SOLS: Self Organising Distributed Location Server for Wireless Ad Hoc Networks

Gareth Owen [1) and Mo Adda [2)

[1) Computing Laboratory, University of Kent, Canterbury, CT2 7NZ, UK.
E-mail: g.h.owen@kent.ac.uk
[2) School of Computing, University of Portsmouth, Buckingham Building,
Lion Terrace, Portsmouth, PO1 3HE, UK. E-mail: mo.adda@port.ac.uk

ABSTRACT

Ad hoc networks allow wireless devices to form a network without any pre-existing knowledge of configuration or topology. Large-scale networks consist of hundreds or more nodes and are unable to use traditional broadcast routing algorithms due to lack of scalability. Geographic algorithms use nodes' geographical locations to route traffic without topology knowledge, but still require knowledge of the destination node's location. This location is often provided up by location service protocols although many of these too use broadcast schemes which limit their scalability. This work describes a location server that uses self-organising behaviour that is often seen in nature to minimise transmission overhead and maximise scalability. The approach outperforms existing solutions by significant margins.

KEYWORDS

Geographical routing, Location server, quorum systems, wireless LAN.

1. INTRODUCTION

Ad hoc wireless networks allow nodes to communicate over multiple hops with no prior knowledge of network topology. Routing in ad hoc networks is often achieved with reactive routing protocols that utilise broadcast searches (e.g. AODV (Johnson et al., 2002)) to find paths between nodes. As a result, such networks often do not scale to large numbers of nodes due to capacity limitations and routing overhead. With larger networks routing paths have a shorter life time due to the increased average path length and the increased likelihood of one node moving.

An alternative to broadcast search schemes is to use geographical or location-based routing. The nodes, or subset of, are aware of their geographical location (e.g. latitude and longitude) and that of the nodes with which they need to communicate. Packets are often routed using locally optimal techniques such as each hop forwarding to the closest neighbour (Ryu et al., 2004). Voids in the network can present a problem and improvements have been suggested such as perimeter routing. (Karp, 2000).

Whilst geographic routing avoids the overheads incurred in broadcast schemes and is tolerant to topology change, a new problem arises of how to discover other nodes' locations. Many of the routing algorithms assume that the location information is known globally which is not always possible. Alternatively, location information can be provided by location servers which work in a similar fashion to the Home Location Register in cellular networks or Home Agent in Mobile IP, providing the location of the mobile node on request. Again many schemes have been proposed and these are described under Related Work.

Existing location servers often are not scalable, have high overheads or lack fault tolerance. In this paper we present a novel solution these problems using self-organising behaviour similar to that seen in nature (Eberhart et al., 2002) to provide location information to nodes in the network. We present the technique along with simulation results, analysis and details for implementation.

2. Related Work

Reactive routing protocols were successful because information was recent and overhead was only incurred when necessary. Drawing upon this to develop a location server, a number of authors have proposed approaches that are similar to their cousin routing protocols (e.g. proactive or reactive broadcast searches/dissemination). The Reactive Location Service (Kurkowski et al., 2005, Fußler et al., 2001, Lochert et al., 2003) is similar to AODV (Johnson et al., 2002) in its technique. A node shares its location with its neighbours through hello messages so they can make local forwarding decisions. When a node wishes to find the location of another node, it launches a broadcast search to all nodes in the network for the information. The advantage of this technique is that the information is always up-to-date and recent; however, the overhead in any broadcast search is O(N) and although this can be reduced marginally through optimistic broadcast schemes (Fehnker and Gao, 2006, Ryu et al., 2004) it inevitably places a limit on the scalability of the network.

This limit can be overcome by using lookup services that do not require broadcast searches, such as a directory service on a known subset of nodes; however using a particular subset of nodes for storing of *all* location information would put a burden on them and that area of the network, exhausting capacity and battery life. Therefore, any location service should be uniformly distributed across the network but allow discovery of the information with a broadcast search.

One such example is the Horizontal Vertical quorum (Liu et al., 2006) shown in Figure 1. Each node creates two packets containing their location and sends one northward and the other southward. Each node receiving such a packet records the location information and forwards the packet in its original direction. Any node then looking to query this information sends two query packets, one easterly and the other westerly. At some point these packets will intersect with a node in the north-south line of nodes storing the information. This approach however is highly susceptible to node mobility and also has a high overhead of O($\sqrt{n}$) that is not independent of network size (n).

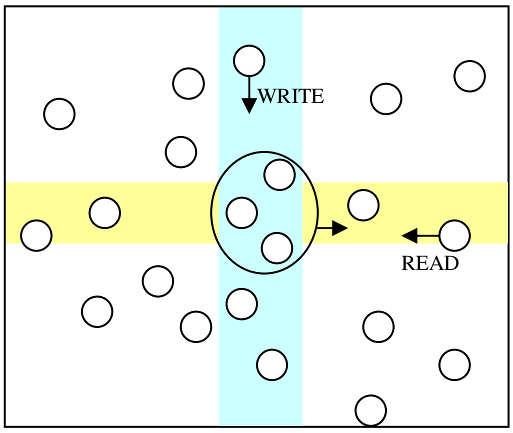

Figure 1: Horizontal/Vertical quorum

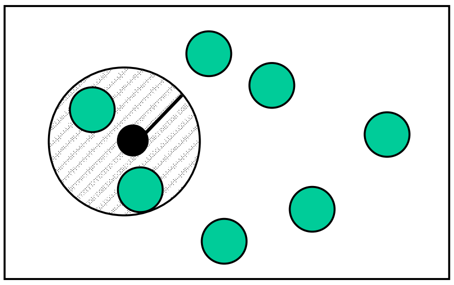

Figure 2: Home-region

The most scalable approach is that of the home region (Sivavakeesar and Pavlou, 2004, Hubaux et al., 2000) where the overhead is independently of network size (assuming uniform distribution). Each node is assigned a geographical region where all nodes in that region become location servers for it. That node frequently sends its location information to all nodes in that region who then record and answer queries from others. The technique scales independently of network size when the regions are distributed uniformly which is usually achieved by using a hash function. This approach is the most similar to our work but the authors propose no method to tolerate node mobility or failure. For example, all the nodes could move out of the region (or fail) in between updates resulting in the information being unavailable.

