# SELF-ORGANISING QUORUM SYSTEMS FOR AD HOC NETWORKS

Gareth Owen and Mo Adda
School of Computing, University of Portsmouth
Buckingham Building, Lion Terrace, Portsmouth
England, PO1 3HE
{gareth.owen, mo.adda}@port.ac.uk

**ABSTRACT**
There are many essential applications for quorum systems in ad-hoc networks, such as that of location servers in large-scale networks. Existing research proposes many approaches to the problems, many of which are incomplete, cumbersome, or incur significant cost. We describe and analyse a self-organising quorum system that creates an emergent intelligence to minimise overhead and maximise survivability. We then examine the quorum's performance as a location server and suggest improvements to the query mechanism and routing algorithm using the information.

**KEY WORDS**
Ad-hoc networks, quorum systems, self-organising systems.

## 1. Introduction

Ad-hoc networks are a means of networking computing devices together without requiring any setup or existing infrastructure. We define an ad hoc network as a graph of a set of nodes, $V$, and a set of communication paths, $E$, that vary through time as nodes move and fail.

$$G_t = (V_t, E_t) \quad (1)$$

Multi-hop ad-hoc networks allow nodes to communicate with each other without being in transmission range by relaying their messages through intermediate nodes. Routing in small networks is usually achieved through an optimal broadcast mechanism; however, when the network becomes large (>200 nodes), this form of discovery becomes extremely expensive.

Routing in large-scale ad hoc networks can be achieved through using location information from a GPS device. Hou [1] describes Most-Forward-within-Radius (MFR) whereby nodes are aware of their geographical position (from GPS) and progressively route packets closer to the location of the destination. Before a node can route packets in this fashion, it will need to know an accurate location for the destination. Many routing protocols propose a function that maps a node's ID to its location, but whilst this is ideal when networks are static and such a function can be defined, it is not when nodes are mobile and networks span many kilometres. Location servers solve this problem by maintaining location information of a node that sends it frequent updates.

Several types of location server have been proposed to provide this information. Li [2] divides the area into grid-squares of different orders. Starting with the grid-square that the node currently occupies, an order-one square, an order-two square is the grid-square containing four of the order-one squares. Li proposes that a number of nodes in each order-n square should host the location information. Therefore, information about a node's location becomes less densely available the further one is from its location.

Giordano proposed [3] the most promising approach to location servers by allocating nodes within a specific radius of a geographical point to serve as a location server. Their work suggested modifying the radius depending upon the node density so that a minimum number of nodes participate; however, they did not address issues such as node mobility, fault tolerance and query success.

Finally, Stojmenovic proposed [4] that all nodes to the north and south in a fixed-width column assume the role of a location server. Location searches are performed by routing packets horizontally so that they will eventually intersect the vertical column. This approach is not feasible for large-scale ad hoc networks because of the large number of nodes. For a more complete review of approaches to location servers, the reader is referred to [5].

The approach shown in this paper is most similar to Giordano's work on home-regions but we further develop the quorum system we proposed in [6] by looking at alternative methods of replication and analysing its performance as a location server. Methods of querying the quorum and ways of mitigating the problem of incomplete quorum updates, along with a method to improve routing success in the face of out-dated information are also addressed.

## 2. Quorum systems in ad hoc networks

In this paper, we propose an approach that is similar to that of the home region but that addresses many of the problems associated with it. Instead of allocating a circular region of nodes as the hosts of a location server, we say that nodes directly adjacent to a particular geographic point should be responsible. This permits a small number of nodes to be used without knowledge of node density.

A node sets up a location server by sending an initiation packet toward its home location that will be received by the node closest to the home location. This node will then assume the role of master of the server, and as it moves the role is transferred to a node that is closer to the home location. Migration is the term we call this process of quorum components moving from node-to-node to remain close to a geographic point.

Presently, with only a master, failure of the node hosting it could result in failure of the server so it is important that the data is replicated. Immediate neighbours of the master are sent a duplication packet from the master and assume the role of a slave; however, the server as a whole needs to make sure that enough replicated copies are kept in the event of node failures so that the data is not lost.

