
Sweety Chauhan (chauhan2@umbc.edu)
Jay Sheth (jay6@umbc.edu)

Department of Computer Science and Electrical Engineering
University of Maryland Baltimore County

December 15, 2005

Abstract

Wireless Sensor Networks are widely deployed in the military and civil applications for monitoring environments. Energy consumption is a big issue in wireless sensor networks. Sensors consume lot of energy if they have to communicate directly to the command node, therefore the concept of multi-hop and energy-aware routing techniques have been introduced. This project implements load balanced clustering algorithm and integration of clustering algorithm in Global Mobile Information Systems Simulation Library (GloMoSim) environment.
1 Introduction

The aim of our project is to implement load balanced clustering mechanism (where there is a gateway node and other sensor nodes) in the wireless network system. Clustering increases the network scalability and life time of the network. We implemented load balancing proximity-aware algorithm for cluster formation around high energy gateway nodes in GloMoSim. The simulation of clustering is also integrated in GloMoSim graphical visualization tool.

2 Motivation

Most of the wireless sensor networks applications require long lifetime of the networks. Many times wireless sensor networks are deployed in an area (depending upon the application) where it is hard to replace the nodes quite often as shown in Figure 1. There is limited amount of sensor’s energy for each node to communicate with command node. Sensor nodes can be networked into well defined clusters. All the nodes in a cluster can communicate with clusterhead instead of directly talking to command node. Hence in clustered networks the amount of energy consumption is highly reduced. Each cluster has a clusterhead/gateway which is high energy node. The clusterhead is responsible for communication between the nodes in a cluster. Packet delays due to congestion in overloaded clusters and wastage of resources in underloaded clusters can be reduced by balancing load among the clusters. The lifetime of system can also be improved if the load between clusterheads is balanced due to efficient use of resources.

Figure 1: Wireless sensor networks application in Battlefield
3 Background

This section of the project report provides the background information required for the implementation of project. It comprises of two main components: Load-Balancing Clustering Algorithm and GloMoSim.

3.1 Load Balanced Clustering

Clustering can be defined as the process of partitioning sensor nodes into groups. In sensor networks, cluster has following properties:

1. Each sensor node can belong to exactly one group except the boundary nodes which may be present in more than one group.

2. There exists a cluster head in each cluster which is responsible for communication between the nodes in a cluster (Intrachannel Communication).

3. Each cluster head can communicate with other cluster heads (Intercluster Communication).

There has been lot of research work done in the clustering of wireless sensor networks. We concentrated on load balanced clustering in this project. The load balanced algorithm discussed in [2].

In load balanced clustering, network setup comprises of 2 stages: Bootstrapping and Clustering.

During bootstrapping phase the following steps takes place:

1. Gateways broadcast a message called probe.

2. Each gateway starts bootstrapping at different times to avoid collisions.

3. Each gateway receives reply from nodes indicating their location and energy reserve.

4. Each node discovered in step 3 is included in a range set range set (RSet) of gateway.

Once bootstrapping is complete, clustering phase starts as follows:
1. Gateways calculate the cost of communication with each node in the range set RSet.

2. Exchange cost information between all the gateways.

3. After receiving data from all the other gateways each gateway starts clustering sensors.

4. Each sensor is allocated to only one cluster.

5. For intercluster communication all the traffic is routed only through the gateway.

For all sensor nodes $S_j$ each gateway $G_i$ constructs:

1. Range set 'RSet'

\[ S_j \in RSet_{c_i} \iff \left[ \left( R_{c_i} > d_{S_j \rightarrow c_i} \right) \land \left( R_{S_j, \text{max}} > d_{S_j \rightarrow G_i} \right) \right] \]

2. Exclusive set 'ESet'

\[ S_j \in ESet_{c_i} \iff \left[ \left( S_j \in RSet_{c_i} \right) \land \left( \forall k \neq i, S_j \in RSet_{c_k} \right) \right] \]

where:
- $R_{G_i}$ - range of gateway $G_i$
- $R_{S_j, \text{max}}$ - maximum range of sensor $S_j$
- $d_{S_j \rightarrow G_i}$ - distance between sensors $S_j$ and gateway $G_i$

The load on a gateway is defined as a function of processing load $PL_{G_i}$ and communication load $CE_{G_i}$ in the cluster:

\[ LG_i = f(PL_{G_i}, CE_{G_i}) \]

Processing load is due to processing of data from the nodes and communication energy is sum of the communication cost of all sensors in the cluster.

\[ CE_{G_i} = \sum_{j=0}^{n} C_{j,i} \]

The two algorithms described in [2] are:
1. Greedy Clustering Approach
2. Proximity-Aware Balanced Clustering

The two different approaches described in the paper are discussed in following sections.