Our work addresses many of the issues outlined by using a novel self-organising approach to storing information at or near to a specific location. The next sections describe and analyse the performance of our approach.

3. Self Organising Location Servers (SOLS)

The approach described in this paper uses self-organising behaviour to support a home-region style location server. We describe the approach and analyse its performance.

The first key difference from the existing home-region approaches is that SOLS uses the concept of a home location rather than a region. This home location is a geographical co-ordinate within the bounds of the network and can be decided on by use of a hash function or by prior assignment. Implementation of such a hash or other mechanism is outside the scope of this paper.

Firstly, the approach requires that nodes know the geographical location of all their 1-hop neighbours. To achieve this each node beacons periodically advertising its node identifier (IP/MAC address) and its geographical location obtained from an onboard GPS or by other collaborative discovery mechanism (Michalson and Ahlehagh, 2004).

Each node in the network is capable of receiving and hosting autonomous agents. Initially, when a node powers up it creates an autonomous agent and provides it with the node's current and home location. This agent will then be routed to the closest node to the home location using geographical routing. On reaching the closest node it will replicate itself to create a defined number of slave agents which will reside on adjacent nodes. This group of agents storing one node's current location are called a Group of Location Servers (GLS), where all agents can service location requests. It is important to note, however, that a node can only host one agent from any particular GLS. A node can host several agents if they are from different GLSes. This is to prevent all agents from one GLS occupying one node and the failure of that node resulting in the loss of the entire GLS.

Beaconing

Each node must be aware of its own location (e.g. through GPS) and that of its neighbours to allow for geographical routing to take place. Therefore, each node periodically broadcasts a beacon packet (B_n) detailing its node identifier and its current (at time t) geographical location (n_x, n_y). In addition, it also broadcasts a list of the agents that it hosts at that time (n_S). This list consists of the identifier of the node for which each agent represents.

$$B_n(t) = \left(n_{ID}, n_x(t), n_y(t), < n_S(t) >\right) \qquad (1)$$

When a node receives a beacon packet from a neighbouring node, it is stored and can be accessed by agents it hosts. Assuming the beacon interval is I, then the beacon information will timeout after a time of $2I$. If this happens then the neighbour is assumed to have moved out of range.

Agent behaviour

Consider the example in Figure 3, where there are two home locations for two different nodes. The figure depicts two GLSes with each residing around the home location for its owning node and each consisting of three agents. Any agent in a GLS on the node closest to the home location is called the master and all other agents in that GLS are called slaves.

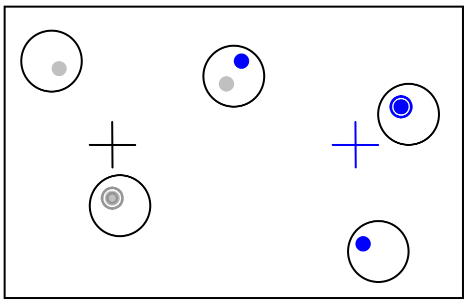

Figure 3: Two GLSes with agents from each occupying the same node

The goals of the GLSes are to provide location information for their owning node and so they must meet the following criteria:

1. **Discoverability:** They must be discoverable by other nodes wishing to query them. To do this they must remain on nodes near to the home location so that simple geographic routing to the home location will encounter a node with an agent on.
2. **Redundancy:** Nodes in the network are prone to failure and so the GLS must contain enough agents on different nodes so that the failure of a handful of nodes will not result in total loss.
3. **Recovery:** The GLS must recover from node failures so that repeated failures will not cause loss of the GLS.
4. **Service:** The GLS should provide location information on request and be updatable by the owning node.
5. **Minimise overhead:** By using a self-organising approach with no central control overhead is minimised and a function of the environmental conditions (node failure rate, mobility, etc).

Each agent acts independently of the GLS but in such a fashion so as there is an emergent intelligence through their self organising behaviour. To minimise communication overhead each agent can only inspect the beacon packets of neighbouring nodes and cannot communicate with other agents directly.

We designate the task of recovery to the agent closest to the home location (HL), and so it becomes the master of the GLS. Every other agent becomes the slave, although their roles are changeable if the node they are on move resulting in them being closest. The master performs the additional task of replication but does not directly control other agents.

Each agent has a set of tasks it performs every time its hosting node receives new beacon information resulting in an emergent behaviour:

1. **Migration:** Each agent checks to see if it is on the closest node to the HL, and if there is a closer node not hosting an agent of the same GLS it will migrate to it. This ensures that all agents stay on the closest nodes without resulting in two agents of the same GLS on one node.
2. **Election:** Each agent tests to see if the current node is the closest to the home location, and if so changes its role to master. Otherwise, if there are closer nodes holding agents, the role changes to slave.
3. **Replication (master only):** The master examines the beacon packets of neighbouring nodes and counts the number of agents; if this number has fallen below a threshold a replication function is executed.
4. **Pruning (slaves only):** Each agent examines the beacon packets of neighbouring nodes and if the number of agents is above a threshold, then it deletes itself with a certain probability *P*.

