Ad Hoc Location Service for Mobile Agents

Ibrahim Lokpo, Tra Goore Bi
Institut National Polytechnique Félix Houphouet-Boigny, BP 1093 Yamoussoukro, Côte d'Ivoire
Email: lokpo@hotmail.com, tgoore@caramail.com

Gérard Padiou
Institut de Recherche en Informatique de Toulouse
ENSEEIHT, BP 7122, 2 rue Charles Camichel, F-31071 Toulouse cedex 7
Email: padiou@enseeiht.fr

Abstract

In this paper, we study two implementations of a mobile agent location service in the context of an underlying ad hoc network. More precisely, this service aims at solving the problem of routing an agent toward a target agent in order to cooperate on a common host. Both the migration number and the tracking time must be minimized by such a service. We compare two implementations based on rumor propagation using either a piggybacking approach or a gossipy agent-based approach. The comparison of the two implementations is based on simulation results. Performance analysis demonstrates that the piggybacking approach minimizes the response time and the gossipy agent-based approach minimizes the network overhead.

1. Introduction

Mobile agents have received a great interest for distributed applications [8, 12] because of their flexibility and capacity of adaptation to very different scenarios. In particular, mobile agents can migrate from machine to machine in a partially disconnected network. A lot of mobile agent platforms [15, 7] have been implemented but they are often experimented in the context of static networks and with few applicative agents.

The mobility of agents can improve the performance of distributed applications. In such applications, when a mobile agent needs a service of an other agent, it moves to the site of the desired service. All interactions becomes local. There is no need of remote procedure call. Thus, the overhead of the network due to such calls are avoided. However, an agent migration cost appears.

Mobile agents are considered as well-adapted to dynamic communication environments such as ad hoc networks thanks to the following interesting properties:

- Communication property: agents do not require permanent connections between nodes: a link must only be available during the migration steps of an agent.
- Fault tolerance property: The loss of an agent can be recovered by using a set of agents. If an agent is lost or locked on its current(disconnected) node, others can continue their task. As long as at least one agent survives, the algorithm still works, albeit with diminished performance.
- Adaptation property: The number of agents can be adjusted according to the applicative load.

In this work, we study a location service for mobile agents in the context of underlying ad hoc networks. In such networks, arbitrary mobile hosts can be recruited to "fill the gap" by serving as intermediate routers between two hosts that may otherwise not be in direct transmission range of one another. More precisely, this service aims at solving the problem of routing an agent toward a target agent in order to cooperate on a common host. Awerbuch and Peleg [1] stated an analogous problem of keeping track of mobile users in a distributed network. Here, users are mobile agents and the goal is different : the requesting agent itself must be routed from its current host to a host where both agents will be eventually co-located.

For the tracker agent, the number of migrations and the time of the routing process must be minimized. We solve this problem in the case where there is possibly never a connected path from the initial tracker agent host to the final meeting host of the two agents or when a network partition exists at the time an agent performs a request to find a target agent. Agent tracking is based upon rumor propagation to route a requesting agent to the target agent host without assuming an underlying location service for nodes.

2. A location service for mobile agents

A location service for mobile agents aims at solving the problem of how to minimize the number of migrations a mobile agent has to perform before reaching a target agent in order to cooperate on a common host. It requires to implement a strategy of adaptative migration. Such a strategy is a mean to route a mobile agent while migrating.

We assume that network mobility is random and agent migration is pro-active. Agents start with no knowledge of either the total number of other agents or the number and the interconnection of active network nodes. In other words, nodes are *anonymous*. We only assume that each applicative agent has a knowledge of the identity of its target agents.

With these weak assumptions, the agent location service cannot be guaranteed with strong fairness properties. Some migration schemes of agents, some race conditions or dynamic network reconfigurations could prevent some requesting agent to reach its target. However, such situations require very specific timing assumptions about the global mobility of agents or nodes. We assume such cases to occur with a very low probability.

