An Urban-driven Service Request Management Model

Christian Cabrera, Andrei Palade, Gary White, and Siobhán Clarke
Distributed Systems Group, SCSS, Trinity College Dublin, Dublin, Ireland
Email: {cabrerac,paladea,whiteg5,Siobhan.Clarke}@scss.tcd.ie

***Abstract*—Pervasive applications in smart cities rely on a large number of IoT devices, which are deployed in large geographic areas. Smart cities can manage these devices using Service-Oriented Architectures (e.g., micro-services) by encapsulating devices capabilities as IoT services. Distributed service discovery architectures reduce search spaces and perform discovery processes closer to consumers on edge devices. However, request management, a key task in distributed service discovery, is still challenging because requests must be forwarded through large networks where nodes have partial knowledge about other participants. Previous research has shown that social-based and bio-inspired methods can be used to manage requests in small-scale environments, but such approaches do not scale to large environments. This paper adds urban context to a social-based and bio-inspired mechanism to forward requests where they are most likely to be solved. Results show that our model has the best rate of solved requests, and intermediate latency.**

***Index Terms*—Internet of Things, Smart Cities, Service Discovery, Context-aware, Bio-inspired, Social-based.**

I. Introduction

The Internet of Things (IoT) integrates a large number of devices in smart cities to provide pervasive applications that support citizens' activities [1]. Service oriented architectures (SOAs) are commonly used to manage these devices by abstracting their capabilities as IoT services [2] (e.g., micro-services). Services are self-contained pieces of software that offer functionalities according to a specification (e.g., a service that reports the wind speed in the city centre). SOAs include processes to register, discover, compose, execute, and monitor services, enabling the development of applications [3]. Service discovery is a process for SOAs in IoT as services need to be located before using them [4]. Web services discovery has been widely explored [5]. However, service discovery is still challenging in smart cities [6]. The IoT encapsulates millions of services that generates enormous search spaces where efficient discovery is difficult. Existing research has addressed these large search spaces by organising services in distributed architectures. Our previous work manages services in a network of IoT gateways based on the city places that surround them [7], [8]. This urban-based organisation improved the discovery efficiency compared against approaches based on services' attributes. However, efficiency is still affected because nodes fail to forward requests to relevant search spaces. Architectures in large environments must forward requests where they are most likely to be solved as it is impractical to scan all service repositories [9].

Research on IoT service discovery has proposed to manage requests based on social-based or bio-inspired methods. Social-based approaches use relationships between devices to forward requests to relevant communities [10], [11]. Bio-inspired approaches take ideas from nature to propagate requests to desired destinations (e.g., ants colony) [11], [12]. However, social-based and bio-inspired approaches have issues in large networks because they are based on flooding strategies [13]. Such strategies usually depend on a time to live (TTL) parameter that limits data propagation. A TTL parameter in large networks can cause network overhead when its value is high. Or, data propagation can fail to reach target nodes if the TTL value is low and the knowledge that drives the propagation is not enough. Contextual information can enrich these approaches to make well-informed decisions, when propagating information in large environments. This paper proposes to add urban context to a social-based strategy and a bio-inspired mechanism to forward requests to the most promising search spaces in large smart city environments. We evaluate our model in a simulated city environment and measure the rate of solved requests, simulated response time, number of hops, and exchanged discovery messages. Results are compared against four baselines, two of them are social-based [10], [14], one is bio-inspired [12], and one is hybrid (i.e., social-based and bio-inspired) [11].

II. Related Work

Current research on IoT service discovery has used social-based or bio-inspired methods to forward consumers' requests.

A. Social-based Management

Fredj et al. [10], [15] propose to cluster services according to their location. This approach uses a hierarchical network that represent different smart spaces and host service descriptions. Nodes are connected in a tree topology, where the lowest level represents spaces containing physical connected devices and upper-levels represent spaces that include lower-levels. This structure is used to forward requests at discovery time. CORDIAL. [16] is an algorithm for service discovery that forwards requests in a distributed network of smart phones according to human periodic movements and interests. CORDIAL relies in the assumption that nodes with similar interests tend to meet more frequently. TSSD is a model for service discovery that exploits the temporal-spatial correlation between mobile nodes [14]. This model creates communities,

978-1-7281-4657-7/20/$31.00 ©2020 IEEE

which evolve over time to reflect nodes interests. TSSD forwards requests in these communities with an epidemic routing. Corbellini et al. [17] propose to create clusters that group services according to users' interests. This model calculates similarity between services using neighbour-based metrics, which assume that nodes are more likely to link if they share neighbours. This structure is used to forward requests to the more relevant servers. Xia et al. [18] propose a service discovery mechanism that mimics human-like behaviour. This approach forwards requests to the most relevant subset of devices according to a correlation degree.

