Simulation of quorum systems in ad-hoc networks

Gareth Huw Owen **Mo Adda**

School of Computing, University of Portsmouth,
Buckingham Building, Lion Terrace, Portsmouth, PO1 3HE, UK.
Tel: +442392 846782, Fax: +442392 846364, E-mail: {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 minimize overhead and maximize survivability. We then examine the quorum's performance as a location server and suggest improvements to the query mechanism and to the routing algorithm using the information.

Keywords:

Ad-hoc networks, quorum systems, self-organising systems.

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 within the 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 increases in size over 200 nodes, this form of discovery becomes extremely expensive.

Mainly, location-based routing in large-scale ad-hoc networks can be achieved through using location information from a GPS device. Nodes can address each other using their locations, by routing packets to neighbours that are closer than they are. However, 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. While this is ideal for networks that are static, it is not the case when nodes are mobile and networks span many kilometres. Location servers solve this problem by maintaining somehow up-to-date locations of nodes that sends frequent updates.

Several types of location server have been proposed to provide this information. Li et al [1] divided 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 proposed 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 you are from its location.

Giordano proposed [2] the most promising approach to location servers by allocating nodes within a specific region to the role of location server. However, they did not address issues such as node mobility, fault tolerance and query success.

Finally, Stojmenovic proposed [3] that all nodes to the north and south in a fixed-width column assume the role of a location server. Location searches are then performed by routing packets in a horizontal fashion that will eventually intersect the vertical column. This approach is not feasible for a ad-hoc network of large scale. For a more complete review of approaches to location servers, the reader is referred to Camp et al [4].

The approach shown in this paper expands Giordano's work on home-regions and develops a technique to provide a low-cost quorum that handles node mobility and failure. Also addressed are means 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.

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. As a result, only a small number of nodes are imposed upon rather than a large number in situations of high node density, or none in low-density scenarios.

A node sets up a location server by sending an initiation packet toward its home location. This packet will be received by the node closest to the point, which will then

assume the role of master of the server. As the host node moves, the role will be transferred to the node that is closest to the home location. Migration is the term referring to the 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 datum 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 [5] is very costly process in wired networks let alone wireless and as a result, 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 [6]. 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 (the environment) a list 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 containing the node's location (v_x, v_y) and the itinerary:

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

Each node maintains a set of all received beacons so that each component of the quorum is able to examine a list of which nodes hold which components.

We allocate the duplication task solely to the master to avoid the situation of excess replication by ill-informed slaves who are only able to see the beacons of the outskirt components. As the master is always the closest component (center), it is best placed to monitor the contents of its neighbours. When the number of neighbours hosting components falls below a threshold, the master will begin a replication 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 not 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.

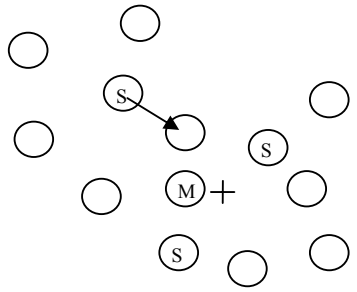

Figure 1: Illustration of quorum and component migration

In addition to the duplication task, the components perform one more tasked based upon information gathered from the beacons. The components self prune 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 die.

This process is performed on each node, for each component held, at regular intervals as described by the ManageComponent function below.

```
MANAGECOMPONENT(c)
1    n = closerNeighbour(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. Replicating one item at a time, like the unicast approach, is not expected to survive well due to the time taken to recover from a significant number of losses.

.

```
REPLICATE(C)  - UNICAST
1    for all n in neighbours
2          if !hasComponent(n)
```

```
3               then unicast(c)
4               exit forloop
5   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 a collapse scenario to occur. Therefore, we also inroduce 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)
```

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 with the node location and routes it towards. We describe this in the RoutePacket 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 closestNeighbour(p_location)
6   if !exists(p_location)
7       then route to
            closestNeighbour(homeLocation(p_dest))
```

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.