2.1. Agent location versus message routing

Agent location exhibits both some similarities and important differences with message routing[2]:

- when an applicative agent requests to meet an other agent on a distant node, it must move toward the requested agent's host. With messages, the message is routed to the receiver and both the sender and the receiver remain free to move anywhere.
- a path from the requesting agent to the target agent must be built in a dynamic way with a very low level of reuse. With messages, routing aims at setting up a path between the sender and the receiver and this path may be reused for a flow of messages.

Several schemes have been proposed for message routing between mobile agents[16]: central server, broadcast and forwarding pointers. These schemes are suitable in mobile agent platforms using a reliable static network but, their basic principles become unappropriate with dynamic networks. The solution of a central server[10, 7] consists in maintaining a connection with a given node. This solution is not appropriate when connections are dynamic. Broadcast consists in flooding many messages through the network. It is not a suitable solution for networks with a low ressource capability. The forwarding pointer approach[5] consists in having on each node visited by an agent, a pointer to the next node where this agent will migrate. This approach is well suited for dynamic networks and is a basic mechanism used to route messages between agents. Therefore, we reuse this notion with, nevertheless, specific update rules.

2.2. Location service specification

The location service is based on a partitioned directory of agents. Each node of the network contains a local directory which only records agents currently located on the node. All consultations and updates of the decentralized directory are local. They do not need remote communication. Therefore, they do not generate any network overhead.

Tracking agent behaviour When an agent arrives at a site, it records its identity in the local directory and looks for the agent it wants to cooperate with. If the target agent is not found in this local directory, then the requesting agent calls a so-called **toward** primitive to obtain a migration destination. This primitive either returns an actual neighbor and the requesting agent attempts to migrate toward this direction or is returns $null$ and, by default, a random migration occurs. The following code expresses such a behaviour :

```
Agent target = lookup(target_agent_name);
if (target == null) {
    Node dest = toward(target_agent_name);
    if (dest != null) move(dest);
    else move(); /* random migration */
}
```

We refine this generic behaviour according to two strategies:

- a blocking strategy in which the **toward** primitive always returns an actual destination neighbor. In this case, this primitive can delay the calling agent until such a neighbor is known;
- a non-blocking strategy in which the primitive immediatly returns an actual destination neighbor or null if no neighbor is available.

Application interface The service exports two primitives:

- **Agent lookup(String target)**: this primitive allows to check if a target agent is currently located on the requesting agent's host. When the primitive fails, it returns a null reference.
- **Node toward(String target)**: this primitive allows to determine a neighbor node toward which the requesting agent must migrate in order to meet the requested agent. According to the service implementation, this primitive can either delay the requesting agent until a neighbor is found (blocking strategy) or return a neighbor or $null$ value immediately (non-blocking strategy).

Location relationships Agent location can be deduced from two kinds of relationship among agents distributed over the network:

- a *visit relationship* $\mathcal{S} \overset{\mathcal{A}}{\hookrightarrow} \mathcal{S}$ in which $\mathcal{S}$ is the domain of site names and $\mathcal{A}$ is the domain of agent names. An occurrence $s \overset{a}{\hookrightarrow} s'$ defines the neighbor site s' which the agent a has moved to when it has left the site s for the last time. A visit relation can be interpreted as a forwarding pointer[5].

- a *neighborhood relationship* $\mathcal{S} \overset{[\![\mathcal{A}]\!]}{\hookleftarrow} \mathcal{S}$ in which $[\![\mathcal{A}]\!]$ is the power set of the set $\mathcal{A}$. For a site s, the relation $s \overset{a_1,a_2,\ldots}{\hookleftarrow} s'$ means: $\{a_1, a_2, ...\}$ is the set of agents assumed by the site s to be located on the neighbor s'.

Neighborhood relations can be captured between neighbor nodes. However, we are interested in propagating these last relations as rumors among nodes to obtain node paths between two agents where ever they are located. For instance, if the following relations are known[1]:

$$s_1\{t\} \overset{a,b,c,d}{\hookleftarrow} s_2\{a,b\} \wedge s_2 \overset{c,d}{\hookleftarrow} s_3\{c,d\}$$

the tracker agent t searching for the agent c, can be routed toward its target along the path $\{s_1; s_2; s_3\}$. To achieve this goal, we adopt an epidemic approach[4] to propagate such rumors. The gossip-based handling of the neighborhood relationship is the main feature of our proposal.