Service discovery based on social networks suits mobile environments because they mimic human social features to forward requests in peer to peer networks by using flooding-based strategies. These strategies affect the discovery efficiency because they are likely to fail when delivering messages in networks, where sources and targets might never have a direct link between them [13]. More informed strategies can forward requests to relevant devices in short time [13].

B. Bio-inspired Forwarding

Ebrahimi [19] present an approach that uses an ant-based algorithm to create clusters based on sensors' QoS attributes. This approach extracts required attributes from consumers' queries and forwards them to the most relevant clusters. Rapti et al. [12] propose a service discovery approach based on artificial potential fields (APFs). Each provider has an APF, and its strength depends on the type of requests that the provider solved in the past. Providers apply attraction or repulsion forces to requests according to APFs to mimic electrically charged particles. These forces drive requests to the most promising providers in IoT networks. Wanigasekara et al. [20] introduce an approach based on usage patterns to support service composition in medium IoT environments such as smart buildings. Service discovery is modelled as a contextual bandit problem. Services are the set of bandits, the reward is based on the service usage (i.e., positive reward if the user selects the service), and the expected pay-off represents how many times the service was successfully selected. The approach maximises the expected pay-off by recommending relevant IoT resources. Yuan et al. [11] combines a social-based strategy with a bio-inspired approach to manage requests in a self-organised network. They build an overlay that manages services, according to the nodes interactions (i.e., friendship). A score is calculated for the associations between nodes, and provides an adaptive request forwarding to discover the shortest paths to a desired service.

Bio-inspired approaches propagate requests in an effective fashion even when nodes have a limited knowledge about other nodes. However, they also use flooding strategies in smaller IoT environments. Urban context has the potential to mitigate the remaining limitations of social-based and bio-inspired strategies in large scale environments. The addition of this knowledge enables to make well informed decisions to forward requests where they are most likely to be solved.

III. Request Management Model

We propose an urban-driven, social-based, bio-inspired request management model for smart cities, which works in a network of IoT gateways that manages services and attends requests. Gateways can be static or mobile and have a local registry where descriptions and other gateways information are stored. Each gateway exchanges messages with providers, consumers, and other gateways, and uses a planner to search for services in the local repository [21]. A gateway forwards requests to other gateways when it cannot solve them locally. The proposed approach uses urban context (i.e., surrounding places) to define gateways interests. A social-based strategy uses these interests to configure the network of gateways, register services, and attend discovery requests. This urban context drives a bio-inspired method based on an $ant\ colony$ mechanism [22] to forward requests in the network. We define a **gateway** as $gw = \langle gw_{id}, gw_{loc}, SP, D, GR, GWS, P, C \rangle$, where gw_{id} is its identifier, and gw_{loc} is its location. The set SP represents the city places that surround the gateway. The set D represents the gateway's domains, which are defined based on SP. The set GR represents the relevance of the gateway for each domain in D, each relevance is computed using the equation 1 according to our previous work [7].

$$rel(gw, d) = \frac{ncs}{ds} \tag{1}$$

where ncs is the number of city services of domain d that the surrounding places of gw offer, and ds is the total number of domains of all these city services. The set GWS is the list of other gateways in the network that the gateway gw knows. P represents the pheromone values for the links between gw and the gateways in GWS. Each gateway updates these pheromone values according to its interactions with other gateways in the network initialisation, service registration, and service discovery processes. The set C represents the cost for sending a message from gw to each gateway in the set GWS.

A. Network Initialisation

Gateways configure the network by sending their information to other gateways. Each gateway uses a **gateway advertisement** message that contains the identifier of the receiver gateway, the information of the sender gateway, and the number of hops of the message to avoid network overhead. When a gateway gw_i receives an advertisement message from another gateway gw_j (Algorithm 1), gw_i stores the data of gw_j(Lines 5 and 6). It updates the pheromones information by creating an entry for each domain of gw_j with its respective relevance score (Lines 7 and 8). Gateways initialise pheromone values based on the urban context represented by the relevance score, which specifies how relevant is gw_j for each domain. These pheromone values enable the integration of the urban context of each gateway to be used in the discovery process. The gateway gw_i also updates the communication cost by creating an entry for gw_j with the respective physical distance (Line 10). Finally, gw_i advertises the gateway gw_j to other gateways, if the number of message hops is less than the