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

Of course, this approach does not guarantee a complete quorum update and so we examine this in our results.

```
RECVUPDATEPACKET(p)
1   c = Component(p_nodeID)
2   if !exists(c) or seenBefore(p) then ignore
3   if p_revision ≤ c_revision then ignore
4   c_data = p_data
5   broadcast(p)
```

Quorum systems are traditionally analysed based upon their performance in terms of load [7], fault-tolerance [8] and failure probability [9, 10]. Therefore, we measure the survivability or failure probability of the quorum over 30-minutes. The overhead or load in packets per minute of any one quorum, and the memory overhead per node in our scenario is measured. Finally, the fault-tolerance of the quorum by querying it and measuring the number of out-of-date results returned.

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 and routing success. As it is possible that the location information received could be out-of-date, we analyse the effect this has on a geodesic routing algorithm. We also propose that if a packet reaches the last known location of a node to find it is not there, then the algorithm switches to using DSR [11] to find the destination with a search radius of three hops We therefore examine the effect out-of-date data has on routing and how this simple DSR-aided modification improves.

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 [12] that provides models of all the layers experienced in a real experiment. We configure simulator with the parameters shown in Figure 2 to represent that of a kilometre-squared 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

Results

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

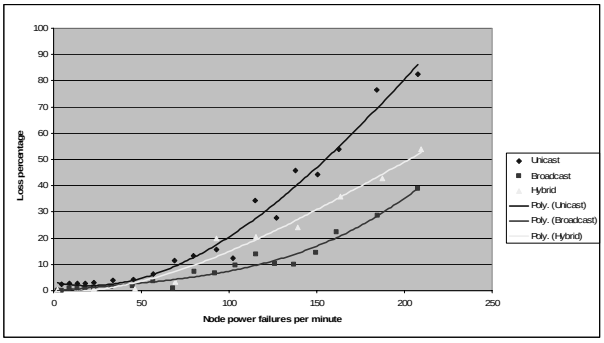

Figure 3: Survivability

All three versions of the algorithm survive exceptionally well even given extremely high failure rates with up to 20% of the quorums surviving despite every node in the network failing once per minute.

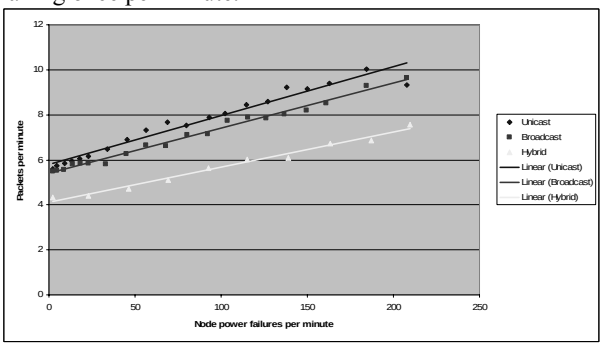

Figure 4: Overhead (packets/min)

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.

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 6.5-9 components per node given zero node failures.

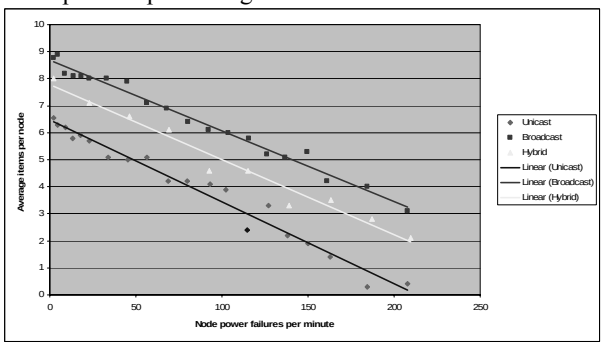

Figure 5: Memory overhead (components/node)

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.

