Mobile Object Location Discovery in Unpredictable Environments

Richard Glassey[†], Graeme Stevenson[‡]and Robert I. Ferguson[†]

[†]Global and Pervasive Computing Group, University of Strathclyde, U.K.
[‡]Systems Research Group, UCD Dublin, I.E.
{rjg, if}@cis.strath.ac.uk
graeme.stevenson@ucd.ie

ABSTRACT

Emerging mobile and ubiquitous computing environments present hard challenges to software engineering. The use of mobile code has been suggested as a natural fit for simplifying software development for these environments. Existing strategies for locating mobile code assume an underlying fixed, stable network. An alternative approach is required for mobile environments, where network size and reliability cannot be guaranteed. This paper introduces AMOS, a mobile code platform augmented with a structured overlay network. We demonstrate how the location discovery strategy of AMOS has better reliability and scalability properties than existing approaches, with minimal communication overhead. Finally, we show how AMOS can provide autonomous distribution of effort fairly throughout a network using probabilistic methods that requires no global knowledge of host capabilities.

Keywords: Mobile Code, Structured Overlay Network, Location Discovery

1 Introduction

The proliferation of mobile and ubiquitous devices introduces hard challenges to the domain of distributed computing. These challenges include managing disconnected operation, where a device may have intermittent network connectivity, making optimal use of a device's limited resources, and host failure where a device may simply run out of battery power or crash unexpectedly. Developing software for these unpredictable environments is therefore much harder than for traditional distributed systems.

The use of mobile code [1] has been suggested as a natural fit for managing some of the challenges of developing software for mobile and ubiquitous computing environments [2]. Host failure, through loss of power, can be mitigated by migrating a process to another device to continue its operation. Disconnected operation can be enabled by migrating a process from a device with weak connectivity into a network of stable hosts to complete its task before returning to the original device. Intensive processes can be migrated from a resource constrained device to another device that has better resources available. These compelling reasons suggest that mobile code has something to offer in the domain of mobile and ubiquitous computing.

However, as soon as mobile code is set free to roam around a network, tracking its location becomes important for supporting communication with other processes. To prevent mobile code from becoming unreachable, location discovery strategies (centralised registry, multicast and home server) have been developed. Unfortunately, each of these strategies has serious flaws that render them neither reliable nor scalable for the unpredictability displayed in mobile and ubiquitous environments.

This paper describes AMOS (Adaptive Mobile Object System), which provides better location discovery of mobile objects[1], in terms of reliability and scalability, than existing strategies, with minimal communication overhead. We introduce the Host Routing strategy, which makes use of an overlay network to discover the location of mobile objects. Overlay networks, such as CAN [3], Chord [4], Pastry [5] et al, are distributed systems that do not rely on centralised control or hierarchical organisation [6]. They are typically self-organising networks, layered upon an IP-based network, that use a flat logical addressing scheme. Each host only needs knowledge of $O(\log n)$ other hosts, yet can send a message to an unknown host, through other hosts using key-based routing - in both cases n is the number of hosts. This creates a global yet distributed index of hosts which is scalable, efficient and reliable.

In AMOS, each mobile object has a globally unique identifier (GUID) that is used to generate a unique network identifier (UNID) in the overlay network address space. When a mobile object migrates, a registration process routes its new location to the host in the overlay network which has the closest UNID to its own generated UNID. 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.

If a process needs to contact a mobile object, the discovery process routes a location request message to the host with the closest UNID to the mobile object's UNID in order to determine the last known location. 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 milticast based solution due to effcent routing; and achieves scalability by distributing the management effort throughout the system.

The remainder of the paper is structured as follows. Sec-

---

[1]Due to object-oriented nature of AMOS, we use mobile object instead of mobile code, process or agent

tion 2 outlines three existing strategies for discovering the location of mobile code and identifies their shortcomings. 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 technique 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 investigating the fusion of mobile code platforms and peer-to-peer systems. Finally, in Sec. 6, we present our conclusions of this work.

2   Location Discovery

Knowledge of where a mobile object is currently located within a network is essential in order to communicate with it. This section looks at three strategies - centralised registry, multicast and home server - and discusses their respective advantages and disadvantages.