Algorithm 1 Network Initialisation.

```
1:  function MESSAGE ARRIVES(msg)
2:      if msg is GwAdv_msg then
3:          gw_j ← msg.gw
4:          hops ← msg.h
5:          if gw_j not in GWS then
6:              GWS.add(gw_j)
7:              for each d ∈ gw_j.D do
8:                  τ(d)_{gw_i,gw_j} ← rel(gw_j, d)
9:              end for
10:             L_{gw_i,gw_j} ← distance(gw_i, gw_j)
11:             hops ← hops + 1
12:             if hops <= hopsLimit then
13:                 destinations ← byInterest(gw_j.D)
14:                 for each gw ∈ destinations do
15:                     sendMessage(GwAdv_msg, gw)
16:                 end for
17:             end if
18:             sendMessage(GwAdv_msg, gw_j)
19:         end if
20:     end if
21: end function
```

limit of hops. These destinations are selected according to the social-based strategy that defines how interesting is gw_j to other gateways (Lines 10 to 17). The **interest** of a gateway gw_k in a gateway gw_j follows equation 2.

$$interest(gw_k, D) = \sum_{x=1}^{n} rel(gw_k, d_x) \tag{2}$$

where d_x belongs to D, which is the list of domains of the surrounding places of the gateway gw_j. This equation reflects that a gateway gw_k is interested in gw_j, if gw_k is relevant for one or more domains from D. Finally, gw_i responds to gateway gw_j by sending its information (Line 18).

B. Services Registration

Gateways register services when they receive registration messages from providers or other gateways. A **registration message** has the identifier of the receiver gateway, the service description to be registered, and message limit of hops. Once a gateway gw_i receives a registration message from a provider p, or a gateway gw_j, to register a description s_{desc}, gw_i notifies p or gw_j about the registration success. This response contains the receiver identifier, the information of the gateway that registered the service, and the service description that was registered. Algorithm 2 shows the process that gateways perform when they receive a registration request (Lines 2 to 20). When a gateway gw_i receives a registration message, it validates if the received description s_{desc} already exists in its repository. gw_i stores s_{desc} and sends a registration response, if the service description is not in the repository and shares domains with gw_i (Lines 6 to 12). Gateways advertise s_{desc} by sending the description to other gateways

Algorithm 2 Service Registration.

```
1:  function MESSAGE ARRIVES(msg)
2:      if msg is Reg_msg then
3:          s_desc ← msg.s_desc
4:          hops ← msg.h
5:          gw_j ← msg.sender
6:          if s_desc not in localRepository then
7:              SD ← s_desc.D
8:              if shareDomains(D, SD) then
9:                  localRepository.insert(s_desc)
10:                 sendMessage(RegRes_msg, gw_j)
11:             end if
12:         end if
13:         hops ← hops + 1
14:         if hops <= hopsLimit then
15:             destinations ← byInterest(s_desc.D)
16:             for each gw ∈ destinations do
17:                 sendMessage(Reg_msg, gw)
18:             end for
19:         end if
20:     end if
21:     if msg is RegRes_msg then
22:         gw_j ← msg.sender
23:         s_desc ← msg.s_desc
24:         for each d ∈ s_desc.D do
25:             τ(d)_{gw_i,gw_j} ← τ(d)_{gw_i,gw_j} + ρ
26:         end for
27:     end if
28: end function
```

(Lines 13 to 19). Destination gateways are selected according to their interest in the description following the equation 2, where D is the list of service domains. These interests (i.e., social-based) enable each gateway to advertise descriptions to relevant gateways based on their surrounding places (i.e., urban context). The gateway gw_i updates its pheromones' values when a registration response arrives (Lines 21 to 27). It increases the pheromones value by adding ρ for each domain in the s_{desc} that was registered by gw_j. This increment reinforces the links between gw_i and gw_j for the service domains (Lines 24 to 26) to be used in the discovery process.

C. Services Discovery

The discovery process starts when a request arrives. A **request** contains the identifier of the receiver gateway, the service request, the consumer, the set of gateways that the request has visited, the set of partial solutions that previous gateways have discovered, and the hops limit for the message. Gateways send **request response messages** when they solve a request. This message has the receiver identifier, the consumer identifier, the request, the list of plans that solve the request, and the set of gateways that participated in the discovery process. Algorithm 3 shows the process when a gateway gw_i receives a request (Lines 7 to 27). The gateway first searches for services in its repository to solve the request (Line 8). If