3.2 Greedy Clustering Approach

The pseudo code for greedy clustering approach is given in [2]. The algorithm calculates the value of the objective function with respect to allocation of the sensor to each gateway.

\[ \sigma^2 = \frac{1}{G} \sum_{i=0}^{G} (X - X') \]

where \( \sigma^2 \) is the variance of load in the system, \( X \) is cardinality of gateway \( G_i \) and \( X' \) is the average cardinality including the node under consideration, \( G \) is the total number of gateways in the system. The algorithm calculates the value of the objective function with respect to allocation of the sensor to each gateway and sensor is allocated to the gateway which has minimum objective function.

3.3 Proximity-Aware Balanced Clustering

This algorithm [3] has two phases:

1. Minimum Communication Cost Clustering
2. Proximity-aware reallocation

// Minimum Communication Cost Clustering Phase (Source [1])

For node = 1 to N
    min_distance = INFINITY
    For gateway = 1 to G
        If (RSet[gateway].search(node) == TRUE)
            If((new_cost = Calculate_Cost(node, gateway)) < min_cost)
                min_distance = new_distance
                min_gateway = gateway
            End If
        End If
    End For
    FSet[min_gateway].add(node)
End For
In proximity aware algorithm, the gateways are considered to be in any of the 3 states based on the average load:

1. Balanced: The number of sensors in the cluster is equal to the average number of sensors in the system. The state of balanced gateway will never change.

2. Overloaded: The gateway will have more than the average number of the nodes in its cluster.

3. Underloaded: The gateway will have less than the average number of the nodes in its cluster.

4 GloMoSim

Global Mobile Information Systems Simulation Library (GloMoSim) is a scalable simulation environment for wireless network systems [1]. It was designed using the parallel discrete-event simulation capability provided by Parsec [1].

GloMoSim is capable of simulating networks with heterogeneous capabilities (e.g. - ad-hoc networking, multicast, IP protocols). Figure 2 gives an overview of the GloMoSim architecture.

The GloMoSim models currently available at each of the major layers are as follows:

- Physical (Radio Propagation) - Free space, Two-Ray
- Data Link (MAC) - CSMA, MACA, TSMA, 802.11
- Network (Routing) - Bellman-Ford, FSR, OSPF, DSR, WRP, LAR, AODV
- Transport - TCP, UDP
- Application - Telnet, FTP

Node aggregation technique in GloMoSim provides significant benefits to the simulation performance. It implies that the number of nodes in the system can
Figure 2: GloMoSim Architecture
be increased while maintaining the same number of entities in the simulation. In GloMoSim, each entity represents a geographical area of the simulation. Hence the network nodes which a particular entity represents are determined by the physical position of the nodes.

We downloaded GloMoSim academic research version from UCLA port. After successful installation of GloMoSim, it was tested by executing

```
# ./glomosim config.in
```

GloMoSim, contains the following directories:

- `/application` contains code for the application layer.
- `/bin` for executable and input/output files.
- `/doc` contains the documentation.
- `/include` contains common include files.
- `/java gui` contains the visual tool.
- `/mac` contains the code for the mac layer.
- `/main` contains the basic framework design.
- `/network` contains the code for the network layer.
- `/radio` contains the code for the radio layer.
- `/scenarios` contains some example scenarios.
- `/tcplib` contains libraries for TCP /`transport` contains the code for the transport layer.
Directory main contains the basic framework to execute GloMoSim, which includes 'driver.pc' file that defines the driver entity. Every Parsec program must include an entity called driver which serves purpose similar to the main function of a C program. This file reads the configuration file, initiate the simulation and writes the final statistics results file glomo.stat from some temporary file (.STAT.x). The sequence of events at run time is as follows:

1. The main function in driver.pc is run. This is the C main function, where GloMoSim starts.

2. The main function calls parsec main() to start the Parsec simulation engine, initialize the simulation runtime variables and create the driver entity. The parsec main function is used when the user wants to write his own main and is found at PCC DIRECTORY/include/pc api.h (since the function is part of the Parsec runtime system, it is not possible to access the source for it).