Shown below is the pseudo code for the MigrateCheck() function, which determines when an agent is eligible for migration. The function first checks to determine if an agent is eligible for migration and if so performs it. It does this by examining the beacon cache of the current node looking to see if there is a node closer to the HL than it is and does not already have an agent from the same GLS.

```
MIGRATECHECK()
dist ← $\infty$
closest ← none
For each n in neighbours
    if DIST(n, HL) < DIST(self, HL) and
       a.id is not element of <$n_S$> then
        dist ← DIST(n, HL)
        closest = n
```

```
8    if dist != ∞ then
9        MIGRATETO(closest)
10       return true
11   else return false
```

Next we show the ManageAgent() function which is called when the hosting node's beacon cache changes. This function decides when to perform Migration, Election, Replication and Pruning. Initially the function determines if the agent is on the node closest to the home location and if so elects that agent as the master, or otherwise makes it a slave. The function then calls the MigrateCheck() function to determine if the agent can move to a more suitable node. If not, and the agent is the master, it examines the number of agents on neighbouring nodes and if this is below a threshold it then replicates. Finally, if none of the above are executed and the agent is a slave, then the agent examines the number of agents on neighbouring nodes and prunes with a particular probability if the number is above a threshold.

```
MANAGEAGENT()
1    count ← 0
2    role ← master
3    for each n in neighbours
4        if n.id is element of <$n_S$> then
5            count ← count + 1
6            If DIST(n, HL) < DIST(self, HL) then
7                role ← slave
11   if MIGRATECHECK() then return
12   if role = master and count < threshold then
13       MULTICAST( count)
14   else if role = slave and count > threshold and RAND() < P then
15       DELETESELF()
```

The value *P* is calculated to result in approximately the correct number of agents destroying themselves to bring the GLS back to the threshold number of agents. This is determines as follows where *count* is the current number of agents in the GLS:

$$P = 1 - \frac{threshold}{count} \tag{2}$$

This calculation does not guarantee the precise number will prune themselves due to the self organising approach but if too many are pruned then replication will recover. Likewise if too few prune then the excess will prune at the next invocation of the function.

Finally, we show the Multicast() function below which is called by the master when the number of agents in the GLS is below the threshold. The purpose of the function is to create the number of required agents to bring the GLS back to the correct number of agents by sending clones to any nodes without agents already. If there are insufficient nodes then the function replicates to all remaining nodes and the function is re-run at the next invocation of MigrateCheck() when mobility will hopefully have changed the number of neighbours. The count parameter is the number of agents needed to bring the GLS back up to the threshold number of agents.

```
MULTICAST(count)
1   <multicastList> ← ()
2   for each n in neighbours
3       if n.id is not element of <n_S> then
4           <multiCastList>.ADD(n.id)
5           if |multicastList| > (count – threshold) then break
6   if |multicastList|> 0 then
7       MULTICASTCLONETO(multicastList)
```

Simulation Parameters
To evaluate the algorithm we use the Jist/SWANS (Barr, 2004) ad hoc network simulator. We attempt to approximate a city centre by simulating a 1000x1000 area with 200 nodes. Although SOLS is designed for large-scale networks, it is not necessary to simulate the algorithm in a large-scale environment as the algorithm scales independent of network size if the GLSes are distributed uniformly. Therefore, we use 50 GLSes in a 200 node network providing sufficient validity in the results obtained.

The simulation parameters are described in

Table 1. The simulations are repeated ten times and the average result plotted as one point on the result figures. As a result, one result figure is made up of possibly hundreds of simulations. We choose to use six agents (*threshold* = 6) per GLS for reasons outlined later.

Table 1: Simulation parameters

Area	1000 x 1000 m	**Simulation time**	3 minutes
Transmission Radius	100m	**Propagation model**	Two-ray
Number of nodes	200	**Beacon rate**	1 second
Mobility Model	Random Waypoint *v = 1 – 10m/s; Pause time = 0s.*	**Simulation repeated for verification**	10 times
Number of GLSes	50	*threshold*	6

Node failures are simulated in the following manner: Every beacon interval a node draws a random number (0-1) and if this is less than the failure probability (as a decimal) then the node deletes all state information (including agents) and is moved to a random place in the simulation area. This allows the number of nodes in the network to remain constant whilst still simulating high node failure rates. The number of failures (*K*) per minute a node encounters can be calculated given the beacon interval (*B*) and the failure probability (*F*):

$$K = F\frac{60}{B} \tag{3}$$

Performance Analysis

This section sets out simulation results for the core SOLS algorithm and later we examine how it performs as a location server. Key attributes to examine are how the algorithm tolerates node failures and how much overhead it incurs. If a GLS is lost then a node is no longer reachable because other nodes are unable to discover its location.

Figure 4 illustrates how the algorithm performs given various numbers of agents per GLS (*D*) and varying failure rates. Increasing the number of agents increases the likelihood of a GLS surviving but this advantage becomes negligible once it reaches the *average number of neighbours of a node*. This is the result of the algorithm not performing multi-hop replication.

Nevertheless, given that nodes would be expected to remain switched on for hours or days, the algorithm survives exceptionally well even given battery lifetimes of less than one minute.

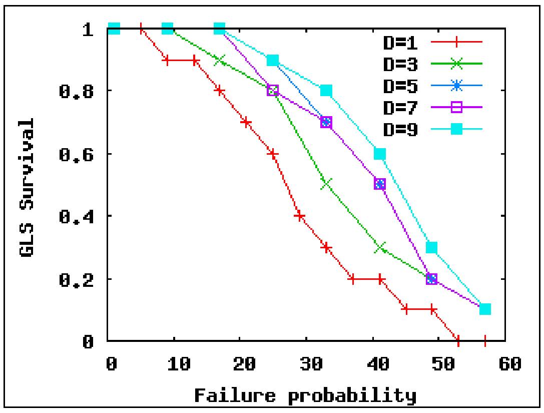

Figure 4: Failure tolerance

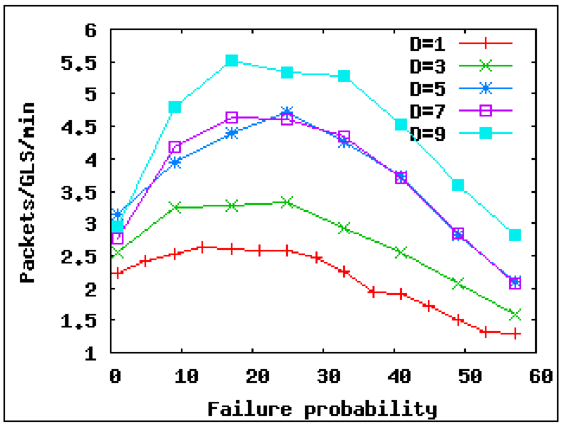

Figure 5: Overhead (hybrid)

Figure 5 shows the overhead in packets per GLS that the algorithm incurs per minute. Even with high numbers of agents the algorithm incurs less than 6 packets per minute. This is negligible when one considers that a packet containing agent state information would be a few tens of bytes and Wireless network speeds are now in excess of several hundred megabits per second/sec (IEEE, 2008).