Algorithm 3 Request Manager.

```
1: function MESSAGE ARRIVES(msg)
2:     r ← msg.r
3:     c ← msg.c
4:     PGWS ← msg.PGWS
5:     PSOL ← msg.PSOL
6:     hops ← msg.h
7:     if msg is Req_msg then
8:         SOL ← heuristicPlanning(r, PSOL)
9:         PGWS.add(gw_i)
10:        if solved then
11:            sendMessage(DiscRes_msg, c)
12:            if ¬received then
13:                PGWS.remove(gw_i)
14:                gw ← getPreviousGateway(PGWS)
15:                sendMessage(DiscRes_msg, gw)
16:            end if
17:        else
18:            PSOL.add(SOL)
19:            hops ← hops + 1
20:            if hops <= hopsLimit then
21:                destinations ← byPheromones(r.D)
22:                for each gw ∈ destinations do
23:                    sendMessage(Disc_msg, gw)
24:                end for
25:            end if
26:        end if
27:    end if
28:    if msg is ReqRes_msg then
29:        sendMessage(DiscRes_msg, c)
30:        if ¬received then
31:            PGWS.remove(gw_i)
32:            gw ← getPreviousGateway(PGWS)
33:            sendMessage(DiscRes_msg, gw)
34:        end if
35:    end if
36: end function
```

the request is solved, gw_i sends the response to the consumer. If the consumer does not receive the response, gw_i sends the response to other participant in the discovery process (Lines 12 to 16). If the request is not solved, gw_i forwards the request and the partial solutions to other gateways. It selects the destinations using the pheromones and cost information as inputs for the forwarding mechanism, which selects gateways according to their **potential** to solve a request (Equation 3).

$$potential(gw_j, D) = \sum_{x=1}^{n} \frac{\tau(d_x)_{gw_i,gw_j}{}^{\alpha} \eta_{gw_i,gw_j}{}^{\beta}}{\sum_{y=1}^{m} \tau(d_x)_{gw_i,gw_y}{}^{\alpha} \eta_{gw_i,gw_y}{}^{\beta}} \quad (3)$$

where D is the list of domains of the request with size n. $\tau(d_x)_{gw_i,gw_j}$ is the pheromone value for the link from gw_i to gw_j for the domain d_x that belongs to D. α controls the influence of $\tau(d_x)_{gw_i,gw_j}$ and is greater or equal to 0. η_{gw_i,gw_j} is the cost of sending a message from gw_i to gw_j (i.e., the distance) and is defined as $\eta_{gw_i,gw_j} = \frac{1}{L_{gw_i,gw_j}}$. β controls the influence of η_{gw_i,gw_j} and is greater or equal to 1. $\alpha = 1$ and $\beta = 2$ in this work based on the previous study on the ACO algorithm [22], and to prioritise closer gateways which might imply less hops and lower latency. m is the number of gateways that are relevant for the d_x according to the pheromones in gw_i. If a gateway gw_i receives a response message, it tries to send the response to the consumer c (Line 29). If the consumer does not receive the response, gateway gw_i sends the response to the previous gateway that participates in the discovery process (Lines 30 to 34).

IV. EVALUATION

We use Simonstrator [23] to simulate a network of gateways that covers Dublin city centre (i.e., $2Km^2$ approx.), with static, semi-mobile, and fully-mobile scenarios. All gateways are fixed in the static scenario, 50% of gateways are static and 50% are mobile in the semi-mobile scenario. All gateways are mobile in the fully-mobile scenario. Static gateways are distributed in a grid and mobile gateways follow a social movement pattern provided by Simonstrator, with a speed that varies from $2.7m/s$ (i.e., $10Km/h$ approx) to $13.8m/s$ (i.e., $50Km/h$ approx). The number of gateways varies from 100 to 500, increased by 200 in each experiment. A service provider per gateway, and one consumer are simulated to register and request services. The number of services varies from 20 thousand to 100 thousand, increased by 20 thousand. Each provider sends a registration message until the desired number of services are registered in the network. The consumer does 100 sequential requests to the network. Service descriptions and requests are randomly selected from a data set, which is composed of service descriptions, service requests, and service responses formulated as JSON documents according to our previous work [7]. Each simulation is repeated 10 times and run on the Kelvin system, a high performance compute cluster managed by the Trinity Centre for High Performance Computing (TCHPC). Each node in the cluster has a Linux OS, 12 2.66GHz Intel processors, and 24GB of RAM[1]. We measure the rate of solved requests, simulated response time, number of hops, and number of exchanged messages in each simulation.

We compare our approach against two baselines that are social-based, one baseline that is bio-inspired, and one baseline that combines both a social-based and a bio-inspired mechanisms. These baselines are described as follows:

- Proximity based: It is a social-based approach that forwards requests through the network based on gateways proximity based on the work of Fredj et al. [10]
- Interactions based: This is a social-based approach that manages requests in the network based on gateways interactions based on TSSD. [14]

