

## Decentralised Discovery of Mobile Objects

RICHARD GLASSEY,<sup>†</sup> GRAEME STEVENSON<sup>††</sup>  
and ROBERT IAN FERGUSON<sup>†</sup>

The partially connected nature of mobile and ubiquitous computing environments presents software developers with hard challenges. Mobile code has been suggested as a natural fit for simplifying software development for these environments. However, existing strategies for discovering mobile code assume an underlying fixed, stable network. An alternative approach is required for mobile environments, where network size may be unknown and reliability cannot be guaranteed. This paper introduces AMOS, a mobile object platform augmented with a structured overlay network that provides a fully decentralised approach to the discovery of mobile objects. We demonstrate how this technique has better reliability and scalability properties than existing strategies, with minimal communication overhead. Building upon this novel discovery strategy, we show how load balancing of mobile objects in an AMOS network can be achieved through probabilistic means.

### 1. Introduction

Mobile and ubiquitous devices introduce difficult challenges to the domain of distributed computing. These include: managing intermittent network connectivity; making optimal use of a device's limited resources; and mitigating determinable failures, such as loss of battery power. Developing software for these unpredictable environments is therefore much harder than for traditional distributed systems.

Mobile code<sup>1)\*</sup>, capable of migrating between distributed devices at runtime, has been suggested as a natural fit for managing some of these challenges<sup>2)</sup>. Disconnected operation can be facilitated by migrating a process from a device with weak connectivity into a network of stable hosts to complete its task. Intensive processes can be migrated from a resource constrained device to another that has better resources available. Loss of power can be pre-empted by migrating a process to another device to continue its operation. These compelling reasons suggest that mobile code has something to offer in the domain of mobile and ubiquitous computing.

As mobile code migrates, knowledge of its network location is required before communication can take place. There are several established strategies for discovering mobile code location: centralised registry; multicast discovery; and home server. However, each has lim-

itations that render them neither reliable nor scalable in the face of unpredictable environments.

This paper describes AMOS (Adaptive Mobile Object System), which provides improved location discovery of mobile objects<sup>☆☆</sup>, in terms of reliability and scalability, than existing strategies, with minimal communication overhead. We introduce the Host Routing strategy, which uses an overlay network to discover the location of mobile objects. Overlay networks, such as CAN<sup>3)</sup>, Chord<sup>4)</sup>, and Pastry<sup>5)</sup>, are distributed message-routing platforms that do not rely on centralised control or hierarchical organisation<sup>6)</sup>. They are typically self-organising networks, layered upon an IP network, that use a flat, logical addressing scheme. With knowledge of only  $O(\log n)$  hosts (in a network of  $n$  nodes), each host can send a message to any point in the network by using intermediate hosts to route the message. Messages only need to be forwarded via  $O(\log n)$  hosts to reach their destination. This global yet distributed index of hosts is scalable, efficient, and reliable.

In AMOS, each mobile object has a globally unique identifier (GUID) from which a unique network identifier (UNID) in the overlay network address space is generated. After each migration, a mobile object instigates a registration process that routes its new IP address to the host with the numerically closest UNID

<sup>\*</sup> At an abstract level, mobile code is to software, as mobile IP is to devices.

<sup>☆☆</sup> We use mobile object rather than mobile code, process, or agent.

<sup>†</sup> University of Strathclyde, UK

<sup>††</sup> University College Dublin, Ireland

to its own. This host assumes responsibility for storing the mobile object’s location whilst it remains active. Should the host fail or leave the network, the host with the next closest UNID assumes responsibility automatically.

The Host Routing discovery strategy exploits this relationship between hosts and mobile objects to locate mobile objects in the network. A location-request message is routed to the host with the UNID closest to the mobile object’s UNID. This host is guaranteed to be the host responsible for storing the current IP address for this mobile object. On receipt, a response is sent containing the current IP address of the mobile object in question. This address can then be used to contact the mobile object.

As this approach has no single point of failure it improves reliability compared to the centralised registry and home server approaches; reduces communication overhead compared to a multicast based solution due to efficient routing; and achieves scalability by distributing the management effort throughout the system.