Given higher numbers of agents the overhead increases as one would expect. The figure also demonstrates the self-recovery of the GLS with the overhead increasing to compensate for an increase in failure rate. Once the failure rate reaches around 25% then the algorithm struggles to cope and the overhead falls as a increasing number of GLSes are lost. A failure rate of 25% is very unlikely to be seen in real world applications.

4. SOLS AS A LOCATION SERVER FOR LARGE AD HOC NETWORKS

This section examines the applicability of the SOLS algorithm to the task of a location server for large ad hoc networks. A location server must meet the criteria described in the previous section. Therefore, we examine methods to update and query the GLS and also examine its performance in a large ad hoc network. Figure 8 gives a diagrammatic overview of how the SOLS approach works as a location server.

Updating the Location Server

Updating the GLS is also to be undertaken in a self-organising fashion to minimise overhead given there is no central control. A node that wishes to update its GLS (periodically, distance moved, etc) creates an update packet and routes it toward the home location. A node receiving this packet determines if it holds an agents belonging to that node's GLS. If so, that agent is updated and the node broadcasts the update packet to all neighbouring nodes. If the node does not hold an agent it forwards the agent to the neighbour that is closest to the home location. If the packet reaches the home location without encountering a node with an agent on, then the nodes closest to the HL broadcasts the update.

In addition, to this first node broadcasting the packet, each subsequent node that also has an agent on also rebroadcasts the packet; however, this only takes place if the update packet contains newer information than the agent. This condition causes the broadcast 'storm' to be self limiting. If there are six agents then there will be a maximum of six broadcasts, or sever in the case where the update reaches the node closest to the HL.

The function RecvUpdatePacket() shows the pseudo code for what actions a node takes upon receiving the update packet. Initially it determines if the node holds an agent (*a*) with the same owner as the update packet (*u*). If the node does, and the agent has older (*a.t*) data than the update packet (*u.t*) then it updates the packet and broadcasts the update to all neighbours. In all other cases, the packet is forwarded to a neighbour that is closest to the home location.

```
RECVUPDATEPACKET($u$)
1   a ← GETAGENT($u_{ID}$)
2   if a is not null and u.t > a.t then
3       UPDATEAGENT($a$, $u$)
4       BROADCAST($u$)
5   else if $u$ was unicasted to this node then
6       ROUTETOHL(u)
```

The algorithm does not guarantee that all agents will be updated as we explore shortly.

Location update performance

To evaluate the performance of the update scheme we measure how many of the agents in a GLS are successfully updated. The simulation is performed as per the parameters in the last section, with each node sending an update to its GLS every 10 seconds.

Figure 6 illustrates the percentage of agents updated on average. This is plotted against node density as this has the most significant impact on update completeness. Node density is described as the number of neighbours in a node's transmission radius plus one for that node. Higher node densities ensure greater network connectivity.

Low node densities result in poor performance as the network is in a disconnected state but once the density is increased to near 100% connectivity then all the agents on average are updated. It is worth noting that speed seems to have negligible effect on these results.

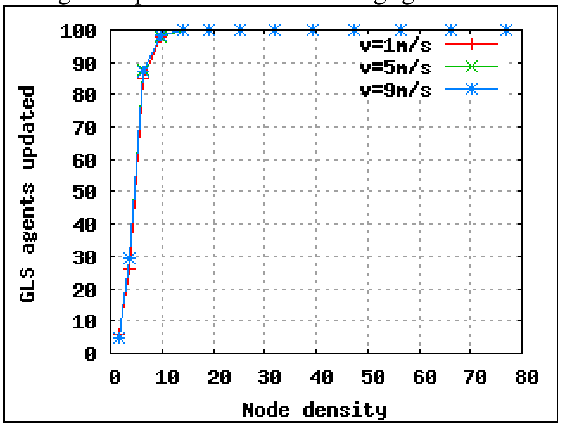

Figure 6: Update completeness against node density

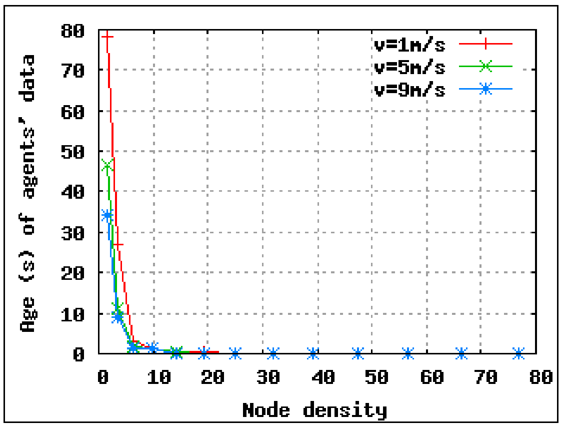

Figure 7: Average age of data in seconds of those agents which were not updated

Figure 7 illustrates the age of data held on agents that were not updated. For example, if an agent was on a permanently disconnected node then its age would increase at every update interval. Samples are taken every update interval and the averages plotted on the figure.

Interestingly, speed does have a slight affect in this case as high mobility. A moving node can become disconnected from the network but for those with high mobility the disconnected period is shorter.