[1]Kelvin Details - https://www.tchpc.tcd.ie/resources/clusters/kelvin

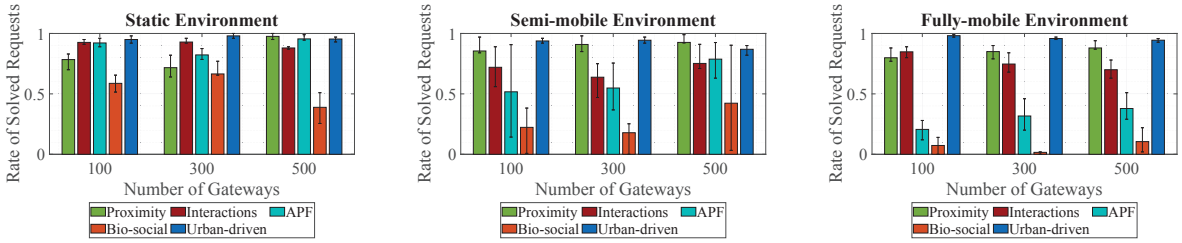

Fig. 1. Median rate of solved requests with a variable number of gateways.

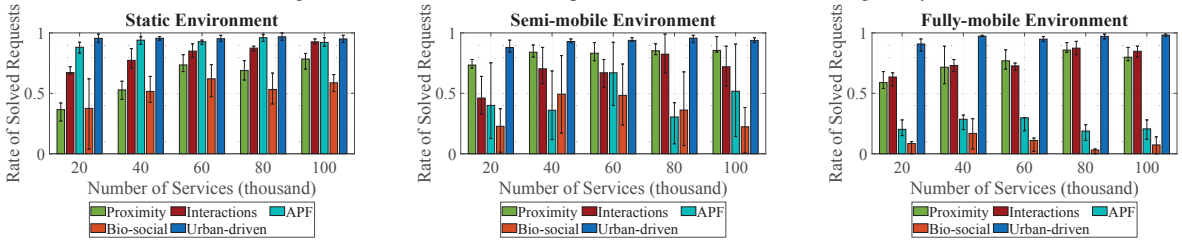

Fig. 2. Median rate of solved requests with a variable number of services.

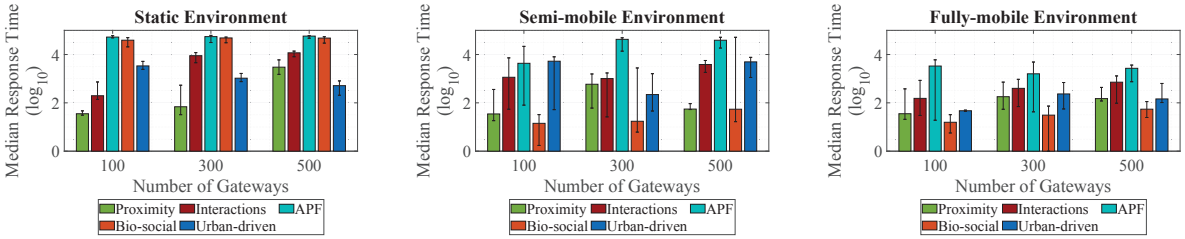

Fig. 3. Median simulated response time with a variable number of gateways.

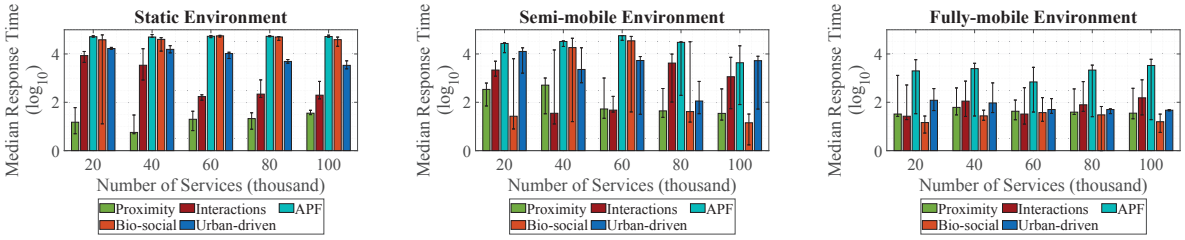

Fig. 4. Median simulated response time with a variable number of services.

- Artificial Potential Fields (APF) based: It is a bio-inspired approach that forwards requests in the network according to attraction and repulsion forces between gateways according to Rapti et al [12].
- Bio-inspired and Social Based (Bio-social): This baseline uses a social-based approach to form communities and a swarm mechanism that reflect changes in the network to manage requests based on Yuan et al. [11]