The remainder of the paper is structured as follows. Section 2 outlines three existing strategies for discovering the location of mobile objects and identifies their limitations in unpredictable environments. Section 3 describes the architecture of AMOS, an adaptive mobile object system developed to improve the discovery of mobile objects in mobile and ubiquitous environments. Section 4 presents the results of the evaluation of the registration process and host routing strategy. A decentralised load balancing application that makes the best use of a heterogeneous host environment is also evaluated to illustrate the benefits of using AMOS. Section 5 provides a brief survey of related work. Finally, in Section 6, we present our conclusions.

## 2. Location Discovery

It is essential to know the current location of a mobile object in the network in order to communicate with it. This section looks at three strategies – centralised registry, multicast and home server – and discusses their advantages and disadvantages.

### 2.1 Centralised Registry

One strategy for tracking mobile objects is to maintain a fixed host for storing their location after each migration. A centralised registry strategy is simple to implement, requiring only that mobile objects report their location to

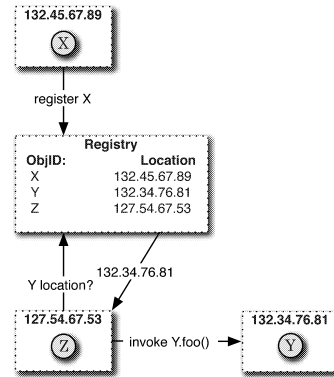


Fig. 1 Registration and look-up in a centralised location discovery strategy.

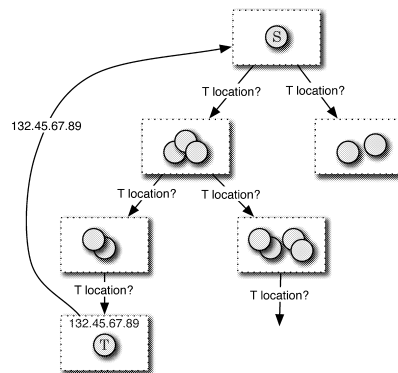


Fig. 2 Location discovery using a multicast strategy.

a well known host. Aglets<sup>7)</sup> and Concordia<sup>8)</sup> are two mobile code platforms that make use of this strategy. **Figure 1** shows a general registry service, where dotted rectangles represent host machines and grey circles represent mobile objects. Although efficient, this is not a robust solution as a network may consist entirely of unreliable hosts. This strategy creates a single point of failure and introduces the potential for performance bottlenecks.

### 2.2 Multicast

A multicast strategy propagates a discovery request throughout a network without relying on any fixed hosts. Emerald<sup>9)</sup> is a mobile code framework that uses a multicast strategy for mobile object discovery. **Figure 2** shows a typical discovery process where S requests the location of target T. Although multicast is more reliable than a centralised registry, it still is costly because the sender floods the network until the target is discovered. For resource-constrained devices, network communication is a costly activity and should be minimised.

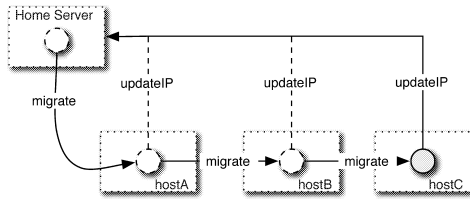


Fig. 3 Registration of IP with home server.

### 2.3 Home Server

The home server strategy distributes the mobile object registry throughout a network in order to avoid a single point of failure. AgentSpace, the mobile code framework that underlies AMOS, makes use of the home server strategy<sup>10</sup>). It operates on the principle that each host launching a mobile object becomes responsible for tracking its location throughout its life-cycle (illustrated in Fig. 3).

A mobile object's globally unique identifier (GUID) includes the IP address of its home server. After each migration, a mobile object contacts its home server and notifies it of its new location. To locate a mobile object, a process must contact its home server. This is achieved by extracting the home server's IP address from the mobile object's GUID. A mobile object cannot update its location if its home server fails, preventing further communication. Furthermore, if one host launches many mobile objects, it may become a bottleneck. Although not completely robust, this improves upon the previous two strategies by removing the single point of failure (replacing it with multiple, and therefore less critical, points) and minimising communication costs.