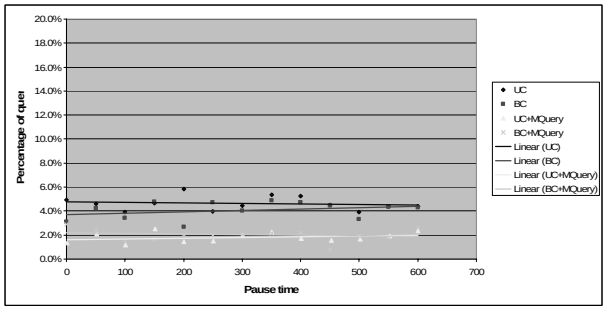

Figure 6: Out-of-date results returned

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

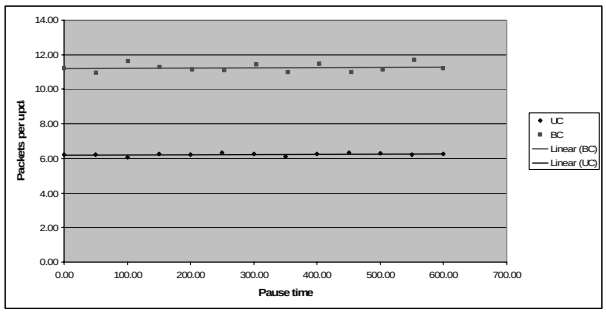

Figure 7: Update cost (packets)

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 add the DSR module to the routing algorithm, less than 3% of packets fail to reach their destination.

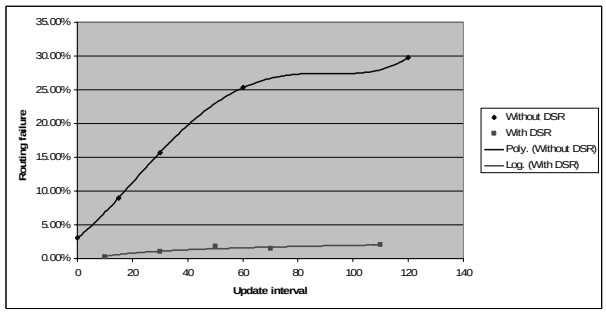

Figure 8: DSR-aided geodesic routing

Conclusion

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 those expected to occur in real-life. 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.

Routing using a simple geodesic algorithm performs poorly in networks of less than 200 nodes per square kilometre. We showed that by aiding the 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.

Our 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] J. Li, J. Jannotti, D. S. J. De Couto, D. R. Karger, and R. Morris, "A Scalable Location Service for Geographic Ad Hoc Routing," presented at 6th ACM International Conference on Mobile Computing and Networking (MobiCom), 2000.

[2] S. Giordano and M. Hamdi, "Mobility Management: The Virtual Home Region," EPFL, Lausanne, Switzerland, 1999.

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

[4] T. Camp, J. Boleng, and L. Wilcox, "Location Information Services in Mobile Ad Hoc Networks," presented at IEEE International Conference on Communications, 2002.

[5] D. Malkhi, M. K. Reiter, A. Wool, and N. Wright, "Probabilistic Quorum Systems," presented at Symposium on Principles of Distributed Computing, 1997.

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

[7] M. Naor and A. Wool, "The load, capacity and availability of quorum systems.," *SIAM Journal of Computing*, vol. 27, pp. 423-447, 1998.

[8] D. Barbara and H. Garcia-Molina, "The vulnerability of vote assignments," *ACM Transactions on Computer Systems*, vol. 4, pp. 187-213, 1986.

[9] D. Peleg and A. Wool, "The availability of quorum systems," *Information and Computation*, vol. 123, pp. 210-233, 1995.

[10] D. Barbara and H. Garcia-Molina, "The reliability of vote mechanisms," *IEEE Transactions on Computers*, vol. 36, pp. 1197-1208, 1987.

[11] D. Johnson and D. Maltz, "Dynamic source routing in wireless ad-hoc networks," in *Mobile Computing*: Kluwer Academic Publishers, 1996.

[12] "GloMoSim - Global Mobile Information Systems Simulation Library," UCLA Parallel Computing Laboratory.