Querying the Location Server

Querying of the location server is also performed in a self organising fashion. Firstly, the querying node creates a query packet and forwards it towards the neighbour closest to the home location. If this neighbour has an agent for that GLS on then the agent creates a reply and it is sent back to the querying node. If the node does not have an agent, then it forwards the packet to the next closest node where the cycle repeats.

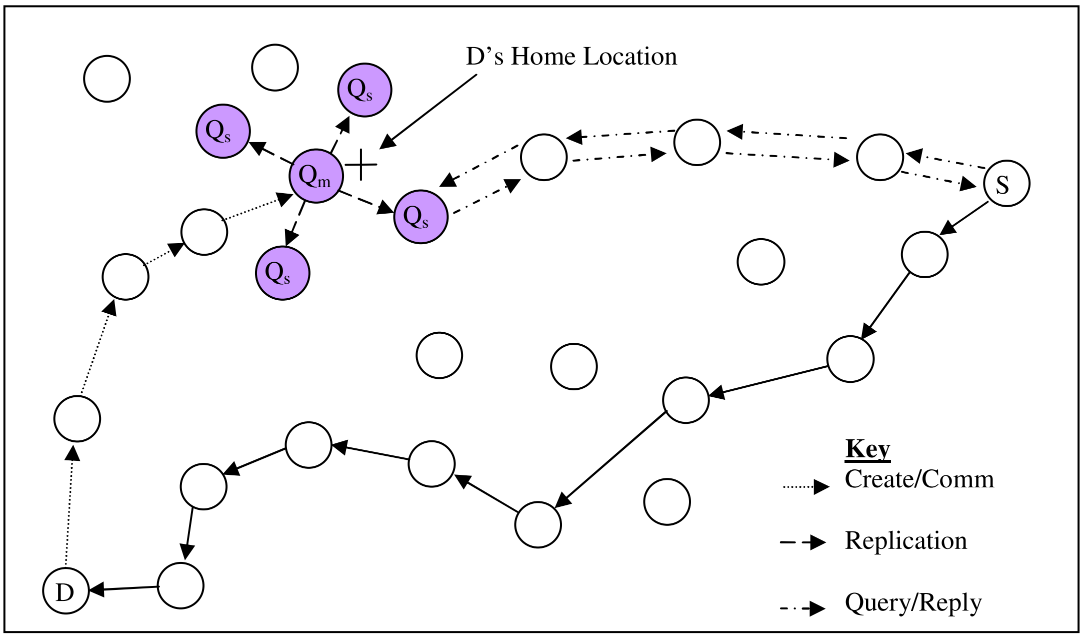

Figure 8: Using SOLS as a location server

If the packet reaches the closest node to the home location (after perimeter routing has been used where necessary) and that node does not have an agent, then the node broadcasts the query packet to all the neighbouring nodes. This ensures that if due to node mobility the agents are on neighbouring nodes then they still receive the query. This procedure is shown as pseudo code in the ReceiveQueryPacket(p) function.

```
RECEIVEQUERYPACKET(P)
1  $a \leftarrow$ GETAGENT($p.n_{ID}$)
2  if a is not null then
3        ROUTETOHL($p$)
4  Else
5        INITIATEQUERYREPLY($a$)
```

Tolerance of Voids

In terms of discovery, SOLS is resistant against areas void of nodes due to its use of a geographical point rather than a region. If the HL is inside a void then its GLS attaches itself to the perimeter of the void and therefore a form of perimeter routing can easily discovery it; however, SOLS exhibits behaviour in the presence of convex voids that is undesirable.

Examine the example in Figure 9 showing the master on the closest node to the home location and it replicating to neighbouring nodes (1). These neighbours will be further away, but if the void is convex then they will possibly have other neighbours which are closer to the home location that are not the node which the master is on. The slaves mark these as eligible migration targets and migrate there (2). These agents then find themselves on the closest node to the home location (or so they believe) and elect themselves as masters. At this point they then begin a replication phase which results in excessive numbers of agents per GLS. As a result, an update is much more likely to be incomplete.

Figure 10 shows a simulation snapshot of this scenario occurring. Each node is represented by a dot with the nodes in the centre of the circles holing masters. All other non-black dots are nodes with slaves.

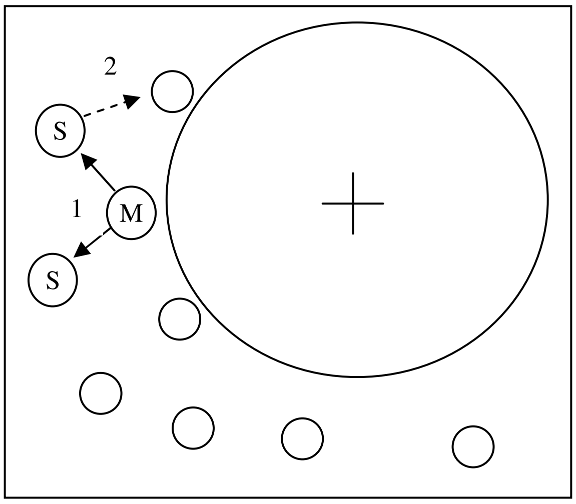

Figure 9: Convex void problem illustration

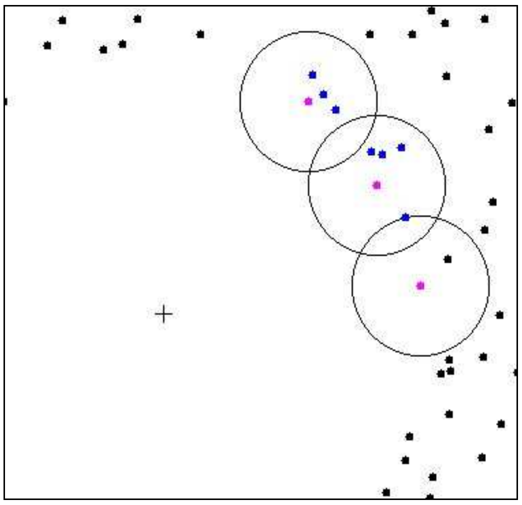
Figure 10: Convex void problem simulation snapshot

The solution to this problem is to change the slaves' home location information. If a master replicates then all its slaves should be told that their home location is the current location of the master. When replicating they then attempt to stay near the master, but if the void was to disappear then the master would quickly migrate to the home location. At this stage, or if the master was lost then as an agent migrates to the closest node to the location where the master was it will reset its target location to that of the home location. We were unable to replicate the above undesirable behaviour once this modification was made.