### 3. Architecture

Whilst mobile objects have properties that make them ideal for mobile and ubiquitous environments, if they become unreachable then their benefits are lost. This section outlines the high level architecture of AMOS; describes how the reliable location registration and discovery of mobile objects is achieved using key-based routing; and illustrates how a load balancing application built on AMOS can autonomously achieve fair distribution of mobile objects in a network of heterogeneous hosts.

#### 3.1 Architectural Overview

Mobile code frameworks depend upon a host running a container process that can create, host and receive mobile objects. The container provides the execution environment in which

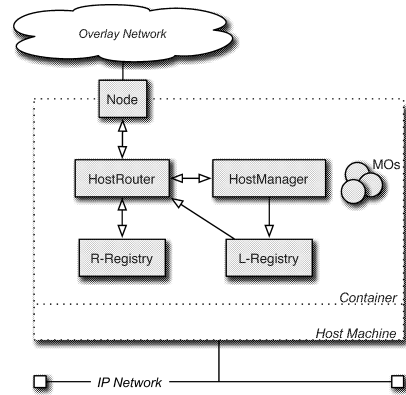


Fig. 4 High-level architecture of AMOS on a single host.

mobile objects can carry out the tasks they have been set to complete. It also acts as a sandbox that can enforce access control policies to prevent malicious mobile objects from damaging the host.

In AMOS, a network of hosts running container processes forms a structured overlay network<sup>6</sup>), using the protocols developed within the Pastry project. Within the overlay network, no single host has a global view of the entire network. Instead, each host is responsible for maintaining a partial view; in effect becoming part of a distributed hash table. This set is of size  $O(\log n)$  where  $n$  is the number of hosts in the network.

Hosts communicate indirectly by sending messages using key-based routing. Messages are deterministically routed through the overlay network address space using a greedy approach. A host forwards a message to a member of its partial view of the network (routing set) that has the closest network identifier to the destination network identifier. This process repeats until the host matching the network identifier receives the message. The performance of key-based routing is  $O(\log n)$  where  $n$  is the number of hosts in the network.

Figure 4 shows a high-level view of AMOS on a single host, where a HostManager and its components are running inside a container process aside mobile objects. When a new HostManager component is created, it uses a well known IP address (or range of addresses) to bootstrap itself into an existing overlay. This involves generating a unique network identifier (UNID) and setting up its partial view of the network. The process is described in Ref. 5).

The L-Registry (local registry) keeps a reg-

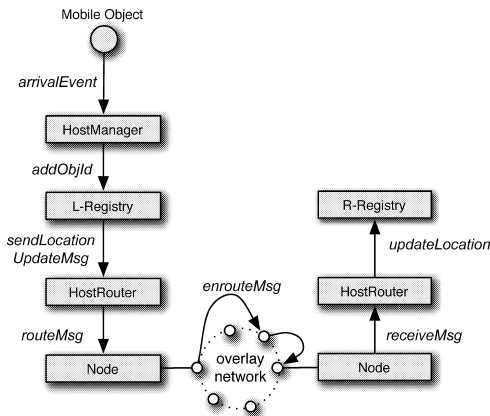


Fig. 5 Registration of a mobile object in AMOS.

ister of locally hosted mobile objects and notifies the HostRouter when mobile objects arrive. The R-Registry (remote registry) stores the IP location for each mobile object that the host is currently responsible for. The next section describes how these components interact to handle the registration and location discovery of mobile objects.

### 3.2 Location Discovery

The limitations of existing location discovery strategies yield a set of three requirements: avoid single points of failure; increase reliability through fault tolerance; and minimize the required communication overhead, irrespective of network size. This section details how AMOS satisfies these requirements during the mobile object registration and discovery processes.

#### 3.2.1 Mobile Object Registration

A registration service should provide efficient, reliable and timely look-up of a mobile object's location. AMOS registration takes the novel approach of registering the mobile object at a remote host within the overlay network. This host is determined using the mobile object's GUID as the unique key for a specific UNID in the network address space. For cases where a host with the generated UNID does not exist the host with the numerically closest UNID assumes responsibility<sup>5)</sup>.

