2. Aydoğan, R., Festen, D., Hindriks, K.V., Jonker, C.M.: Alternating offers protocols for multilateral negotiation. In: Fujita, K., et al. (eds.) Modern Approaches to Agent-based Complex Automated Negotiation. SCI, vol. 674, pp. 153–167. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-51563-2_10
3. Cabrera, C., Palade, A., Clarke, S.: An evaluation of service discovery protocols in the internet of things. In: Proceedings of the Symposium on Applied Computing, pp. 469–476. ACM (2017)
4. Czajkowski, K., Foster, I., Kesselman, C., Sander, V., Tuecke, S.: SNAP: a protocol for negotiating service level agreements and coordinating resource management in distributed systems. In: Feitelson, D.G., Rudolph, L., Schwiegelshohn, U. (eds.) JSSPP 2002. LNCS, vol. 2537, pp. 153–183. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-36180-4_9
5. Elfatatry, A., Layzell, P.: Negotiating in service-oriented environments. Commun. ACM **47**(8), 103–108 (2004)
6. FIPA, F.f.I.P.A.: FIPA Contract Net Interaction Protocol Specification. Architecture (SC00029H), 9 (2002). http://www.mit.bme.hu/projects/intcom99/9106vimm/fipa/XC00029E.pdf
7. Gaillard, G., Barthel, D., Theoleyre, F., Valois, F.: Service level agreements for wireless sensor networks: a WSN operator's point of view. In: 2014 IEEE/IFIP Network Operations and Management Symposium (NOMS), pp. 1–8. IEEE (2014)
8. Grubitzsch, P., Braun, I., Fichtl, H., Springer, T., Hara, T., Schill, A.: ML-SLA: multi-level service level agreements for highly flexible IoT services. In: 2017 IEEE International Congress on Internet of Things (ICIOT), pp. 113–120. IEEE (2017)
9. Hadidi, N., Dimopoulos, Y., Moraitis, P.: Argumentative alternating offers. In: McBurney, P., Rahwan, I., Parsons, S. (eds.) ArgMAS 2010. LNCS (LNAI), vol. 6614, pp. 105–122. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21940-5_7
10. Li, F., Cabrera, C., Clarke, S.: A WS-agreement based SLA ontology for IoT services. In: Issarny, V., Palanisamy, B., Zhang, L.-J. (eds.) ICIOT 2019. LNCS, vol. 11519, pp. 58–72. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-23357-0_5

11. Li, F., Clarke, S.: Service negotiation in a dynamic IoT environment. In: Proceedings of the Service-Oriented Computing Workshop (ICSOC) (2018)
12. Li, F., Clarke, S.: A context-based strategy for SLA negotiation in the IoT environment. In: 2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops). IEEE (2019)
13. Ludwig, H., Keller, A., Dan, A., King, R.P., Franck, R.: Web service level agreement (WSLA) language specification, pp. 815–824. IBM Corporation (2003)
14. Menascé, D.A.: QoS issues in web services. IEEE Internet Comput. **6**(6), 72–75 (2002)
15. Mingozzi, E., Tanganelli, G., Vallati, C.: A framework for QoS negotiation in things-as-a-service oriented architectures. In: 2014 4th International Conference on Wireless Communications, Vehicular Technology, Information Theory and Aerospace & Electronic Systems (VITAE), pp. 1–5. IEEE (2014)
16. Misura, K., Zagar, M.: Internet of Things cloud mediator platform. In: 2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), pp. 1052–1056. IEEE (2014)
17. Palade, A., et al.: Middleware for Internet of Things: an evaluation in a small-scale IoT environment. J. Reliable Intell. Environ. **4**, 3–23 (2018)
18. Ragone, A., Di Noia, T., Di Sciascio, E., Donini, F.M.: Alternating-offers protocol for multi-issue bilateral negotiation in semantic-enabled marketplaces. In: Aberer, K., et al. (eds.) ASWC/ISWC -2007. LNCS, vol. 4825, pp. 395–408. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-76298-0_29
19. Razzaque, M.A., Milojevic-Jevric, M., Palade, A., Clarke, S.: Middleware for Internet of Things: a survey. IEEE Internet Things J. **3**(1), 70–95 (2016)
20. Richerzhagen, B., Stingl, D., Rückert, J., Steinmetz, R.: Simonstrator: simulation and prototyping platform for distributed mobile applications. In: Proceedings of the 8th International Conference on Simulation Tools and Techniques (SIMUTOOLS), pp. 99–108. ACM, August 2015
21. Saravanan, K., Rajaram, M.: An exploratory study of cloud service level agreements-state of the art review. KSII Trans. Internet Info. Syst. **9**(3) (2015)
22. Smith, R.G.: The contract net protocol: high-level communication and control in a distributed problem solver. IEEE Trans. Comput. **12**, 1104–1113 (1980)
23. Swiatek, P., Rucinski, A.: Iot as a service system for eHealth. In: 2013 IEEE 15th International Conference on e-Health Networking, Applications & Services (Healthcom), pp. 81–84. IEEE (2013)
24. Thoma, M., Meyer, S., Sperner, K., Meissner, S., Braun, T.: On IoT-services: survey, classification and enterprise integration. In: 2012 IEEE International Conference on Green Computing and Communications (GreenCom), pp. 257–260. IEEE (2012)
25. Waeldrich, O., Battré, D., Brazier, F.F., Clark, K., Oey, M., Papaspyrou, A., Wieder, P., Ziegler, W.: WS-Agreement Negotiation Version 1.0, p. 64 (2011)
26. Yao, Y., Ma, L.: Automated negotiation for web services. In: 2008 11th IEEE Singapore International Conference on Communication Systems. ICCS 2008, pp. 1436–1440. IEEE (2008)
27. Zanella, A., Bui, N., Castellani, A., Vangelista, L., Zorzi, M.: Internet of things for smart cities. IEEE Internet Things J. **1**(1), 22–32 (2014)
28. Zulkernine, F., Martin, P., Craddock, C., Wilson, K.: A policy-based middleware for web services SLA negotiation. In: 2009 IEEE International Conference on Web Services. ICWS 2009, pp. 1043–1050. IEEE (2009)