A. Results

Figure 1 shows the rate of solved requests with a variable network size, different mobility scenarios, and 100 thousand registered services. The rate of solved requests for the urban-driven approach varies from 0.87 with 500 gateways in the semi-mobile environment to 0.98 with 100 gateways in the fully mobile environment. Our approach manages to keep this rate of solved requests despite the mobility scenarios and the network size. This performance demonstrates the effectiveness of adding urban information to the social-based and the bio-inspired mechanisms. The proximity based approach has the second best rate of solved requests, which varies from 0.71 with 300 gateways to 0.96 with 500 gateways in the static scenario. This approach has the best performance with 500 gateways because gateways are closer to each other. The rate of solved requests for the interactions based approach varies from 0.64 with 300 gateways in the semi-mobile environment to 0.93 with 300 gateways in the static scenario. The rate of solved requests for the APF based approach varies from 0.2 with 100 gateways in the fully-mobile environment to 0.95 with 500 gateways in the static environment. Finally, the bio-social based approach has the worst rate of solved requests which varies from 0.016 with 300 gateways in the fully-mobile environment to 0.67 with 300 gateways in the static environment. Figure 2 shows the approaches performance with a variable number of services, different mobility scenarios, and 100 gateways. The proposed approach has the best rate of solved requests (i.e., from 0.88 to 0.98) despite the number of services and the mobility environments. The proximity and interactions based approaches have similar behaviour in

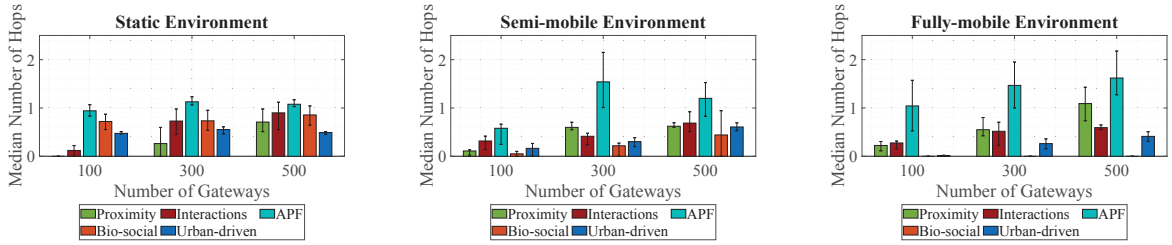

Fig. 5. Median number of hops with a variable number of gateways.

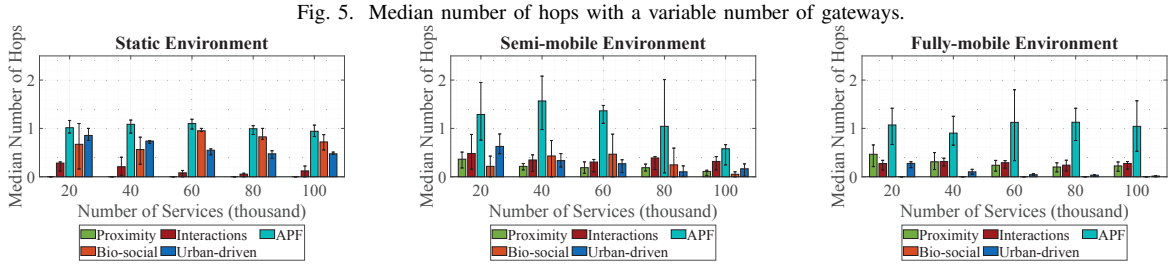

Fig. 6. Median number of hops with a variable number of services.

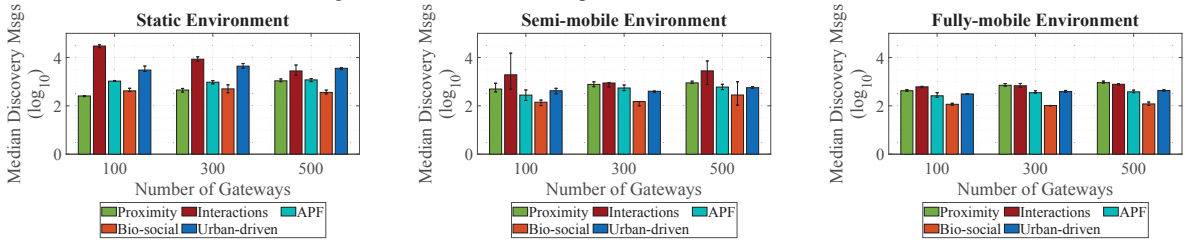

Fig. 7. Median number of discovery messages with a variable number of gateways.