2.1   Centralised Registry

Perhaps the most basic way of providing location discovery of mobile objects is to maintain a fixed host that is responsible for tracking their location throughout their life-cycle. This centralised registry strategy is simple to implement, only requiring the mobile objects to report their location to a well known registry service and for other processes to be aware of the service. Aglets [7] and Concordia [8] 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. Despite being efficient, this is not a robust solution, especially when a system may consist entirely of mobile and unreliable hosts. This strategy creates a single point of failure, and introduces the potential for performance bottlenecks should location updates or requests be numerous.
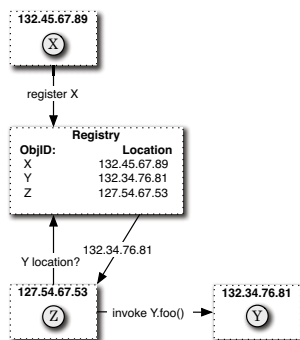

Figure 1: Registration and look-up in a centralised location discovery strategy

2.2   Multicast

A multicast strategy involves propagating a discovery request throughout a network of hosts with no reliance upon a registry service. Emerald [9] is a mobile code framework that uses the multicast strategy. Figure 2 shows a typical discovery process where sender S propagates a location request through the network in order to discover the location of target T. Whilst multicast is more efficient than broadcast, it still is costly in terms of communication because the sender floods the network until the host containing the target is discovered. For resource-constrained devices, network communication is a costly activity and should to be minimised where possible.
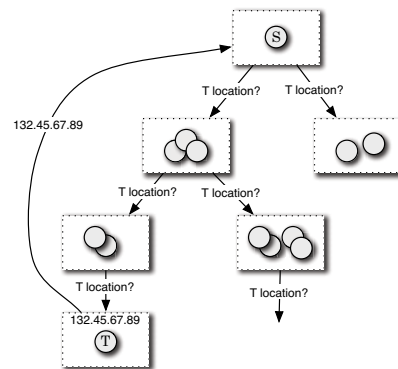

Figure 2: Location discovery using a multicast strategy

2.3   Home Server

The home server strategy distributes the registry of mobile objects, thus improving reliability to a degree and reducing communication overhead. AgentSpace, the mobile code framework that AMOS is built upon, makes use of the home server discovery strategy [10]. It operates on the principle that each host that launches a mobile object becomes responsible for tracking its location throughout its life-cycle (illustrated in Fig. 3).
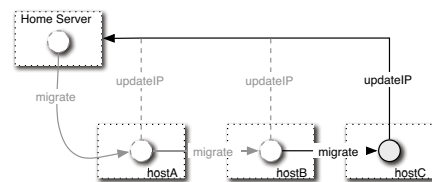

Figure 3: Registration of IP with home server

When a mobile object is launched, its globally unique identifier (GUID) includes the IP address of the home server. Every time it migrates, the mobile object contacts its home server and notifies it of its new location. If another process needs to contact the mobile object, it must inspect its GUID and contact the appropriate home server to get its current location".

Whilst this is an improvement over the previous two strategies, it still is not robust. If its home server fails, a mobile object cannot update its location, and no other process can contact it. Furthermore, if one host launches many mobile objects, it may become a performance bottleneck.

3 Architecture

The shortcomings of the location discovery strategies described in Sec. 2 limit the use of mobile object based systems for mobile and ubiquitous computing environments. We now describe the architecture of AMOS, which solves the problem of providing a reliable and scalable location discovery strategy for mobile objects in unpredictable environments - reducing the difficulty of developing a distributed application. We outline the high level architecture of AMOS; detail how the reliable location registration and discovery of mobile objects is achieved using key-based routing; and illustrate how AMOS can make the best use of resources in a network of heterogeneous hosts by using probabilistic methods.

3.1 Architectural Overview

Most mobile code frameworks depend upon the notion of a host running a container process that can launch, host and receive mobile objects. The container provides the execution environment in which 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 that prevents a malicious mobile object from damaging the host. Mobile objects are free to migrate to any other host that is running the container process.