Figure 5 shows the AMOS component interaction during the registration process. The arrival of a mobile object generates an arrival event which the HostManager detects. The HostManager then adds the GUID of the mobile object to its local registry (L-Registry). This causes the local registry to notify any observers that a new mobile object has arrived. The HostRouter component is one such ob-

server. On notification, it creates a new location update message containing the mobile object's GUID and the host's IP address. This message is dispatched to the Node, which forwards the message to the next Node within its routing table possessing the numerically closest UNID to that generated from the mobile object's GUID. Once the message arrives at its destination, it is added to the HostManager's remote registry (R-Registry).

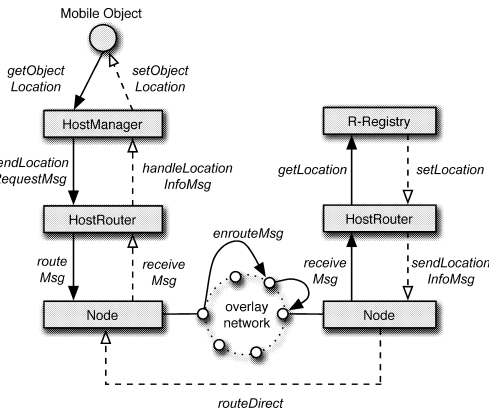
This re-occurs each time a mobile object migrates from one host to another. The chief benefit of this indirect registration is that if the host which has the UNID closest to the mobile object's generated UNID leaves the network or fails, the next active host with the closest UNID will automatically assume responsibility for registration of the mobile object. The process is transparent to the mobile object and the HostManager attempting to register it, as the overlay network manages the UNID resolution process.

If the host managing a mobile object's location fails and the mobile object does not migrate, any attempts to discover the location will fail. However, each host periodically checks its partial view of hosts in the overlay for failures. AMOS leverages this by ensuring that each node replicates their remote registries of the other nodes in their set. If a host failure is detected, the registration process can be invoked for all the mobile objects belonging to the failed host – repairing the registry. This approach increases the reliability of registering mobile objects and occurs transparently. By taking advantage of the self-organising structure of the overlay network we avoid less efficient strategies, such as forcing mobile objects to re-register periodically, or requiring HostManagers to monitor mobile objects that have not recently moved.

#### 3.2.2 Discovery of Location

Compared with multicast, Host Routing significantly limits network overhead by using the distributed index of mobile objects to achieve reliable look-up. Host Routing also avoids using well known hosts to manage the location information of mobile objects. As with other strategies, the caveat of Host Routing is that the requester must already possess the GUID of the target mobile object. Providing the GUID location information is known, discovery of the mobile object is guaranteed.

The discovery process begins when the re-



**Fig. 6** Host Routing strategy for discovering the location of a mobile object.

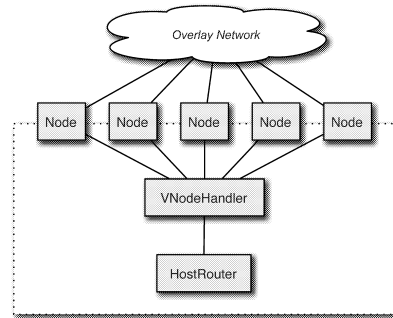
requester contacts its local HostManager and issues a request for the location of a mobile object. The HostManager delegates the task to the HostRouter. A location request message is generated and passed to the Node, which routes it to the host responsible for registering the location of the target mobile object. The receiving HostRouter performs a look-up within its remote registry using the target mobile object’s GUID as the key, which returns the mobile object’s last known IP address. The HostRouter then routes this information directly back to the host that issued the request. Once the HostManager receives the response, it calls back the requester and informs it of the target mobile object’s location.

This process is illustrated in **Fig. 6**, where solid black arrows indicate the first phase of sending the location request message, and the dotted arrows indicate the location information message generated in response.