Formally, we define our quorum or location server as a set $Q_t$ where $M_t$ and $S_t$ are the sets of nodes that host master and slave components respectively:

$$Q_t \subseteq V_t = M_t \cup S_t \tag{2}$$

Such maintenance of what is essentially a quorum [7] is very costly process in wired networks let alone wireless, and so a novel approach is needed.

We draw our inspiration from ants in a colony who find their way to food by laying pheromones for each other to follow [8]. These pheromones serve to modify the environment so that other ants may detect these modifications avoiding the need for directed communication. Drawing upon this to develop a location server, we suggest that instead of components of a quorum communicating with each other they modify their environment. As every node has to beacon its location to its neighbours for routing to occur, we add to this beacon packet (which we call the environment) an itinerary of location server components held at that particular node. A node's itinerary, $I_{v,t}$, is simply a list of identifiers for quorums which it hosts a component of. Therefore, a beacon frame can be described as the n-tuple:

$$B_{v,t} = \{v_x, v_y, I_{v,t}\} \tag{3}$$

Each node maintains a list of all received beacons so that each component of the quorum is able to examine a list of which nodes hold which components. Beacons stored in the list expire after twice the beacon interval.

The master is the only node permitted to duplicate as it is most likely to be in the center of the quorum and so able to correctly count the number of components. When the number of neighbours hosting components falls below a *threshold*, the master will begin a duplication phase to raise this number.

We now have a set of quorum components who do not directly communicate with each other, but who act in a self-organising manner forming an emergent intelligence (like ants). Each attempts to get as close as possible to the home location by moving from node to node (except where a destination node is already in possession of one). Each component assumes the role of master and responsibility for duplicating if it finds itself, through observing its environment, to be on the closest node to the home location. If it realises it is no longer the closest then it will automatically remove its responsibility and assume the role of a slave. We illustrate the master and slaves, along with migration in Figure 1, where the master is held on the closest node to the point and has duplicated its data to several slaves on adjacent nodes.

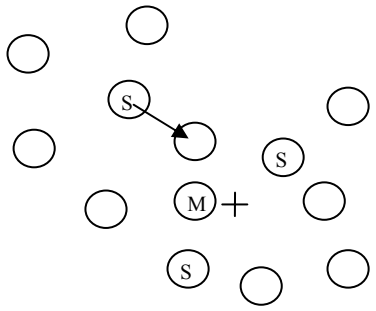


*Figure 1: Illustration of quorum and component migration*

In addition to the duplication task, the components perform one more tasked based upon information gathered from the beacons. The components self delete the quorum to stop a large number of replicated components consuming resources. They do this by each monitoring the number of replicas, and if they rise above a *threshold* then that component will delete itself.

This process is performed on each node at regular intervals for each component held, as described by the ManageComponent($c$) function:

```
MANAGECOMPONENT(c)
1    n ← closerNeighbor(homeLocation)
2    if exists(n) and hasComponent(n)
3          then c_role ← slave
4          else c_role ← master
5    if exists(n) and !hasComponent(n)
6          then c.migrate(n)
```

```
7    else if c_role=master and
                    hasComponent(all) < threshold
8        then replicate(c)
9    else if c_role=slave and
                    hasComponent(all) > threshold
10       then delete(c)
```

The Replicate(*c*) function has several definitions representing several variations of the algorithm. The unicast variant sends a replica to the first node in its neighbour database without a component already. It is expected this approach will not survive well due to the period it would take to replace a number of components.

```
REPLICATE(C)  - UNICAST
1    for all n in neighbors
2        if not hasComponent(n) then
3            unicast(c)
4            exit forloop
5        endif
6    end for
```

The broadcast variant sends replicas to all neighbours in the vicinity and is expected to survive better due to it being able to recover from a number of lost components with just one packet.

```
REPLICATE(C)  - BROADCAST
1    broadcast(c)
```

The broadcast algorithm should perform better in the face of high failure rates; however, if just one component is lost from the quorum then a broadcast could increase the number of components significantly over the *threshold*, which would then require pruning causing an expand and collapse scenario to occur. Therefore, we also produce a hybrid approach that uses broadcast when the number of components is low and unicast when it is nearer the *threshold*.

```
REPLICATE(C)  - HYBRID
1    if hasComponent(all) > 50% of threshold
2        then unicast(c)
3        else broadcast(c)
```