In AMOS, a network of hosts, besides running containers, attempts to form a structured overlay network, using the protocols developed by the Pastry project [5]. An overlay network is simply an abstraction over physical networking that permits hosts to address each other with logical addresses [6]. No host has a global view of the entire network, instead each host becomes part of a distributed hash table. A sub-set of host addresses from the global address space are also stored to aid message routing. This set is generally of size $O(\log n)$ where n is the number of hosts. Hosts therefore must communicate indirectly by using key-based routing unless they already know the host address that they wish to contact. Messages are deterministically routed through the overlay network address space using a greedy approach, whereby a host forwards a message to a member of its routing set that has the closest network identifier to the target network identifier. The performance of key-based routing is $O(\log n)$ where n is the number of hosts.

Figure 4 illustrates a high-level view of AMOS on a single host, where a HostManager and its associated helper components are running in a container with other mobile objects. When a new HostManager component is created on a host, it uses a well known IP address (or range of addresses) to bootstrap itself into an existing overlay network using the HostRouter and Node components. The bootstrap phase in-
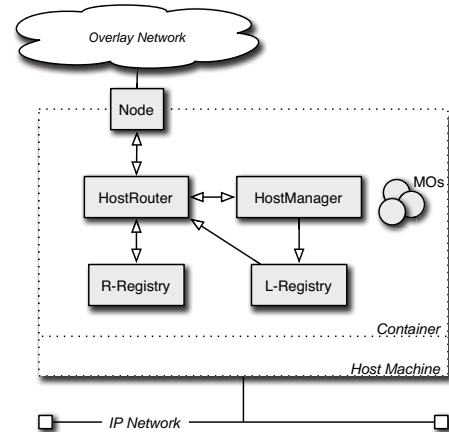

Figure 4: High-level architecture of AMOS on a single host

volves choosing a unique network identifier (UNID) and determining the set of other Nodes that this HostManager is aware of. The Node manages the HostManager's position in the overlay network by building and maintaining a routing table and a leaf-set of other Nodes, its part of distributed global index. The process of bootstrapping into overlay networks is described in greater detail in [5]. The L-Registry (local registry) is a simple component that stores the mobile objects currently on this host, notifying the HostRouter of all mobile object arrival events. The R-Registry (remote registry) component stores records of mobile objects identities and IP locations that this host is currently responsible for, forming part of the distributed index of mobile objects. The next section explores how these components interact to handle the registration and location discovery of mobile objects.

3.2 Location Discovery

To provide a better location discovery strategy than the strategies discussed in Sec. 2, we must be able to minimise the amount of communication required to locate a mobile object irrespective of network size, and ensure fault tolerance in the face of host failures, whether permanent or temporary. To demonstrate how AMOS satisfies these requirements, this section details the interaction between a mobile object (MO) and the HostManager during the registration process, and describes the location discovery strategy that allows MOs to communicate without prior knowlege of each others location

3.2.1 Mobile Object Registration

A well designed registration service should facilite an efficient, reliable and timely lookup of a mobile object's location. The centralised registry and home server strategies both have the problem that if a host fails, it becomes impossible to reach the MO. Within AMOS, an MO arriving at a host is detected by the HostManager, which starts a registration process. Rather than register the MO on this host, or choose a specific host address, AMOS registration takes the novel
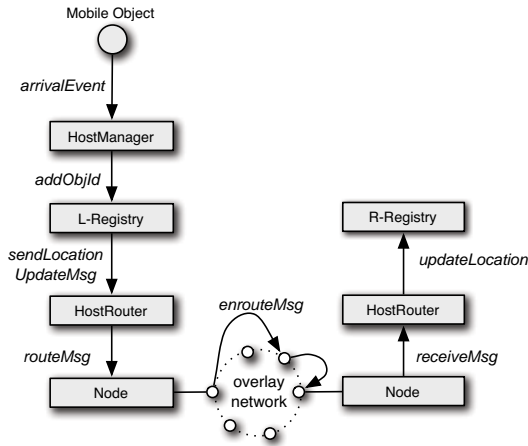
Figure 5: Registration of a Mobile Object in AMOS

Figure 6: Host Routing strategy for discovering location of a mobile object

approach of using the GUID of the MO to generate a valid UNID within the overlay network address space. It is not necessary that a node with the generated address exists in the network. AMOS handles such situations by choosing the choosing the numerically closest UNID that does have a host - a property provided by the overlay network. We therefore are always guaranteed to get an active host to handle the registration request.