As the paths that messages are routed across to reach their destination are not fixed, the overlay network can sustain damage whilst maintaining its ability to route messages correctly. From a developer’s perspective, the process of locating a mobile object using the Host Routing strategy occurs transparently. As with other strategies described in Section 2, the location of a mobile object is obtained by invoking a look-up process that takes the mobile object’s GUID as an argument. Thus, use of the Host Routing strategy has little impact on the complexity of application development.

### 3.3 Distribution of Effort

Load balancing is particularly important for mobile and ubiquitous environments, where the number and location of capable hosts within a



**Fig. 7** Multiple nodes per single host.

network cannot be pre-determined. It is impractical to maintain an accurate index of capable hosts, and therefore difficult to distribute load fairly according to capability. It is important that such networks can approach a balanced state without global knowledge.

To achieve load balancing in AMOS, we modify the architecture in two ways: allowing hosts to occupy variable amounts of network address space; and by modifying mobile objects to prefer more capable hosts. More capable hosts spawn multiple Nodes – in effect, making themselves more ‘visible’ within the overlay network.

**Figure 7** demonstrates how this is achieved. The *VirtualNodeHandler* acts like a multiplexer, randomly choosing a single node to forward an outgoing message whilst passing messages from all nodes back to the HostRouter. There is a higher probability that a mobile object migrating to a randomly chosen point in the network will arrive at a host with a greater number of Nodes than another. A host with more Nodes will handle more registration requests because of its increased ‘visibility’.

To mitigate mobile objects arriving at a resource constrained host, they are modified to prefer more capable hosts. This is achieved through ‘host introspection’; querying how ‘busy’ the host currently is via the HostManager. A HostManager defines its load by dividing a host capability metric by the number of active mobile objects it hosts. Capability can be customised (e.g., using a device profiling standard such as CC/PP<sup>11</sup>), but, for simplicity, we define it as a numerical value. A resource-constrained device has a low number, whilst a fixed server has a high number.

On arrival at a new host, a mobile object uses host introspection to determine if the host is suitable for executing its tasks. If not, it migrates to another host. A host is selected by

generating a random UNID and requesting the IP of the host with the closest UNID. When applied by a population of mobile object, random migration combined with host introspection leads to a balanced, non-deterministic distribution of load.

#### 4. Evaluation

This section details three experiments that were carried out in the evaluation of AMOS. The first investigates the impact of increasing network size on the registration costs in a population of mobile objects. The second measures the cost, in time, of AMOS' mobile object look-up strategy in comparison with the home server strategy. Finally, the third experiment assesses the efficacy of our approach to load balancing through decentralised control and probabilistic methods. The following experiments were conducted on a network of twenty Sun Blade workstations, running Solaris 2.5.1, connected via 100-BaseT ethernet. Larger networks were simulated using virtual nodes (see Fig. 7).

##### 4.1 Registration

The first experiment measures the cost, in terms of time, incurred by the registration process, across networks of different sizes. The source code of the HostManager was altered to measure the time taken to register a mobile object. To avoid significant clock discrepancies, a Network Time Protocol server was used to synchronise all hosts. A network of  $n$  nodes was constructed, and for each size of network a single mobile object was launched, invoking the registration process. This was repeated 10K times for each size of network. The results are shown in Fig. 8.

The median value is shown as a thick horizontal line. The box represents the range of registration costs falling within the 25th and 75th percentiles, and the 'whiskers' represent the 5th and 95th percentiles. As the network increases in size, the average cost to register a mobile object remains within the range of 25–40 ms. The narrow ranges suggest that for each network size, registration cost remains consistent.

##### 4.2 Discovery

The second experiment compares the time taken to discover the location of a mobile object using the Host Routing and home server strategies. We measure the total time taken to discover the location of and invoke a method on a remote mobile object.

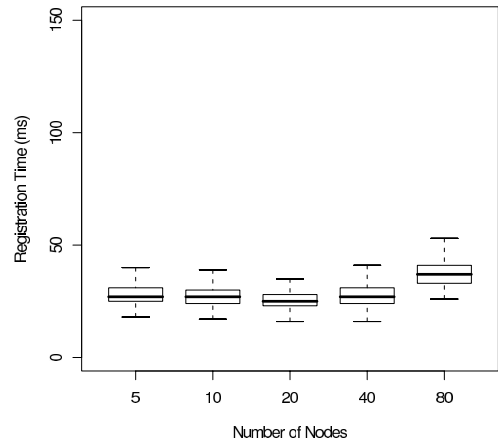


Fig. 8 Cost incurred, in time (ms), for registering a mobile object in network of increasing size.