In an ideal world, each node should know which one of its neighbors must be chosen to find the host of any existing distant agent along the shortest path. Such a state cannot exist during an execution in so far as the mobility of sites and agents quickly makes out of date propagated gossips.

3. Service implementation

The implementation of the service aims at providing the most accurate and global knowledge of the preceding relationships to each site. To achieve this goal, each host records a set of hints. A hint $< A, n >$ about an agent A provides the neighbor node n for routing a requesting agent located on the current host a hop away toward its final target agent.

Hints are used to record both relations: a visit relation $s \overset{a}{\hookrightarrow} s'$ is recorded on the node s by a hint $<a, s'>$ and a neighborhood relation $s \overset{a,b,c}{\hookleftarrow} s'$ is recorded by a set of hints $\{<a, s'>, <b, s'>, <c, s'>\}$.

In such a dynamic system, a hint can quickly become out of date. Therefore, a key problem is the validity of hints and a critical issue is the strategy of their updates.

[1]Notation: $s\{x, y, \ldots\}$ means the agents $\{x, y, \ldots\}$ are actually located on the node s

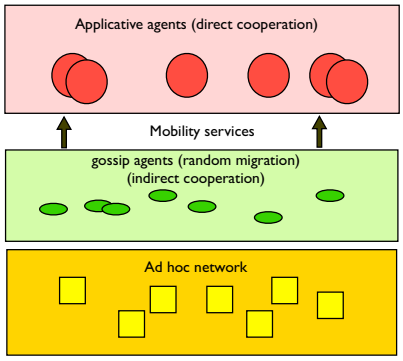

Figure 1. Gossip-based architecture

Classical approaches for message routing use distributed datation mechanisms based upon timestamps[11] or mobility counters[14]. We choose an other approach: since hints have a very low level of reuse, a hint is automatically deleted as soon as an applicative agent has used it. Therefore, we avoid datation handling.

We experiment two different approaches based upon the common principle of rumor propagation used for instance for replicated database maintenance [4] and message routing in ad hoc networks [13].

In the first case, we adopt a hierachical gossip-based approach [3] described in the figure 1:

- a low level contains so-called gossipy agents which move randomly. They perform the rumor propagation about the current hosts of applicative agents.

- an upper level contains applicative agents that can cooperate when they are located on the same host. They use the location service provided by the underlying level. If no hint exists on the current host, the requesting agent is blocked and maintains its outstanding request until a hint becomes available.

By using mobile agents randomly traveling a randomly evolving network, rumors can be propagated through the global system. Such random walks have been proposed as a primary algorithmic principle in protocols addressing searching and topology maintenance of unstructured peer-to-peer networks [9], in group communication protocols in ad-hoc networks[6] and routing in wireless networks[2].

In the second case, we adopt a piggybacking approach: during their migration, applicative agents piggyback gossips about agent location on the visited hosts and perform hint updates. However, in such a case, the location service is partial in so far as agents migrate at random if no hint exists on a host.

In both cases, the visit relationship is handled by applicative agents: when such an agent leaves a node, it creates a

hint containing its identity on its departure node. Such hints can be interpreted as forwarding pointers [5] and can replace hints generated by gossips.

3.1. Gossipy agent-based implementation

This implentation can be stated as follow: a fixed set of gossipy agents performs the rumor propagation about the current hosts of applicative agents. However, the rumor propagation is not based on a flooding approach with a gossiping probability[13]. In our case, the gossipy agents move randomly and try to build paths between applicative agents located on different nodes. They propagate rumors about the current locations of applicative agents. These gossips are used to update the hints recorded on each host.

Hint structure In this implementation, hints have the following structure $< A, n, s, d >$ where:

- A identifies the target applicative agent, used as a key to find a hint about this agent ;
- n provides the neighbor node to route a requesting agent toward A and is obtained thanks to gossip propagation and/or forwarding pointers;
- s specifies which node is the source of the gossip from which the hint is generated. Consequently, this is also the node where the agent A should be located ;
- d evaluates the number of propagation hops of the gossip which has generated this hint.