Figure 5 illustrates component interaction during the registration process. The arrival of an MO generates an arrival event which the HostManager detects. The HostManager adds the GUID of the MO to its local registry (L-Registry). Adding the MO causes the local registry to notify any observers that a new MO has arrived. The HostRouter component is one such observer. Its purpose is to handle messages that are sent and received using overlay network routing. Upon notification of the arrival of a new MO, it creates a new location update message containing the MO's GUID and the host's IP address. This message is then dispatched to the Node component which is the access point into the overlay network. The Node forwards the message onwards (to the next Node within its routing table possessing the numerically closest UNID) based on the UNID generated from the MO's GUID. Once the message arrives at its destination it is registered in the remote registry (R-Registry) of the HostManager component. This completes the registration process.

This process re-occurs every time an MO migrates from one host to another. The chief benefit of this indirect form of registration is that should the host which has the UNID closest to the MO's generated UNID fail, the next active host with the closest UNID will assume responsibility for registering the location of the MO. This process is transparent to the MO and the HostManager attempting to register it with another host because the overlay network takes care of the UNID resolution.

The only situation of concern is when the host managing an MO's location information fails and the MO does not migrate for a length of time, or ever again. Any process attempting
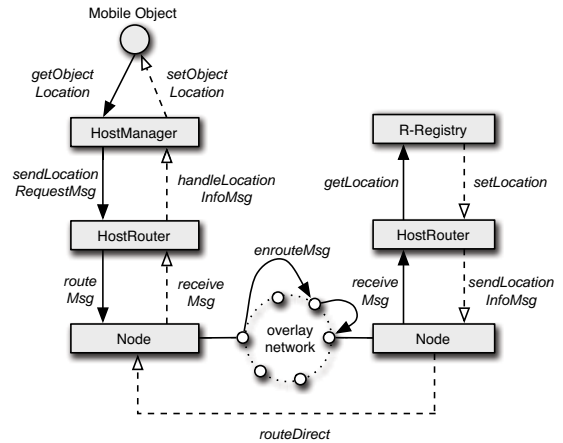
to contact the MO would be unable to find the host responsible for storing the MO's location. Each node in the network maintains a set of other node UNIDs to facilitate routing and maintain the integrity of the network. Periodically, this set is monitored check for node failures. AMOS leverages this process by ensuring that each node replicates the remote registries of the other nodes in their set. If a host failure is detected, the discoverer can invoke the registration process described above for all MOs in the registry belonging to the failed host to repair the registry. This approach increases the reliability of registering MOs and occurs transparently. By taking advantage of the self-organising structure of the overlay network, we can avoid using less efficient strategies, such as forcing MOs to re-register after a time-out of not migrating, or requiriing HostManagers to monitor MO's that have not recently moved.

3.2.2 Discovery of Location

With the exception of multicast, other location discovery strategies involve contacting specific hosts known to manage the location information of MOs. As noted earlier, multicast is an expensive discovery method in terms of network communication overhead. The discovery strategy used in AMOS, Host Routing, uses the distributed index of MOs that is created by the registration process to achieve reliable look-up, whilst significantly limiting network overhead. The caveat of Host Routing is that the requester must already possess the GUID of the target MO. As long as the requester has the GUID of the target, then AMOS guarantees the discovery of the location information.

To begin with, the requester contacts its local HostManager and issues a request for the location of a target MO. The HostManager delegates the task to the HostRouter. It generates a location request message and passes it to the node, which routes it towards the UNID generated from the target MO's GUID. This message is forwarded through the overlay

network until it reaches its destination, the node of the host responsible for the location of the target MO. Upon receiving this message, the HostRouter on the remote host performs a look-up within its remote registry using the target MO's GUID as a key, which returns its last known IP location. The HostRouter then routes this information directly back to the host machine that issued the request. Once the originating HostManager receives the message, it calls back the requester and informs it of the IP location for the target MO. 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 info message generated in response.