Finally, we also examine using multicast for replication of the data. When a node senses that there is less than the *threshold* of slaves, then it will create a multicast packet to a number of neighbours defined by the *threshold* minus the number of slaves. The neighbours chosen will be those who do not all ready have a component.

```
REPLICATE(C)  - MULTICAST
1    needed ← threshold – hasComponent(all)
2    destcount ← 0
3    while needed > 0 and neigh not equal null
4        if not hasComponent(neigh) then
5            c_dests[destcount++] = neigh
6            needed--
7        endif
8        neigh ← neigh_next
9    endwhile
```

We can define the setup cost, $C_{Setup}$, as the number of hops to the home location, $h$, plus one for a broadcast duplication packet (eq. 4) or *threshold* packets for the unicast (eq. 5). The maintenance cost of the quorum, $C_{Maintenance}$, is defined (eq. 6) as the number of migrations of components due to node mobility ($m$) plus the number of replication packets ($r$) to maintain the *threshold*.

$$C_{Setup} = h + 1 \quad (4)$$

$$C_{Setup} = h + threshold \quad (5)$$

$$C_{Maintenance} = m + r \quad (6)$$

Keeping with the simplicity of the current algorithm, we follow the same philosophy here to develop a quorum query mechanism. When a packet's destination location is not known, it is routed towards the home location and upon encountering a node containing a quorum component for the destination, it updates the packet ($p$) with the node location and routes it towards. We describe this in the RoutePacket($p$) function below.

```
ROUTEPACKET(p)
1    c ← component(p_dest)
2    if exists(c)
3        then p_location ← c_location
4    if exists(p_location)
5        then route to closestNeighbor(p_location)
6    if not exists(p_location)
7        then route to
                closestNeighbor(homeLocation(p_dest))
```

### 2.1 Query and update mechanisms

The mechanism to update the quorum is equally simple and based upon a zero knowledge approach to minimise overhead. When receiving a new update packet, a node simply updates itself if it holds a component and then rebroadcasts it. This way, each element of a quorum should receive a copy and the number of overhead packets will be equal to the number of components present ($q_{component}$) as shown in equation 7.

$$C_{update} = h + q_{components} \quad (7)$$

This approach does not guarantee a complete quorum update however due to its self-organising approach. Therefore, we examine the age of the data returned and propose a technique that we call Multi-query (MQuery) to try to reduce the likelihood of retrieving old data. MQuery requires every quorum query to attempt to query at least two quorum components and take the most up-to-date result. Simply, if a query is received first by a component, then the packet will be routed to the next

closer node to the home location. Hopefully, this node will also be a participant in the quorum and therefore could have more correct data. In our results, we examine the effect this has on the age of data.

RECVUPDATEPACKET($p$)
1 $c \leftarrow$ Component($p_{nodeID}$)
2 **if not** exist($c$) **or** seenBefore($p$) **then** ignore
3 **if** $p_{revision} \leq c_{revision}$ **then** ignore
4 $c_{data} \leftarrow p_{data}$
5 broadcast($p$)

Quorum systems are traditionally analysed based upon their performance in terms of load [9], fault-tolerance [10] and failure probability [11]. Therefore, we measure the survivability or failure probability of the quorum over 30-minutes and the overhead in packets per minute of one quorum.

In addition to analysing the performance as a quorum, we need to look at the ability of this approach to act as a location server for large-scale ad hoc networks. The metrics we will measure are update cost, query success, age of results and routing success.

### 2.2 DSR-aided MFR to improve routing success

It is possible that the location information received could be out-of-date and so we analyse the effect this has on a routing and ways to improve it. For comparison, we use the MFR approach used in RoutePacket($p$) and compare it with the technique proposed here. It is worth noting that several other techniques have been examined to improve routing success (recursive search [12], GRA [13], and a planar sub-graph [14]); however, these do not examine success in the face of incorrect location information. Here, we propose a technique to handle not only small voids but also out-of-date location information.

We also propose that if a packet reaches a node in which the MFR algorithm is unable to recover, then the node initiates a DSR [15] search (with limited TTL) to find a node that is closer. Upon finding one, the packet is source routed there and then resumes using MFR to continue to the destination. If the DSR discovered path becomes invalid before the packet is routed completely, then the node will reinitiate discovery. This is described in the DsrMfrRoutePacket function below.