Gossip list definition A gossipy agent migrates at random and handles a gossip list. Such a list $gl = \{gl_i\}_{i \geq 1}$ contains items $gl_i =< A_i, s_i, d_i >$ where A_i identifies the applicative agent, s_i is the source of the gossip and d_i the number of propagation hops of this gossip.

Algorithm description For each item in a list gl coming from a neighbor $source$, the following algorithm is performed on the current node:

- *Cycle detection step*: if a gossip gl_i comes back to its initial node s_i (this means the gossipy agent has moved along a ring), this gossip is deleted from the list gl.
- *Hint management step*:
  - if a hint $<A_i, n'_i, s'_i, d'_i>$ already exists about the agent A_i, then the new gossip is used to update the existing hint if and only if the path length d_i is shorter than d'_i. In such a case, the hint is assigned the value $< A_i, source, s_i, d_i, >$.
  - if no hint about A_i is available, a new hint $<A_i, source, s_i, d_i>$ is recorded and if applicative agents were currently waiting for such a hint, they are all notified.
- *Gossip list update*: for each applicative agent located on the current host, a new gossip is created and inserted in the gossip list gl of the agent for further propagations. When a gossipy agent leaves a node, it carries this final updated list.

3.2. Piggybacking-based implementation

All mobile agents in the system are involved in the location service. In this implementation, rumors are propagated by applicative mobile agents themselves. An applicative agent is never blocked during its migration steps (non-blocking strategy). If an agent invokes the **toward** primitive and it does not obtain an available hint, it then migrates at random. In this implementation, an agent can be blocked if and only if its host is partially disconnected.

This implementation only requires basic hints. A hint is a pair $<agent, neighbor>$. The applicative agents piggyback the same gossip lists as the gossipy agents. However, their use partially differs. The first steps are similar: before leaving a node, agents located on the current node are inserted in the gossip list and, on the destination node, an agent coming from a node $source$, uses its current piggybacked gossip list to update the local hints.

For each item in a gossip list gl, the following algorithm is performed:

- *Cycle detection step*: if a gossip gl_i comes back to its initial node s_i (this means the gossipy agent has moved along a ring), this gossip is deleted from the list gl.
- *Hint management step*: if a hint $< A_i, n'_i >$ already exists about the agent A_i, then the new gossip is used to update the existing hint else a new hint is created.
- *Gossip list update*: the gossips with a distance d greater than a maximum fixed hop number are removed (propagation over a limited neighborhood).

4. Performance Analysis

We present experimentation results obtained with a simulator we have developped in Java. The simulation aims at evaluating global performance of the proposed location service by comparing the two implementations. It should be noted that the performance of a mobile agent-based application is mainly determinated by the cost due to the number of migrations.

The simulation platform implemented in Java, has the three following main modules : a supervision module in

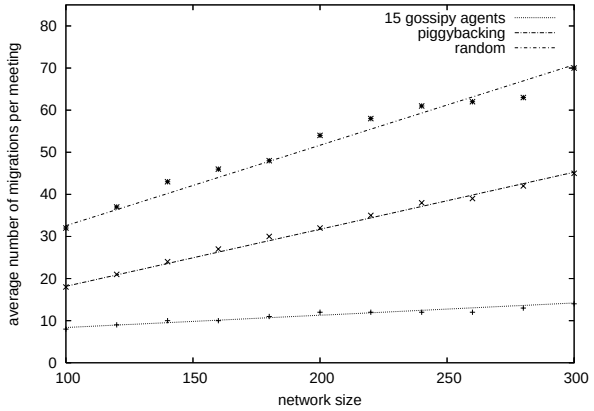

Figure 2. Network overhead comparison

charge of checking the simulation duration and mobile agents activity, a mobility management module in charge of making site neighborhood updates (it is assumed in this simulation that a node has an average of 4 one-hop neighbors) and a mobile agent management module in charge of the agent life cycle including their migrations.