The requester can now reliably communicate with the target MO to achieve its goal. The registration process described in the previous section ensures that the location of the target MO will be correct. As the actual paths that messages are routed across to reach their destination are not fixed, the overlay network can sustain damage whilst maintaining the ability to route messages correctly - a property inherited from the overlay network

From the developer's perspective, the process of locating a mobile object using the Host Routing strategy occurs transparently. As with other strategies described in Sec. 2, developers obtain a reference to a MO by calling a method that takes an object ID as a parameter. Thus, the use of the Host Routing strategy has no impact on the complexity of application development.

3.3 Distribution of Effort

Balancing of effort, or load, is particularly important for mobile and ubiquitous computing environments where reliable and capable hosts may be in short supply. Although it can be beneficial to migrate processes, it is important to consider where to send them, and the implications of doing so. It is preferable if the system collectively approaches a balanced state with no global knowledge required, rather than maintain a list of reliable resources in a network.

To achieve autonomous load balancing in AMOS, we modify the architecture in two ways: by allowing hosts to occupy variable amounts of address space; and by modifying MOs to prefer more capable hosts.

A single host can occupy more address space by spawning multiple nodes - in effect, making itself more 'visible' in the overlay network. Figure 7 demonstrates how this is achieved by introducing a new component, the VirtualNode-Handler. The VirtualNodeHandler acts as a multiplexer, randomly choosing a single node to forward an outgoing message, whilst passing all messages from all nodes back to the HostRouter. This increases the probability that an MO will arrive when migrating from one host to a randomly selected host. Furthermore, this host will handle a greater percentage of registration requests because there is a higher probability that more network traffic will be handled by the host's collection of nodes.

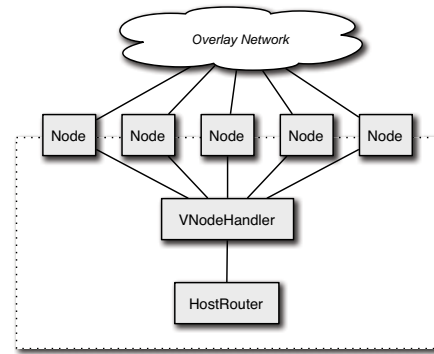Although this process allows more capable hosts to han-

Figure 7: Multiple nodes per single host

dle more work, it does not prevent MOs from migrating to a resource constrained host. To to mitigate this, MOs are modified to prefer more capable hosts. They achieve this through host introspection . Apart from requesting the location of another MO, they can also query how 'busy' the host currently is through the HostManager. At present, a HostManager defines its host load by dividing a host capability metric by the number of MOs currently active on this host. Host capability can be customised (e.g. using a device profiling standard such as CC/PP [11]), but for the sake of simplicity, we define it as a numerical value reflecting the class of device the host is. A resource-constrained device would have a low number, whilst a fixed server would have a high number.

On arrival at a new host, an MO uses host introspection to determine if this is a good host on which to carry out its activities. If it is not, it migrates to a random host. The MO chooses a random host by generating a random UNID, requesting the IP of the host that has the closest UNID (similar to selecting an active host to register with), and migrates there instead. When this random migration is combined with more capable hosts representing a larger proportion of the address space of an overlay network, effort will be distributed fairly without needing any centralised co-ordination. Whilst simple, relying purely on probability that an MO will get to a capable host, this enables an autonomous load balancing scheme in AMOS that fairly distributes the effort according to host capability.