3. When the simulation ends, parsec main() returns, and the rest of the main function is executed.

The reason that GloMoSim uses a main function is because it is needed to collect the statistics into a single file and close the file. Therefore a function that runs after all the glomo entities have been finalized is required.

In GloMoSim, the driver entity (./main/driver.pc) reads the input file descriptor, establishes partitions, allocates memory for node information, calls appropriate functions depending on the read input values such as simulation time and node placement, and finally starts simulation by sending a StartSim message to the partitionEntityName instance of the GLOMOPartition entity type (defined in the 'glomo.pc' file).

The configuration parameters for the simulation are provided in config.in file. Once the simulation is complete, all the statistics is available in glomo.stat file (created in /bin directory).

The following files [1] are required to set up the simulation:

1. Basic Configuration File (Config.in file) It contains the configuration parameters like Simulation time, Seed, terrain dimensions, number of nodes, node-placement, radio-type, radio-bandwidth, MAC protocol, Network protocol, Routing protocol, app-config-file.
2. Application Configuration File (app.conf file). Applications such as FTP and Telnet are configured in this file.

4.1 Coding in GloMoSim

Source files in GloMoSim must have the following: There should be a block comment giving the filename, its purpose and any changes made to it. For header files, there should always be a

```
#ifndef HEADERFILE.H, #define HEADERFILE H, #endif
```

block around the header to allow for multiple includes. Source file must be organized into a logical order (grouping related functions together). Functions, types and macros that are only used in a single file should be declared static and put in the source file and not in a header file. Functions, types and macros that are used by other source files must be declared in a header. For function commenting, precede every prototype by a block comment containing the function name, a description of its purpose, what it returns and the assumptions it makes. Optionally, list each argument and its effect if not already obvious from the description.

Different steps have to be considered when adding a protocol to a layer in order to maintain consistency and proper functionality in GloMoSim. Three main functions have to be elaborated to add a new protocol:

1. Initialization Function - allocates and initializes the protocol specific data.

2. Finalization Function - generates the output statistics from the simulation run for the protocol.

3. Simulation Event Handling Function - performs simulation actions when scheduled with an event.

5 Routing Protocols in GloMoSim

5.1 AODV

GloMoSim implements AODV and the protocol assumes the MAC protocol sends a signal to the routing protocol when it detects link breaks. MAC protocols such as IEEE 802.11 and MACAW have this functionality. In IEEE 802.11 for example, when no CTS is received after RTS, and no ACK is received after
retransmissions of unicast packet, it sends the signal to the routing protocol.
If someone want to use MAC protocols other than IEEE 802.11, they must implement schemes to detect link breaks. A way to do this is, for example, using HELLO packets, as specified in AODV documents. Unsolicited RREPs are broadcasted and forwarded only if the node is part of the broken route and not the source of that route. If more than one route uses the broken link, multiple RREP messages are sent.

5.2 FSR
FSR is based on the premise that changes in a network region’s topology have less effect on a router’s packet forwarding decisions as the distance (in hops) between the router and the network increases. FSR is an adaptation of GSR where instead of propagating information through the network by periodic exchanges, a node exchanges individual link state table entries at different rates depending on the distance to the link’s source. Each update message does not contain information about all nodes. It exchanges information about closer nodes more frequently than it does about farther nodes thus reducing the update message size.

5.3 Wireless Routing protocols
Wireless Routing Protocol (WRP) is a pro-active protocol that maintains routing information through the exchange of triggered and periodic updates. A node that successfully receives an update message, transmits an acknowledgement back to the sender, indicating the link is still alive. In the event that a node has not transmitted anything within a specified period of time, it must transmit a Hello message (instead of exchanging the entire route table) to ensure connectivity. Otherwise, the lack of messages from a node indicates the failure of that link. When a node receives a Hello message from a new node, it sends that neighbor a copy of its routing table information. Each node maintains a distance table, a routing table, a link-cost table, a message retransmission list and an ack-status table.

6 Implementation
We installed GloMoSim 2.0.3 version and tried the sample program to check whether the installation has been done correctly. The implementation is divided into 2 main phases:

- Load Balanced Clustering
6.1 Load Balanced Clustering

In the beginning, the approach we followed was to introduce a new service layer in GloMoSim which can provide different options for the clustering algorithm. Using this new layer approach, different clustering algorithms can be integrated into GloMoSim environment.