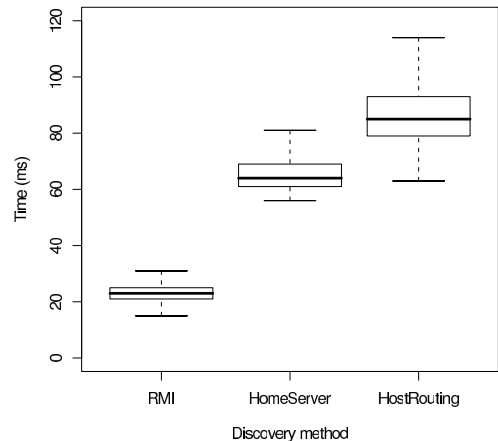


Fig. 9 Completion times for RMI look-up, home server and Host Routing strategies.

Two mobile objects, A and B are launched into the network of hosts. B migrates to a randomly selected location in the network and waits. A then attempts to discover where B is, using each of the discovery strategies. As a benchmark for comparison, A uses Java RMI with the known IP address of B to invoke a method. This measurement indicates the network cost without any discovery process. A then uses the home server discovery strategy to invoke a method on B. Finally, A uses the Host Routing strategy to discover the location of B. Each approach is repeated for 10K iterations, mitigating the effects of any host or network anomalies. Figure 9 summarises the results.

Host Routing costs more than the home server strategy, with mean completion times of 91.5 ms and 67.4 ms respectively, and displays a

**Table 1** Distribution of mobile object population after three decisions.

Decision	Host Type				
	A	B	C	D	E
0	500	0	0	0	0
1	11	48	100	160	181
2	8	48	99	155	190
3	8	50	94	152	196
<i>Optimal</i>	<i>24</i>	<i>47</i>	<i>95</i>	<i>144</i>	<i>190</i>

wider distribution of results. We believe that an additional cost of  $\approx 25$  ms is reasonable given the reliability and self-healing properties of the Host Routing strategy.

### 4.3 Load Balancing

This experiment demonstrates how AMOS's load balancing application can distribute a population of mobile objects across a network of hosts. We use the host capability measure from Section 3.3 to vary the number of nodes per host. For the purpose of this experiment, we simulate a range of host types with various capabilities. Type A is resource constrained and only deploys one Node, whereas Type E is more capable and deploys eight Nodes. We build a network using five hosts of each type.

To create a 'worst case' load scenario, we populate one Type A device with five hundred mobile objects. Each mobile object waits a random period of time, uses host introspection, and decides whether to stay or migrate.

**Table 1** illustrates how the population of mobile objects distributed themselves after three decisions. It is clear that after one round of decisions the distribution is already approaching optimal. The optimal proportion of hosted mobile objects for each device type is calculated by  $P = M(\frac{D \times F}{N})$ , where  $M$  is the total number of mobile objects,  $D$  is the number of devices of this type,  $F$  is the number of nodes deployed, and  $N$  is the total number of nodes.

Although subsequent runs yielded similar distributions, it is not clear if significant mobile object oscillation is occurring. Whilst this example is simulated, it illustrates how the design of AMOS can be easily extended to achieve decentralised control, in this case balancing the load proportionally throughout a network of hosts.

### 4.4 Summary

This section detailed the evaluation of AMOS. We demonstrated that: the cost incurred by the registration process does not vary significantly as network size increases; the Host Routing strategy is competitive with the home

server strategy; and that autonomous load balancing can be achieved through decentralised control and probabilistic methods.

## 5. Related Work

This section gives a brief overview of existing mobile code frameworks and peer-to-peer (P2P) systems. We also discuss an earlier project with the goal of combining these technologies to provide reliable agent communication, and compare its performance with AMOS.