4 Evaluation

This section details the evaluation of AMOS. We demonstrate that the cost incurred using the registration process does not significantly vary as the network size increases; the cost of the using the Host Routing strategy is competitive with the home server strategy whilst significantly improving reliability; and finally that autonomous load balancing can be achieved using decentralised control and probabilistic methods

The following experiments were conducted using a network of twenty Sun Blade workstations, running Solaris 2.5.1, connected via 100-BaseT ethernet. This setup was chosen for two practical reasons: it is costly to assemble a large enough
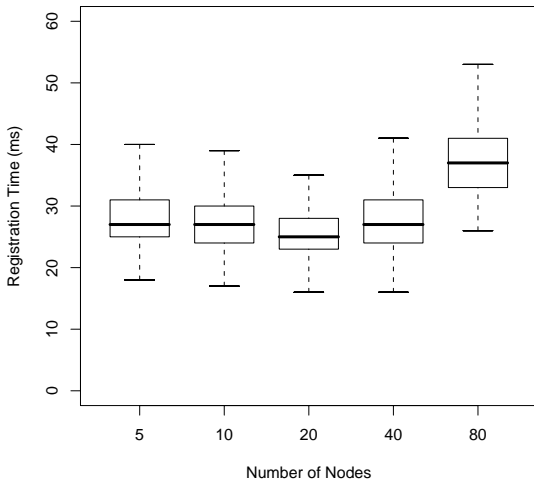
Figure 8: Cost incurred, in time (ms), for registering a mobile object in network of increasing size

Figure 9: Completion times for RMI look-up, Home Server and Host Routing strategies

collection of mobile devices to achieve significant results (although this would give a better idea of unpredictability); the focus of the experiments is to demonstrate that the benefits provided by AMOS come without a significant cost compared to other methods independent of execution environment.

4.1 Evaluation Methodology & Results

4.1.1 Registration costs over increasing network size

The first experiment measures how much cost, in terms of time, is incurred when using the mobile object registration process, detailed in Sec 3.2.1, on networks of different sizes.

The source code of the HostManager was altered to output a time-stamp for the beginning and end of the registration process for a mobile object. Because the registration process occurs on separate machines, a Network Time Protocol server is used to synchronise all hosts, avoiding significant clock discrepancies.

A network of n nodes is created across the hosts. For each size of network, a single mobile object is launched onto one host. This invokes the registration process. Once complete, the time-stamps can be subtracted to give the total time it took to register the mobile object. This is repeated 10K times for each size of network. The results are shown in Fig 8.

As the network increases in size, the average cost to register a mobile object remains mostly within the range of 25–40ms. 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. The narrow ranges suggest that for each network size, registration cost remains consistent.
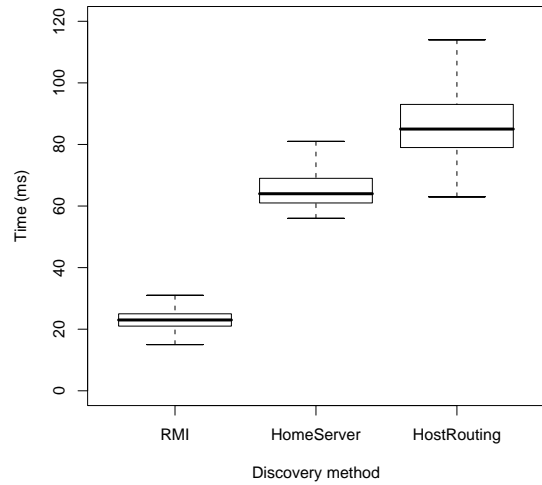
4.1.2 Location discovery of a mobile object

The second experiment measures the cost, in time, of discovering the location of a mobile object using the Host Routing strategy compared to the home server strategy. We measure the total time it takes to discover the location and invoke a method upon the remote mobile object.

Two mobile objects, Caller and Receiver are launched into the network of hosts. The Receiver migrates to a randomly selected location in the network and waits. The Caller then attempts to discover where the Receiver is, using each of the discovery strategies. Firstly, as a benchmark for comparison, the Caller uses Java RMI with the known IP location of the Receiver to invoke a method. This measurement indicates the network cost without any discovery process. Secondly, the Caller uses the Home Server discovery strategy provided by AgentSpace to invoke a method on the Receiver. Finally the Caller uses the Host Routing strategy to discover the location of the Receiver. Each attempt to discover and invoke a method is repeated for 10K iterations, mitigating the effects of any host or network anomalies. Figure 9 shows a series of boxplots that summarise the completion time for each method.