DSRMFRROUTEPACKET($P$)
1 **if not** routePacket($p$) **then**
2 **if** routes.exist($p_{location}$) **then**
3 use existing route
4 **else**
5 initiateDSRdiscovery($p_{location}$, $p_{dest}$)
6 queuePacket($p$)
**endif**

A density of 200 nodes per square kilometre with 140.5m TX range almost guarantees connectivity. Therefore, we set the TTL of the DSR search to three hops as we expect that routing failures will be due to a locally poor node arrangement rather than large voids. The TTL could be varied depending on the node density but we do not examine this here.

## 3. Methodology

Simulation is widely accepted as a means for analysing ad hoc networking protocols due to the mathematical complexity of the scenario. Here we use the GloMoSim (v2.03) simulator [16] that provides models of all the layers experienced in a real experiment. We configure the simulator with the parameters shown in Figure 2 to represent that of a kilometre-squaed section of a large-scale ad-hoc network.

*Figure 2: Simulation parameters*

| Parameter | Value | Parameter | Value |
|---|---|---|---|
| **Terrain** | 1000x1000 | **Number of nodes** | 200 |
| **Propagation Model** | Two-ray | **Number of quorums** | 50 |
| **Tx/Rx Range** | 140.5m | ***threshold* (number of slaves)** | 5 |
| **Mobility Model** | Random Waypoint $v_{min}$ = 0.1m/s $v_{max}$ = 3.0m/s | **Beacon rate** | Every 5 seconds + jitter |
| **Simulation-time** | 30 minutes | **Manage Component call rate** | Every 6 seconds + jitter |

## 4. Results

Firstly, we examine (in Figure 3) the ability of the algorithm to survive over a 30-minute simulation given various node failure rates.

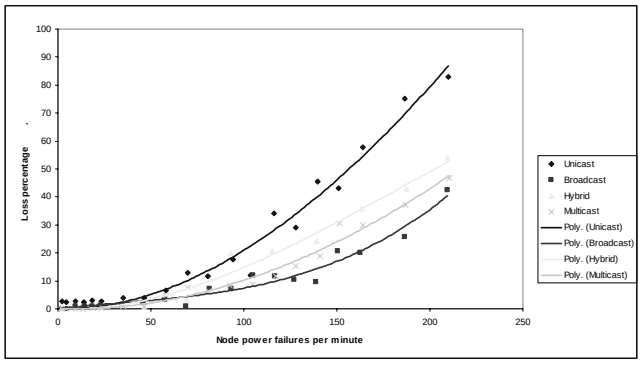


*Figure 3: Survivability*

All four versions of the algorithm survive exceptionally well even given the exceptionally high failure rates, with up to 20% of the quorums surviving.

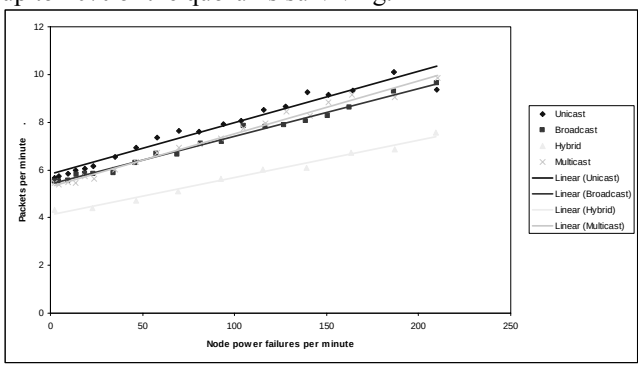


*Figure 4: Overhead (packets/min)*

The overhead (Figure 4) is less than 10 packets per minute despite node mobility and node failures. Interestingly, the hybrid version of the algorithm incurs less overhead and we explain this as due to the reduction of several duplication packets into one when near the *threshold*, and the avoidance of the compaction and expansion problem explained earlier. Perhaps even more interesting is that the multicast version has higher overhead than the hybrid given that one would expect this to provide the ideal solution. This is explained as due to the hybrid algorithm maintaining an optimum between replication and quorum size.