5. PERFORMANCE COMPARISON

In this section we describe the performance comparison of the SOLS approach against that of Terminodes in terms of delivery success and packet overhead. Due to the variability of ad hoc networks we repeat the simulation under a wide variety of parameters.

The Terminodes home-region work does not describe implementation details although its concept is the nearest and most comparable to our work. Therefore, we derive an implementation from their concept and describe it here. To create a Terminode location servers, a node creates a packet and sends it to the home region. As soon as the packet reaches a node inside the home region that node broadcasts it, and each neighbour rebroadcasts it further until it has been delivered to all nodes in the home region. The broadcast is a self-limiting approach as described earlier in the paper where a node only rebroadcasts the packet if it holds an older version of the location data. To update the Terminode server then the same approach as creating one is used. Finally, to query it the same mechanism as described in this paper for SOLS is used. We believe this provides an optimal implementation of the Terminode concept and now use it for comparison against SOLS.

The simulation is configured as before with the addition of the query and update mechanisms described. Each of the fifty nodes with GLSes picks another node and sends a constant bit-rate stream of packets to it. The SOLS algorithm intercepts packets for which the location is not known and initiates the query mechanism. To avoid location queries for every packet we also implement a location caching service on each node. When a location reply is received, the information is cached for a period equal to the update interval (*U*) at that node.

Bidirectional flows (allocated randomly)	5
Terminode *R* value	100m
Location Server Update interval	10 seconds

Initially, we examine the performance of the two algorithms in a low mobility scenario with high update frequency, whilst varying the failure probability of nodes (Figure 11). SOLS outperforms Terminode in delivery success by just under 10% for all realistic failure probabilities. At extreme failure rates Terminodes excels and this is because each update packet recreates the servers on all participating nodes.

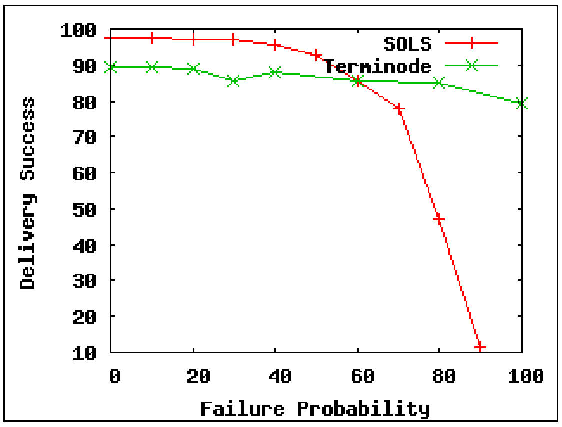

Figure 11: Delivery success (v=1m/s, u=10s)

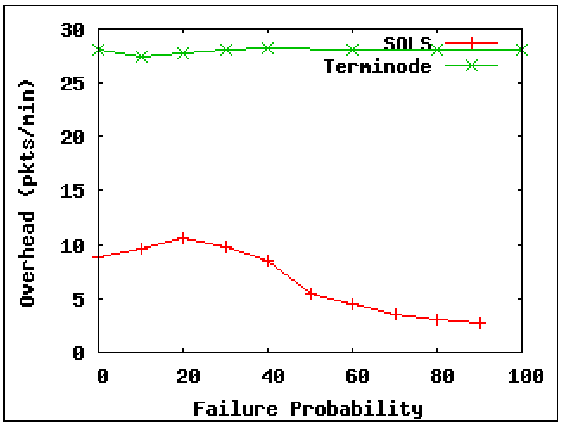

Figure 12: Overhead (v=1m/s, u=10s)

Examining overhead in packets per minute of the two schemes (Figure 12), SOLS again outperforms Terminode, this time by a factor of almost three. This is because Terminodes recreates the server for each update where as SOLS maintains it. SOLS performance drops off as node failures rates become high as this is a result of the loss of some the GLSes.

Figure 13 illustrates delivery success against update interval with moderate speed (5m/s) and high node failure (20%). SOLS significantly outperforms Terminodes and this is for several reasons. Firstly, Terminodes does not mitigate either mobility or node failure, and when updates become infrequent the Terminode location servers quickly become inaccessible. SOLS mitigates both of these and remains accessible throughout.

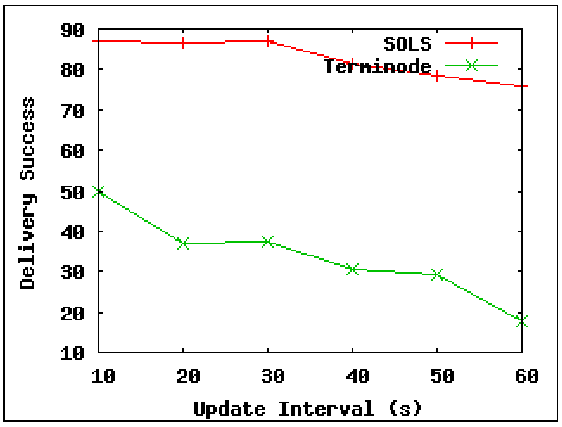

Figure 13: Delivery Success (v=5m/s, f=20%)