We assume the agents are reliable and the nodes are infinitely often connected. A simulation starts with agents located at random on nodes. An applicative agent repeats the following behavior until the end of the simulation:

- first, it chooses a target agent at random to cooperate with,
- then, it migrates toward the host of the target agent using the location service,
- finally, when it arrives at the site of its target, it cooperates during a bounded service time with its target.

The principle of our simulation is to count, on the one hand the total number of migrations (hops) $\sigma_h(t)$ done by the applicative mobile agents and on the other hand the total number of meetings $\sigma_m(t)$ performed by these agents in order to calculate the average number $A(t)$:

$$A(t) = \frac{\sigma_h(t)}{\sigma_m(t)}$$

This number represents the average number of migrations performed by a mobile agent before meeting a target ($\equiv$ the tracking path length) and indicates the efficiency of the location service.

The gossipy agent-based and piggybacking implementations are compared to each other and compared to the random migration scheme according the following criteria:

- network overhead (average number of migrations per meeting)
- meeting rate (average time needed to realize a meeting)
- stability (variation of average numbers according to the network size)

We fix the average number of neighbors of each node (namely 4) and we consider the following parameters: number of nodes, number of applicative agents, number of gossipy agents. The number of nodes (size of the network) is in the range $[100 : 300]$. The experimentation was done with 100 applicative agents and 15 gossipy agents. The mobility of gossipy agents is 1000 times faster than the mobility of nodes.

In figure 2, results show that these two implementations are better than the random migration scheme with respect to the network overhead. The piggybacking approach reduces this number about 2 times. The use of gossipy agents reduces 5 times the average number of migrations.

With gossipy agents, the average number of migrations per meeting increases very slowly according to the number of gossipy agents. For instance, with a network size in the range $[100 : 300]$, this average number remains almost constant if the actual number of gossipy agents is beyond a threshold of fifteen agents. Moreover, the average number of migrations per meeting also appears closed to the network diameter.

With the piggybacking solution, gossips are only propagated over a limited neighborhood, namely less than five hops. Greater values do not provide any significative improvement of the service performance.

Figure 3 shows the variation of the number of meetings during one-hour simulations. This parameter quantifies the performance of the service with respect to the same applicative agent load. We can deduce from this value a mean traking time: if n meetings occur in an hour, the mean tracking time in seconds is $3600/n$. The piggybacking approach gives the best results (the greatest number of meetings during a simulation). The following table compares the mean tracking time to satisfy an outstanding meeting request to the network overhead with 100 nodes.

implementation	tracking time	network overhead
no service	1.56 sec.	56 hops
piggybacking	1 sec.	35 hops
gossipy agents	7.80 sec.	12 hops

The gossipy agent-based location service minimizes the network overhead but increases the average response time and the piggybacking approach minimizes the response time but increases the network overhead. According to the application requirements, the most efficient implementation can be chosen. Moreover, with gossipy agents and

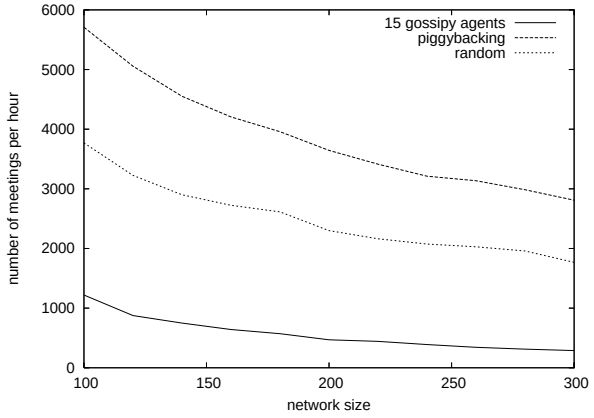

Figure 3. Meeting rate comparison

a blocking strategy, applicative agents can also perform rumor propagation in the same way as gossipy agents. Their participation decreases the mean tracking time but this improvement strongly depends on the number of applicative agents.

5. Conclusion

Throughout this study, we have proposed two different implementations of a fault-tolerant ad hoc location service. Our gossipy agent-based approach avoids message flooding generally used for rumor propagation. Moreover, the hint update does not require a datation mechanism.