Figure 5 shows the overhead in terms of memory used per node. Fifty quorums are distributed across the network and each node incurs a memory overhead of approximately 6.5 to 9.0 components per node given zero node failures.

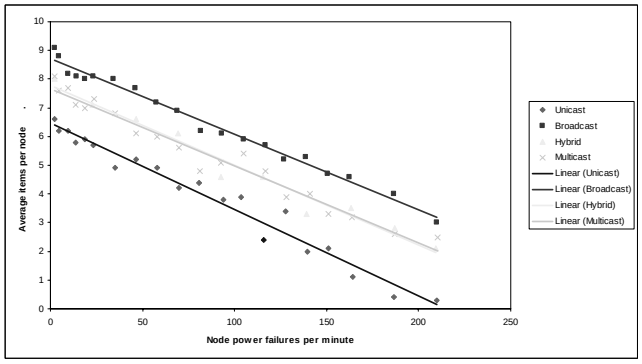


*Figure 5: Memory overhead (components/node)*

Figure 6 shows the number of queries returned by the quorum that were out-of-date due to an incomplete update of all components. We then also show how this can be improved by querying at least two components and taking the most up-to-date result. This simple technique halves the number of old-results returned.

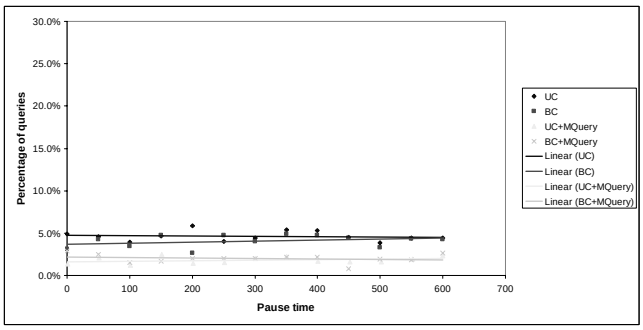


*Figure 6: Out-of-date results returned*

The overhead incurred to fully update the quorum depends on the number of components present. Figure 7 clearly shows the unicast algorithm maintains the lower number of components while the broadcast version maintains almost twice as many.

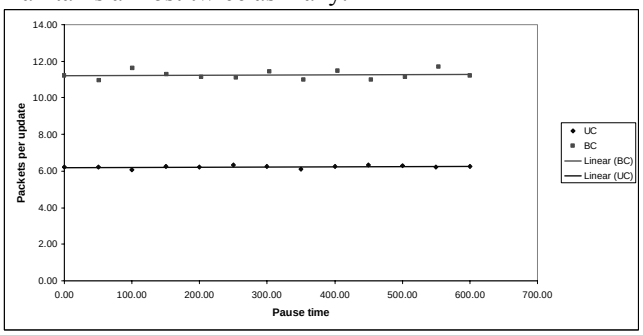


*Figure 7: Update cost (packets)*

Figure 8 shows the results obtained when the information from the quorum is out-of-date by varying degrees. With update intervals of two minutes, almost a third of packets do not reach their destination. However, when we use the DSR-aided MFR routing algorithm, less than 3% of packets fail to reach their destination.

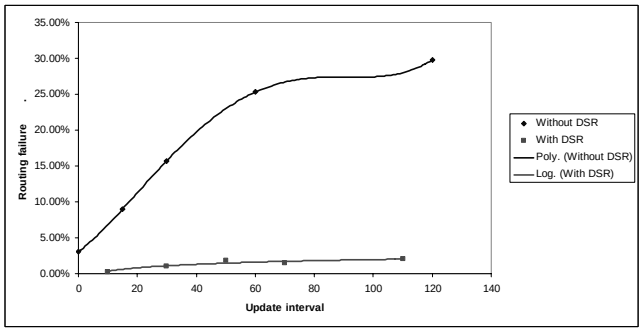


*Figure 8: DSR-aided geodesic routing*

## 5. Conclusion

Existing algorithms are not comparable with the approaches we have presented here, as they do not address node mobility and failure that are the key motivation to our work. Therefore, we present our results without comparison with them.