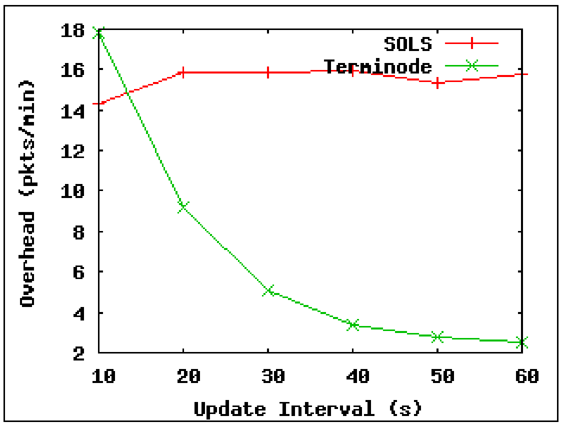

Figure 14: Overhead (v=5m/s, f=20%)

Figure 14 shows the overhead of the two techniques for the above scenario. As expected Terminode's overhead decreases with increased update interval as there is no maintenance overhead. Meanwhile, SOLS' overhead remains approximately constant as there is no variance in speed or failure probability which would cause an increase in maintenance overhead.

We now compare the two techniques' performance when speed is varied. Figure 15 shows the delivery success of the two techniques with SOLS again significantly outperforming Terminodes. As speed increases Terminode's performance quickly drops off due to its lack of mobility mitigation. SOLS decreases slightly but this is attributable to inaccurate location information stored in the GLSes (updates and queries are too infrequent) rather than a failure of the algorithm. Mobility prediction techniques could increase the performance (e.g. (Dongjin et al., 2004)) further.

Our justification for comparing our approach against Terminodes is that along with Terminodes the overhead of various functions is not related to network size. We compare the overhead of our approach against other methods in Table 2 below.

Table 2: Overhead comparison of differing location servers

	Creation	**Maintenance**	**Update**	**Query**	**Failure tolerance**	**Hash function**
SOLS	*O(1)*	*O(v)*	*O(1)*	*O(1)*	High	Yes
Terminode	*O(1)*	*n/a*	*O(1)*	*O(1)*	Low	Yes
Horiz/Vert quorum	$O(\sqrt{n})$	n/a	$O(\sqrt{n})$	$O(\sqrt{n})$	Low	No
Reactive Location Server	$O(n^2)$	n/a	$O(n^2)$	*O(1)*	Medium	No
Grid Location Service	$O(\log n)$	n/a	$O(\log n)$	$O(\log n)$	Low	No

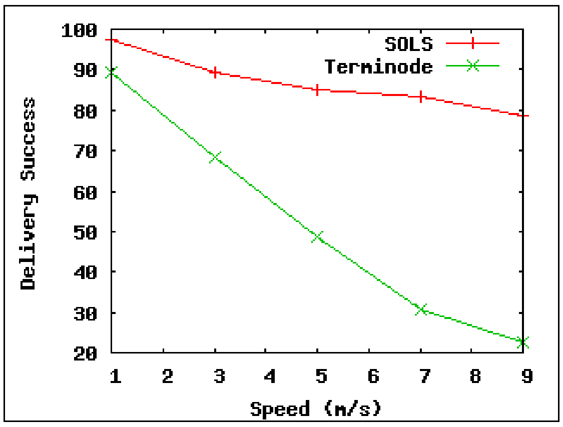

Figure 15: Delivery Success (f=0%, U=10s)

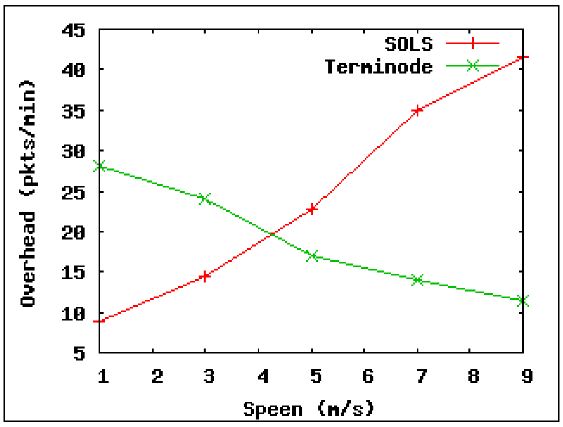

Figure 16: Overhead (f=0%, U=10s)

Figure 16 shows the overhead for the last scenario where SOLS starts out with the least overhead and Terminodes the most. The overhead of Terminodes drops due to nodes being unable to find the location server they wish to query. SOLS' overhead increases as it tries to mitigate the increased mobility of nodes, demonstrating the adaptability to changing conditions.

Finally, we examine how the two approaches perform in the face of voids in the network. To simulate a void we create an area in the middle of the network in which nodes are not permitted to travel within as illustrated in Figure 17. This area is of equal height and width (*a*) and is of equal distance from all sides of the simulation area (*b*).

Terminodes uses a region to store location information, but when this region is void (or low) of nodes it is unable to store the information. As SOLS uses the idea of a geographical location which result in the GLS being on the perimeter of the void, perimeter routing quickly finds the GLS.

Figure 18 shows the delivery success of the two schemes given a varying void placed in the centre of the network. SOLS remains roughly constant in its delivery success whilst Terminodes quickly drops off as more and more regions are enclosed in the void.