The majority of existing mobile object platforms make use of the location strategies described in Section 2. Aglets<sup>7)</sup> and Concordia<sup>8)</sup> implement a centralised registry, whilst AgentSpace<sup>10)</sup> and Adjanta<sup>12)</sup> use the home server strategy. In Emerald<sup>9)</sup>, discovery is supported by following forwarding-pointers left by migrating objects or by using a multicast strategy.

P2P overlays are useful for managing a population of hosts in a distributed system. Such isolation allows service participants to see only each other, and allows properties of the overlay to be fine tuned to meet overlay specific needs (e.g., resilience or locality). Overlays also provide population management features (self-organisation and self-repair) that are essential for the management of ad-hoc networks. This project is interested in the capability of P2P overlays to provide object location, messaging, and node organisation in ad-hoc networks.

AMOS is built on top of Pastry<sup>5)</sup>. Pastry supports application level message routing and object location in a large overlay network of nodes. When given a message and a node ID (key), Pastry efficiently routes the message to the node with the ID that is numerically closest to the key among all live pastry nodes. The expected number of routing steps is  $O(\log n)$  where  $n$  is the number of nodes in the network.

Other P2P overlays such as CAN<sup>3)</sup> and Chord<sup>4)</sup> provide hash table like functionality. Chord maps keys to nodes and provides an  $O(\log n)$  routing-hop lookup algorithm that can be used to find the IP address of the node responsible for any key. CAN maps a virtual d-dimensional Cartesian coordinate space across all participant nodes, with each node responsible for all points within a zone. The number of routing-hops in the CAN lookup algorithm grows faster than  $O(\log n)$ .

Although mobile objects have long been suggested as a promising technology for use in ad-

hoc networks, existing discovery strategies have lacked the robustness required for such environments. In this section we compare AMOS to another system which has tried to address this problem through use of a P2P overlay, Armada<sup>13)</sup>.

Armada is a mobile object communication system built on top of Tornado<sup>14)</sup>. In Armada, a community of mobile objects distributed across a set of hosts autonomously manage their communication. Each mobile object keeps a list of other mobile objects that it knows how to send messages to. If a mobile object wishes to send a message to a node that it does not know how to contact directly, it looks up the mobile object whose key is numerically closest to, but lower than the key of the destination host. This receiving host then repeats this process until the message reaches its destination. Although, multiple communities may co-exist on the same set of hosts, only mobile objects in the same community are used to aid routing.

On joining a community, a mobile object constructs a set of pointers to other mobile objects in the community. Mobile objects periodically perform a secondary algorithm to ensure that they maintain pointers to the closest neighbouring set. On migration, a mobile object updates its location in the community by finding the host numerically closest to its own. A mobile object on that host receives the updated location and joins a sub-set of mobile objects responsible for storing the location of mobile objects in that community.

From the results in Ref.13), Armada performs best when the size of a community of mobile objects is small. The number of messages required for a mobile object to join a community is  $O(\log a)$ , where  $a$  is the size of the mobile object community. This can be compared to  $O(\log n)$  messages for an AMOS mobile object to register, where  $n$  is the number of hosts in the network. Armada incurs a cost of  $O(\log a) + O(\log a)^2$  messages every time a mobile object migrates to update its community. In AMOS, mobile object migration requires  $O(\log n)$  messages to be sent. Finally, sending a message to an Armada mobile object requires  $2 \times O(\log a)$  messages to be sent, whilst in AMOS this figure is  $O(\log n)$ .

We conclude that for registration and messaging, the relative performance of each system is highly dependent on the ratio of mobile objects

to hosts in the network. However, the poor performance of the update operation in Armada, and the recurring need for each mobile object to recalculate its closest neighbouring set restricts its applicability to large mobile object communities and highly dynamic networks. As the size of the mobile object population has no bearing on AMOS' performance, it is more suited to such environments.

## 6. Conclusion

This work contributes to the simplification of development in mobile and ubiquitous computing environments – providing a layer of transparency over the problematic issue of reliable discovery of mobile objects. The Host Routing strategy provides a scalable and robust method of discovery, resolving the limitations of existing approaches. The complexity of Host Routing in terms of network hops is  $O(\log n)$ , whilst the communication cost is  $\approx 25$  ms more than the currently preferred strategy. We demonstrated a load balancing application that leverages this strategy to make optimal use of available hosts without prior knowledge of their capabilities.