The Host Routing strategy is more costly than the home server method with mean completion times of 91.5ms and 67.4ms respectively, and displayed a wider distribution of results. We believe that ≈ 25ms is a reasonable extra cost when the reliability and self-healing properties of Host Routing is taken into consideration. AMOS automatically repairs the registry when a host fails so there are no extra costs incurred, justifying the slight increase in discovery time.

4.1.3 Autonomous Load Balancing

Because mobile and ubiquitous computing environments will most likely consist of a collection of devices that have differ-

Decision	A (1n)	B (2n)	C (4n)	D (6n)	E (8n)
0	500	0	0	0	0
1	11	48	100	160	181
2	8	48	99	155	190
3	8	50	94	152	196
Optimal	*24*	*47*	*95*	*144*	*190*

Table 1: Distribution of mobile object population after three decisions

ent capabilities, it makes sense to make optimal use of the resources available. However, without prior knowledge of the devices that are more capable, optimality is difficult to achieve. This experiment illustrates how using AMOS ensures that a population of mobile objects will attempt to make best use of the available resources, whilst avoiding resource-constrained devices.

We use the host capability measure outlined in Sec 3.3 to vary the number of nodes that a HostManger deploys. For the purpose of this experiment, we simulate a range of devices with various capabilities. Type A is a resource constrained device and will only deploy one node, whereas type E is a more capabale device that will deploy eight nodes. We build a network using five of each device 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 amount of time, uses host introspection and makes a decision whether to stay or migrate to a random location. Table 1 illustrates how the population of mobile objects distributed themselves after three decisions. It is clear that just after one round of decisions the distribution is already approaching optimal in the sense that higher capability hosts are handling more mobile objects. 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.

However, it is not clear if significant oscillation is occuring where a mobile object may be constently switching between devices. Subsequent runs created similar distributions of mobile objects across the devices. The shape of this distribution is easily controlled by the choice of how devices are classified. Whilst this example is artificial, it illustrates how the design of AMOS can be easily extended to achieve decentralised control, in this case balancing the load proportionally throughout the network of hosts.

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 to that of AMOS.

5.1 Mobile Objects and Agents

The majority of existing mobile object and agent platforms make use of the object location strategies described in Section 2. Aglets [7] and Concordia [8] implement a centralised registry, whilst AgentSpace [10] and Adjanta [12] use home server based techniques. Emerald [9] uses a multicast strategy as backup to its primary technique, which involves a mobile object leaving a forwarding pointer on migration.

Although applicable in many domains, such as parallel processing and distributed search, there are none where mobile agents are unique in providing a solution. Other technologies can usually be used in combination that yield equal or better performance results. However, mobile agents provide a generic framework with which to easily implement a wide range of distributed applications. The technology excels in dealing with dynamic or volatile network environments, without requiring the developer to a master a number of application specific techniques.

5.2 P2P Overlay Networks

Overlays that provide an address space over a subset of a network are a useful tool 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). This project is interested in P2P overlays, for their capability to provide object location, messaging, and node organisation in ad-hoc networks. Such overlays also provide population management features (self-organisation and self-repair) that are essential for the management of ad-hoc networks.

AMOS is built on top of Pastry [5]. 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 [3] and Chord [4] 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 given key. CAN maps a virtual d-dimensional Cartesian coordinate space across all participant nodes, with each node responsible for all points within a given zone. The number of routing-hops in the CAN lookup algorithm grows faster than $O(\log n)$.

5.3 Hybrid Approaches

Although mobile objects and agents have long been touted as a promising technology for use in ad-hoc networks, existing object location algorithms have lacked the robustness and efficacy required for such environments. In this section we compare AMOS to another system which has tried to address this problem through the use of a P2P overlay, Armada [13].

Armada is an agent communication system built on top of Tornado [14]. In Armada, a community of agents running on a set of hosts autonomously manage their communication. Each agent has a GUID, and keeps a list of other agents that it knows how to send messages to. If an agent wishes to send a message to a node that it does not know how to contact directly, it looks up the agent whose key is numerically closest to, but lower than the key of the destination home. This receiving node then repeats this process until the message reaches its destination. Multiple communities of agents may run on the same set of nodes, but use only agents in the same community to aid routing.