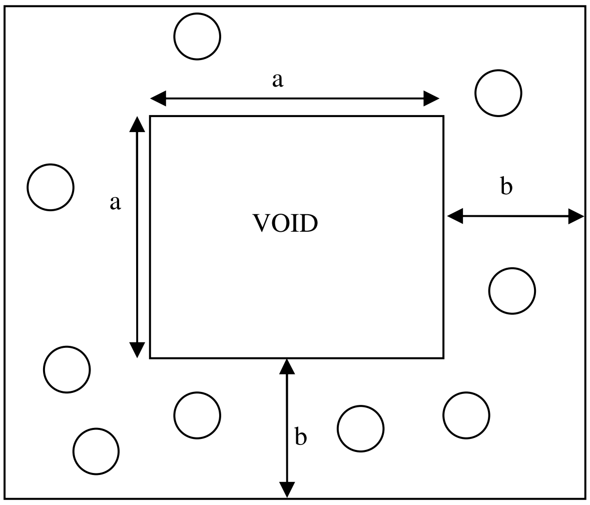

Figure 17: Simulation configuration for examining voids

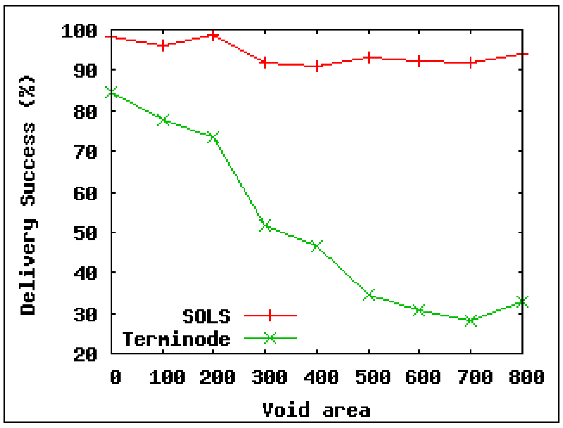

Figure 18: Handling voids

6. Conclusion

In conclusion, SOLS demonstrates a self organising location server that is highly adaptable to changing conditions. This adaptively minimises the overhead on the network whilst maximising delivery success. SOLS outperformed the existing Terminodes approach significantly in all realistic scenarios, whilst often showing much lower overhead.

We recommend more attention be paid to exploring self organising techniques for such adaptive behaviour in ad hoc networks. Ad hoc networks are highly unpredictable and a computationally simple technique has been shown to mitigate the main problems encountered in this case.

7. References

BARR, R. (2004) An efficient, unifying approach to simulation using virtual machines., Cornell University.

DONGJIN, S., HELMY, A. & KRISHNAMACHARI, B. (2004) The effect of mobility-induced location errors on geographic routing in ad hoc networks: analysis and improvement using mobility prediction. *Wireless Communications and Networking Conference 2004. IEEE.*

EBERHART, R. C., SHI, Y. & KENNEDY, J. (2002) *Swarm Intelligence*, Morgan Kaufmann.

FEHNKER, A. & GAO, P. (2006) Formal Verification and Simulation for Performance Analysis for Probabilistic Broadcast Protocols. *Lecture Notes in Computer Science*, 128-141.

FUßLER, H., WIDMER, J., KASEMANN, M., MAUVE, M. & HARTENSTEIN, H. (2001) Beaconless Position-Based Routing for Mobile Ad-Hoc Networks. REIHE INFORMATIK.

HUBAUX, J. P., LE BOUDEC, J. Y., GIORDANO, S., HAMDI, M., BLAZEVIC, L., BUTTYAN, L. & VOJNOVIC, M. (2000) Towards Mobile Ad-hoc WANs: Terminodes. *Wireless communications and networking conference.* Chicago, IL, Ieee.

IEEE (2008) 802.11n: Report (Status of Project).

JOHNSON, D., PERKINS, C. & ARKKO, J. (2002) RFC 3775: IP Mobility Support for IPv6. Internet Engineering Task Force.

KARP, B. (2000) Geographic Routing for Wireless Networks. *The Division of Engineering and Applied Sciences.* Cambridge, MA, Harvard.

KURKOWSKI, S., CAMP, T. & COAGROSSO, M. (2005) MANET Simulation Studies: The Incredibles. *ACM SIGMOBILE Mobile Computing and Communications Review,* 9.

LIU, D., STOJMENOVIC, I. & JIA, X. (2006) A Scalable Quorum Based Location Service in Ad Hoc and Sensor Networks. *IEEE International Conference on Mobile Ad-hoc and Sensor Systems MASS.* Vancouver.

LOCHERT, C., HARTENSTEIN, H., TIAN, J., FUßLER, H., HERMANN, D. & MAUVE, M. (2003) A routing strategy for vehicular ad hoc networks in city environments. *IEEE Intelligent Vehicles Symposium*, 156-161.

MICHALSON, W. R. & AHLEHAGH, H. (2004) A 3D Location Discovery Algorithm for Ad Hoc Networks. IN ARABNIA, H. R., YANG, L. T. & YEH, C. H. (Eds.) *Wireless networks; ICWN '04.* Las Vegas, NV, CSREA Press.

RYU, J. P., KIM, M. S., HWANG, S. H. & HAN, K. J. (2004) An Adaptive Probabilistic Broadcast Scheme for Ad-Hoc Networks. IN MAMMERI, Z. & LORENZ, P. (Eds.) *High speed networks and multimedia communications: 7th IEEE International Conference, HSNMC 2004, Toulouse, France, June 30-July 2, 2004, proceedings /.* Toulouse, France, Berlin.

SIVAVAKEESAR, S. & PAVLOU, G. (2004) Cluster-based Location-Services for Scalable Ad Hoc Network Routing. IN BELDING-ROYER, E. M., AL AGHA, K. & PUJOLLE, G. (Eds.) *IFIP TC6/WG6.8 Conference on Mobile and Wireless Communication Networks; (MWCN 2004).* Paris, France, New York.

AUTHORS

Dr Gareth Owen, University of Kent
Dr Owen received his BSc in Internet Technologies in 2004 and his Ph D (Computer Science) in 2007. He is a lecturer at the University of Kent and has published a number of papers in international conferences and journals. His primary research interests are in ad hoc networks and wireless mesh networks.

Dr Mo Adda, University of Portsmouth
Dr Adda holds a PhD (Computer Science) and is currently a principal lecturer at the University of Portsmouth. He an has published a number of papers in international conferences and journals. His primary research interests are computer networks and agent systems.