The simulation has shown that this location service improves the performance of the application by actually decreasing the network overhead. The average number of migrations is reduced around twice with the piggybacking implementation. In the case of the gossipy agent-based scheme, this average number is reduced about five times.

As a future work, we have to evaluate the impact of gossipy agents with respect to the network overhead. We want also implement a slightly different version of the gossipy agent-based solution. In this version, a deadline will be assigned to the waiting time for a hint. If a deadline occurs, by default, the requesting agent will perform a migration at random. We have to verify whether a higher meeting rate will be obtained with such a version without a too expensive augmentation of network overhead.

Acknowledgments The authors would like to thank the referees for their useful remarks and encouraging comments.

References

[1] B. Awerbuch and D. Peleg. Online tracking of mobile users. In *Proceedings of the ACM SIGCOMM Symposium on Communication Architectures and Protocols*, 1991.

[2] M. Bui, S. K. Das, A. K. Datta, and D. T. Nguyen. Randomized mobile agent based routing in wireless networks. *International Journal of Foundations of Computer Science*, 12(3):365–384, 2001.

[3] C. Cubat dit Cros. Agents mobiles coopératifs pour les environnements dynamiques. In *Les Nouvelles Technologies de la Répartition NOTERE'2004*, Saidia, Maroc, juin 2004.

[4] A. Demers et al. Epidemic algorithms for replicated database maintenance. In *6th Symposium on Principles of Distributed Computing*, pages 1–12, Aug. 1987.

[5] J. Desbiens, F. Renaud, and M. Lavoie. Communication and tracking infrastructure of a mobile agent system. In *Thirty-First Annual Hawaii International Conference on System Sciences*, volume 7, pages 54–63, January 1998.

[6] S. Dolev, E. Schiller, and J. Welsh. Random walk for self-stabilizing group communication in ad hoc networks. In *Proceedings of the twenty-first annual symposium on Principles of distributed computing (PODC'02)*, pages 259–259, New York, NY, USA, 2002. ACM Press.

[7] P. Domel. Mobile Telescript agents and the Web. In *Digest of Papers. COMPCON '96. Technologies for the Information Superhighway. Forty-First IEEE Computer Society International Conference*, pages 52–57. IEEE Computer Society Press, February 1996.

[8] A. Fuggetta, G. P. Picco, and G. Vigna. Understanding Code Mobility. *IEEE Transactions on Software Engineering*, 24(5):342–361, May 1998.

[9] C. Gkantsidis, M. Mihail, and A. Saberi. Random walk in peer-to-peer networks. In *INFOCOM, The Conference on Computer Communications*. IEEE, 2004.

[10] R. S. Gray, D. Kotz, G. Cybenko, and D. Rus. D'agents: Security in a multiple-language, mobile-agent system. *LNCS*, 1419:154–187, 1998.

[11] JinHoAhn. Decentralized inter-agent message forwarding protocols for mobile agent systems. In *Computational Science and Its Applications ICCSA 2004: International Conference*, volume 3045, pages 376–385. LNCS - Springer-Verlag, May 2004.

[12] D. B. Lange and M. Oshima. *Programming and Deploying Java™Mobile Agents with Aglets™*. Addison-Wesley, 1998.

[13] L. Li, J. Halpern, and Z. Haas. Gossip-based ad hoc routing. In *INFOCOM, The Conference on Computer Communications*. IEEE, June 23-27 2002.

[14] L. Moreau. Distributed directory service and message routing for mobile agents. *Science of Computer Programming*, 39(2-3):249–272, March 2001.

[15] A. Ohsuga, Y. Nagai, Y. Irie, M. Hattori, and S. Honiden. PLANGENT: an approach to making mobile agents intelligent. *IEEE Internet Computing*, 1(4):50–57, July/Aug. 1997.

[16] P. T. Wojciechowski. Algorithms for location-independent communication between mobile agents. In *Proceedings of Artificial Intelligence and the Simulation of Behaviour (AISB '01) Symposium on Software Mobility and Adaptive Behaviour*, march 2001.