On joining a community, an agent constructs a set of pointers to nodes in the community. Agents periodically perform a secondary algorithm to ensure that they maintain pointers to the closest neighbouring set. On migration, a mobile agent updates its location in the community by finding the agent platform numerically closest to its own. An agent on that node receives the updated location of the agent and joins a community of agents responsible for storing the location of mobile agents in the community.

From the results in [13], Armada appears to perform best when the size of a community of agents is small. The number of messages required for an agent to join a community is $O(\log a)$, where a is the size of the agent community. This can be compared to $O(\log n)$ messages for an AMOS agent to register, where n is the number of nodes in the network. Armada incurs a cost of $O(\log a) + O(\log a)^2$ messages every time an agent migrates to update its community. In AMOS, agent migration requires $O(\log n)$ messages to be sent. Finally, sending a message to an Armada agent 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 agents to nodes in the network. However, the poor performance of the update operation in Armada, and the recurring need for each agent to recalculate its closest neighbouring set restricts its applicability to large agent communities and highly dynamic networks. As the size of the agent population has no bearing on AMOS' performance, it is more suited to such environments.

6 Conclusion

The task of developing software for mobile and ubiquitous computing environments is challenging, but the use of mobile code provides one route to easing the difficulty. However, the task of discovering mobile code location becomes a problem in unpredictable environments when using existing strategies, designed with fixed and relatively stable networks in mind. This paper has introduced AMOS, a mobile code framework that was augmented with a structured overlay network. This fusion provides an elegant solution to the location discovery problem in the form of the Host Routing strategy. It removes the single point of failure issue of centralised registry and home server strategies. The complexity of discovery with host routing is $O(\log n)$, and incurs a minimal cost of $\approx$ 25ms more than the home server strategy Finally, an autonomous load balancing scheme was presented that demonstrated how a population of mobile objects can distribute themselves fairly across a network of mixed capability devices without global knowledge.

REFERENCES

[1] A. Fuggetta, G. P. Picco, and G. Vigna. Understanding code mobility. *IEEE Transactions on Software Engineering*, 24 (5):342–361, 1998.

[2] A. Zaslavsky. Mobile agents: Can they assist with context awareness? In *Proceedings of the IEEE Int. Conf. on Mobile Data Management (MDM'04)*, 2004.

[3] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and Scott Schenker. A scalable content-addressable network. In *Proceedings of the 2001 Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 161–172. ACM Press, 2001.

[4] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and Hari Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In *Proceedings of the 2001 ACM SIGCOMM Conf.*, pages 149–160, 2001.

[5] A. Rowstron and P. Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. *Lecture Notes in Computer Science*, 2218:329–350, 2001.

[6] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim. A survey and comparison of peer-to-peer overlay network schemes. *IEEE Communications Survey and Tutorial*, 7 (2), 2005.

[7] B. Venners. The architecture of aglets, April 1997.

[8] T. Walsh, J. DiCelie, M. Young, D. Wong, N. Paciorek, and B. Peet. Concordia: An infrastructure for collaborating mobile agents. In *Proceedings of the 1st Int. Workshop on Mobile Agents (MA '97)*, 1997.

[9] E. Jul, H. Levy, N. Hutchinson, and A. Black. Fine-grained mobility in the Emerald system. *ACM Transactions on Computer Systems*, 6(1):109–133, 1988.

[10] Agentspace. http://www.agentspace.co.uk.

[11] CC/PP: http://www.w3.org/mobile/ccpp.

[12] A. Tripathi, N. Karnik, T. Ahmed, R. Singh, A. Prakash, V. Kakani, M. Vora, and M. Pathak. Design of the ajanta system for mobile agent programming. *Journal of Systems and Software*, 62(2):123–140, 2002.

[13] H-C. Hsiao, P-S. Huang, A. Banerjee, and C-T. King. Taking advantage of the overlay geometrical structures for mobile agent communications. In *IPDPS*, 2004.

[14] H-C. Hsiao and C-T. King. Tornado: a capability-aware peer-to-peer storage overlay. *Journal of Parallel and Distributed Computing*, 64(6):747–758, 2004.