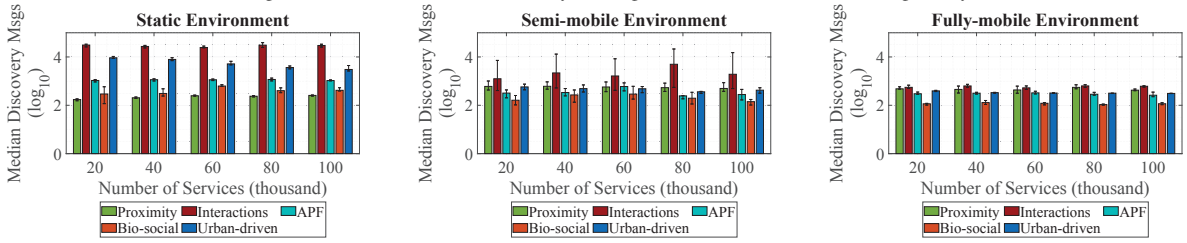

Fig. 8. Median number of discovery messages with a variable number of services.

different mobility environments, where their rate of solved requests increases as the number of services does. However, these rates of solved requests are lower than the rate of our approach (i.e., urban-driven). The APF based approach has a similar performance to the urban-driven approach in the static environment. However, it is negatively impacted by gateways mobility because attraction and repulsion forces get outdated. Finally, the bio-social based approach has the lowest rate of solved requests, which varies from 0.08 with 20 thousand services in the fully-mobile environment to 0.62 with 60 thousand services in the static environment.

Figures 3 and 4 show the simulated response time in logarithmic scale with different number of gateways and services respectively. Latency values are highly related to the median number of hops (Figures 5 and 6) as the response time increases when more gateways are explored to solve a request. All approaches need more time in the static environment than in the mobile ones because there are more information about gateways where to forward requests. The proximity based approach has a low latency in most of the cases, which varies from 35ms to 3s because most of the requests are solved by the gateway that receives them (i.e., number of hops close to 0 in Figures 5 and 6). The interactions based approach has a high latency (i.e., 11s) in the static environment with a larger network (i.e., 500 gateways) when the median number of hops is around one (Figure 5), but it decreases when there are mobile gateways to a minimum of 26ms in a network with 100 gateways and 20 thousand services when the median number of hops is closer to 0. The APF based approach has the highest latency (e.g., around 50s in the static environments) as this approach also uses more hops (i.e., more than one in most cases) than others according to Figures 5 and 6. The bio-social based approach has a high latency in the static scenario (i.e., around 50s) when it reaches the highest rate of solved requests (i.e., 0.66 in Figure 1) and uses more hops (i.e., around 1 in Figure 5). This latency decreases when there are mobile gateways together with the rate of solved requests and the number of hops, which means that the approach fails to find relevant gateways

where to forward requests. The urban-driven approach has an intermediate latency compared against baselines, which varies from 47ms in the fully-mobile environment, where requests are solved where they are received (i.e., number of hops close to 0 in Figures 5 and 6), to 16s in the static environment with 100 gateways where the number of hops is close to 1. The proposed approach exploits the service organisation provided by the social strategy when there are mobile gateways and the bio-inspired approach when there are more information about other gateways in the static environment.

Figure 7 and 8 show the median number of discovery messages that each approach uses to solve requests in a logarithmic scale. Approaches send more messages in static environments because there is more information about available gateways. Similarly, the number of messages increases with the network size. The interactions based approach sends more messages than other approaches. This number varies from around 650 in fully mobile environments to around 30 thousand in static environments. The urban-driven approach sends around 300 discovery messages in fully-mobile environments, around 500 in semi-mobile environments, and more than 3500 in static environments. The APF based approach sends around 350 messages when there are mobile gateways and around 1000 messages in static environments. The proximity based approach sends between 400 and 800 messages in mobile environments and a maximum of 1100 messages in the static environment with 500 gateways. Finally, the bio-social based approach sends around 100 messages in mobile environments and around 500 messages in the static environment.

V. Conclusions and Future work

This paper presents an urban-driven approach that adds urban context (i.e., information about city places) to a social-based and a bio-inspired mechanisms that set up a network of IoT gateways to manage requests in city-scale environments. We compare our model against social-based and bio-inspired approaches with regard to the discovery efficiency and network metrics. Results show that our approach has a better rate of solved requests, and an intermediate latency regardless of the network size, the number of services, and gateways mobility, at the cost of a higher network usage.

VI. Acknowledgments

This work is supported by Science Foundation Ireland under grant 13/IA/1885. Computational resources have been provided by the TCHPC funded by eINIS.

References

[1] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols and Applications," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 1–1, 2015.

[2] D. Lu, D. Huang, A. Walenstein, and D. Medhi, "A secure microservice framework for iot," in *2017 IEEE Symposium on Service-Oriented System Engineering (SOSE)*. IEEE, 2017, pp. 9–18.

[3] M. P. Papazoglou and D. Georgakopoulos, "Service-oriented computing," *Communications of the ACM*, vol. 46, no. 10, pp. 25–28, 2003.