The results show that the broadcast algorithm performs better in terms of survivability while the hybrid provides

lower overhead. Maximising survivability is the main priority and all the algorithms cope with failure rates significantly higher than expected. Therefore, we recommend the use of the hybrid algorithm due to the avoidance of the expand-collapse problem and the ability to recover quicker than the unicast approach to a number of node failures.

In terms of ability to query the quorum, the approach used to maintain the quorum had little effect of the results. We found that approximately 4% of queries returned out of date results, but that this is halved by querying at least two components. Further improvements could be made if elements of the quorum communicate with each other, or revision information is added to the beacon packet.

We showed that by aiding the routing algorithm with DSR when the packet was unable to make any further progress forward, with a search radius of just three hops, increased the routing success by up to a factor of 10 even when the location information is up to two minutes old.

This work demonstrates that it is possible to implement reliable quorum systems in ad-hoc networks with negligible overhead. The technique used is a self-organising system relying on an intelligent behaviour emerging from the individual actions of each component. We demonstrated the performance of the quorum for obtaining location information and measured the percentage of out-of-date data returned. Location information does not necessarily have to be up-to-date as long as the node has not moved too far and the routing algorithm is able to recover. One way to improve geodesic routing algorithms to cope with incorrect information would be to aid the algorithm with a limited DSR search when the node is not at the given location. Undertaking this simple improvement increased the routing success by up to a factor of ten; however, this will of course incur a higher routing overhead.

## References


[1] T.-C. Hou & V. Li, Transmission range control i multihop packet radio networks, *IEEE Transactions on Communications, 34* (1), 1986, 38-44.

[2] J. Li, J. Jannotti, D. S. J. De Couto, D. R. Karger & R. Morris, A Scalable Location Service for Geographic Ad Hoc Routing. *Proceedings of 6th ACM International Conference on Mobile Computing and Networking (MobiCom)*, 2000.

[3] S. Giordano & M. Hamdi Mobility Management: The Virtual Home Region, EPFL, Lausanne, Switzerland(1999).

[4] I. Stojmenovic A scalable quorum based location update scheme for routing in ad hoc wireless networks, University of Ottawa(1999).

[5] T. Camp, J. Boleng & L. Wilcox, Location Information Services in Mobile Ad Hoc Networks. *Proceedings of IEEE International Conference on Communications*, 2002, 3318-3324.

[6] G. H. Owen & M. Adda, Quorum based geographically static data storage in ad-hoc networks. *Proceedings of International Network Conference*, Plymouth, UK, 2006.

[7] D. Malkhi, M. K. Reiter, A. Wool & N. Wright, Probabilistic Quorum Systems. *Proceedings of Symposium on Principles of Distributed Computing*, 1997, 267-273.

[8] S. Johnson, *Emergence: The Connected Lives of Ants, Brains, Cities and Software* (Scribner 2001).

[9] M. Naor & A. Wool, The load, capacity and availability of quorum systems., *SIAM Journal of Computing, 27* (2), 1998, 423-447.

[10] D. Barbara & H. Garcia-Molina, The vulnerability of vote assignments, *ACM Transactions on Computer Systems, 4* (3), 1986, 187-213.

[11] D. Peleg & A. Wool, The availability of quorum systems, *Information and Computation, 123* (2), 1995, 210-233.

[12] G. Finn Routing and addressing problems in large metropolitan-scale internetworks, USC/Information Sciences Institute(1987).

[13] R. Jain, A. Puri & R. Sengupta, Geographical Routing Using Partial Information for Wireless Ad Hoc Networks, *Ieee Personal Communications, 8* (1), 2001, 48-57.

[14] B. Karp & H. T. Kung, GPSR: Greedy Perimeter Stateless Routing for Wireless Networks. *Proceedings of Mobile computing and networking; MobiCom 2000*, Boston, MA, 2000, 243-254.

[15] D. Johnson & D. Maltz, Dynamic source routing in wireless ad-hoc networks.*in*. Mobile Computing. T. Imelin-sky&H. Korth, Kluwer Academic Publishers, 1996 153-181.

[16] X. Zeng, R. Bagrodia & M. Gerla, GloMoSim: a library for parallel simulation of large-scale wireless networks. *Proceedings of 12th Workshop of Parallell and Distributed Simulations - PADS '98*, 1998.