**Acknowledgments** This work is partially supported by Science Foundation Ireland under grant number 04/RPI/1544, 'Secure and Predictable Pervasive Computing'.

## References

- 1) Fuggetta, A., Picco, G.P. and Vigna, G.: Understanding code mobility, *IEEE Trans. Softw. Eng.*, Vol.24, No.5, pp.342–361 (1998).
- 2) Zaslavsky, A.: Mobile agents: Can they assist with context awareness?, *Proc. IEEE Int. Conf. on Mobile Data Management (MDM'04)* (2004).
- 3) Ratnasamy, S., Francis, P., Handley, M., Karp, R. and Schenker, S.: A scalable content-addressable network, *Proc. 2001 Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pp.161–172, ACM Press (2001).
- 4) Stoica, I., Morris, R., Karger, D., Kaashoek, F. and Balakrishnan, H.: Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications, *Proc. 2001 ACM SIGCOMM Conf.*, pp.149–160 (2001).
- 5) Rowstron, A. and Druschel, P.: Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems, *Lecture Notes in Computer Science*, Vol.2218, pp.329–350 (2001).
- 6) Lua, E.K., Crowcroft, J., Pias, M., Sharma, R.

- and Lim, S.: A survey and comparison of peer-to-peer overlay network schemes, *IEEE Communications Survey and Tutorial*, Vol.7, No.2 (2005).
- 7) Venners, B.: The architecture of aglets (Apr. 1997).
  - 8) Walsh, T., DiCeglie, J., Young, M., Wong, D., Paciorek, N. and Peet, B.: Concordia: An infrastructure for collaborating mobile agents, *Proc. 1st Int. Workshop on Mobile Agents (MA '97)* (1997).
  - 9) Jul, E., Levy, H., Hutchinson, N. and Black, A.: Fine-grained mobility in the Emerald system, *ACM Trans. Comput. Syst.*, Vol.6, No.1, pp.109–133 (1988).
  - 10) Agentspace. <http://www.agentspace.co.uk>
  - 11) CC/PP: <http://www.w3.org/mobile/ccpp>
  - 12) Tripathi, A., Karnik, N., Ahmed, T., Singh, R., Prakash, A., Kakani, V., Vora, M. and Pathak, M.: Design of the ajanta system for mobile agent programming, *Journal of Systems and Software*, Vol.62, No.2, pp.123–140 (2002).
  - 13) Hsiao, H-C., Huang, P-S., Banerjee, A. and King, C-T.: Taking advantage of the overlay geometrical structures for mobile agent communications, *IPDPS* (2004).
  - 14) Hsiao, H-C. and King, C-T.: Tornado: A capability-aware peer-to-peer storage overlay, *Journal of Parallel and Distributed Computing*, Vol.64, No.6, pp.747–758 (2004).

(Received January 22, 2007)

(Accepted April 6, 2007)

(Released June 6, 2007)



**Richard Glassey** is a Ph.D. candidate studying in the Computer and Information Sciences department at the University of Strathclyde, Glasgow, Scotland. He is currently affiliated with the Software Engineering Group. Research interests include issues in mobile and ubiquitous computing, disruption tolerant systems and fostering emergent behaviour in mobile agent systems.



**Graeme Stevenson** joined the Systems Research Group at University College Dublin in April 2005 as a Research Assistant. He holds a bachelors degree in Computer Science from the University of Strathclyde. Graeme's interests include ubiquitous computing, mobile object systems, semantic web technologies, and programming language support and middleware for smart spaces. He is heavily involved in developing the Construct middleware platform for context collection and dissemination in *ad hoc* environments (<http://construct-infrastructure.org>).



**Robert Ian Ferguson** is currently a Lecturer in the Department of Computer and Information Sciences at the University of Strathclyde, Glasgow, Scotland. He holds a Ph.D. in Software Engineering from the University of Sunderland, UK (1998). His research interests include mobile/location aware computing and agent based systems.