[4] C. Cabrera, F. Li, V. Nallur, A. Palade, M. Razzaque, G. White, and S. Clarke, "Implementing heterogeneous, autonomous, and resilient services in iot: An experience report," in *A World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2017 IEEE 18th International Symposium on*. IEEE, 2017, pp. 1–6.

[5] M. Klusch, P. Kapahnke, S. Schulte, F. Lecue, and A. Bernstein, "Semantic web service search: a brief survey," *KI-Künstliche Intelligenz*, vol. 30, no. 2, pp. 139–147, 2016.

[6] C. Cabrera, A. Palade, and S. Clarke, "An evaluation of service discovery protocols in the internet of things," in *Proceedings of the Symposium on Applied Computing*. ACM, 2017, pp. 469–476.

[7] C. Cabrera, G. White, A. Palade, and S. Clarke, "The right service at the right place: A service model for smart cities," in *Proceedings of Pervasive Computing Conference*. IEEE, 2018, pp. 469–476.

[8] C. Cabrera and S. Clarke, "A self-adaptive service discovery model for smart cities," *IEEE Transactions on Services Computing*, 2019.

[9] N. K. Tran, Q. Z. Sheng, M. A. Babar, and L. Yao, "Searching the web of things: State of the art, challenges, and solutions," *ACM Computing Surveys (CSUR)*, vol. 50, no. 4, p. 55, 2017.

[10] S. B. Fredj, M. Boussard, D. Kofman, and L. Noirie, "Efficient semantic-based iot service discovery mechanism for dynamic environments," in *Personal, Indoor, and Mobile Radio Communication (PIMRC), 2014 IEEE 25th Annual International Symposium on*. IEEE, 2014, pp. 2088–2092.

[11] B. Yuan, L. Liu, and N. Antonopoulos, "Efficient service discovery in decentralized online social networks," *Future Generation Computer Systems*, vol. 86, pp. 775–791, 2018.

[12] E. Rapti, A. Karageorgos, C. Houstis, and E. Houstis, "Decentralized service discovery and selection in internet of things applications based on artificial potential fields," *Service Oriented Computing and Applications*, pp. 1–12, 2016.

[13] M. Girolami, S. Chessa, and A. Caruso, "On service discovery in mobile social networks: Survey and perspectives," *Computer Networks*, vol. 88, pp. 51–71, 2015.

[14] Z. Li, Y. Song, and J. Bi, "Tssd: Exploiting temporal-spatial correlation for service discovery in mobile social networking," in *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, Dec 2017, pp. 1–7.

[15] S. B. Fredj, M. Boussard, D. Kofman, and L. Noirie, "A scalable iot service search based on clustering and aggregation," in *Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing*. IEEE, 2013, pp. 403–410.

[16] M. Girolami, S. Chessa, and E. Ferro, "Discovery of services in smart cities of mobile social users," in *2015 IEEE Symposium on Computers and Communication (ISCC)*. IEEE, 2015, pp. 210–215.

[17] A. Corbellini, D. Godoy, C. Mateos, A. Zunino, and I. Lizarralde, "Mining social web service repositories for social relationships to aid service discovery," in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 2017, pp. 75–79.

[18] H. Xia, C.-q. Hu, F. Xiao, X.-g. Cheng, and Z.-k. Pan, "An efficient social-like semantic-aware service discovery mechanism for large-scale internet of things," *Computer Networks*, 2019.

[19] M. Ebrahimi, E. Shafieibavani, R. K. Wong, and C.-H. Chi, "A new meta-heuristic approach for efficient search in the internet of things," in *2015 IEEE International Conference on Services Computing*. IEEE, 2015, pp. 264–270.

[20] N. Wanigasekara, J. Schmalfuss, D. Carlson, and D. S. Rosenblum, "A bandit approach for intelligent iot service composition across heterogeneous smart spaces," in *Proceedings of the 6th International Conference on the Internet of Things*. ACM, 2016, pp. 121–129.

[21] C. Cabrera, A. Palade, G. White, and S. Clarke, "Services in iot: A service planning model based on consumer feedback," in *International Conference on Service-Oriented Computing*. Springer, 2018, pp. 304–313.

[22] P. Y. Cheong, D. Aggarwal, T. Hanne, and R. Dornberger, "Variation of ant colony optimization parameters for solving the travelling salesman problem," in *2017 IEEE 4th International Conference on Soft Computing & Machine Intelligence (ISCMI)*. IEEE, 2017, pp. 60–65.

[23] B. Richerzhagen, D. Stingl, J. Rückert, and R. Steinmetz, "Simonstrator: Simulation and prototyping platform for distributed mobile applications," in *Proceedings of the 8th International Conference on Simulation Tools and Techniques*, ICST, Brussels, Belgium, Belgium, 2015, pp. 99–108.