In this strategy, sink node looks up in the whole network by using the flooding technique. We say that sink nodes provides service of clustering. The sink node sends a requesting packet to all its neighbors. When receiving the packet, each neighbor checks if it has already been received. If the packet has not been received before then the packet is broadcasted again until all the network is flooded with the packet. If the packet was received before then the nodes does nothing. In other words each node keeps track of receiving packets and maintains a table.

In the first approach the clusters are formed around head-clusters and are identified by a head-cluster ID. To enable the cluster formation and maintenance, all nodes keep the information about their neighbors in the neighboring table. Each node periodically broadcasts Hello packets which contain the Node-ID and the Node’s Head-Cluster ID. The Hello packets are only delivered to the immediate neighbors and are not allowed for multi-hop. When a node receives a Hello packet, it updates its neighboring table. If the entry with the received Node-ID does not exist in the table, a new entry is added, and a timer for that entry is initialized. When the timer expires, the corresponding entry is removed from the neighboring table.

In the beginning, a nodes assumes and advertises itself as head-cluster in Hello packet. When a head-cluster hears a Hello packet from another cluster-head, it compares its ID with the one of the heard head-cluster. The node with higher ID will revoke itself from cluster-head status and will advertise its new status on the network.

If an ordinary nodes does not hear hello packets form an head-cluster in a period of time, then it becomes head-cluster. Cluster formation time can be reduced if the time period between Hello broadcasts is chosen small. Moreover, Hello packets are very short and do not consume much bandwidth.

When the timer in the Neighbor table expires and the entry is being removed, the node is expected to check whether removed entry belonged to the node’s head-cluster. If so, the nodes searches its neighbor table for the first entry with (Neighbor-ID = Neighbor head-cluster ID) and accepts this neighbor as its new
Flood layer: Node 1 sends a SERVICE_REPLY (serv_id: 2) from APP layer at [s] 4.561529594 to be sent to Transport layer

Node 1: Msg sent by ServiceSendToTransportLayer FUNCTION to NODE 4
DATA PKT sent= [source: 1, ServId: 2, seqNum: 0]
RANDOM DELAY before sending to Transport Layer [s]: 0.144056952
Flooding Layer: Node 1 sends a BROADCAST to Neighbors
Node 1: Msg sent by ServiceSendToTransportLayer FUNCTION to ANY_DEST
DATA PKT sent= [source: 4, ServId: 2, seqNum: 0]
RANDOM DELAY before sending to Transport Layer [s]: 0.068043072
Flooding layer: Node 3 received SERVICE_REQUEST (serv_id: 2) from TRANSPORT layer at [s] 4.561529664
Flooding Layer: Node 3 - This packet HAS been received before.
Therefore, NOTHING is done

Flooding Layer: Node 3 => Size of the Received-Packets Table is: 1
  Node 3: Packet Received Table:
       sourceAddr  pktSeq
              4      0

Flooding layer: Node 0 received SERVICE_REQUEST (serv_id: 2) from TRANSPORT layer at [s] 4.644904301
Flooding Layer: Node 0: This packet HAS NOT been received before
Flooding Layer: Node 0 - FIRST time this packet is received.
Flooding Layer: Node 0 => Size of the Received-Packets Table is: 1
  Node 0: Packet Received Table:
       sourceAddr  pktSeq
              4      0

Figure 3: Service layer approach for clustering in GloMoSim
head-cluster. If no entry is found, the node itself becomes a Head-Cluster.

The greedy clustering algorithm was supposed to be integrated during cluster formation (determination of head-clusters and assignment of cluster members). Each cluster-head maintains a table called 'OtherHeadClusters' which contains the cluster-heads of neighboring clusters.

We tried implementing this in the service layer so that it will remain open to routing algorithms that use different metrics (power and stability) to reach the destinations. This scheme uses lower layers to achieve connectivity among clusters. The development of new service layer in the GloMoSim architecture was quite challenging because we need to develop our own APIs to interface with all GloMoSim components.

The learning curve of GloMoSim was high. After working for a month on this approach, we realized that within the semester time frame it can’t be completed.

In the middle of semester we decided to implement the clustering algorithm in the Ad-hoc On-demand Distance Vector (AODV) routing protocol to meet the deadlines of the project. Though the time frame didn’t allow us to implement clustering using the above approach but definitely it is an efficient approach to develop different clustering mechanism which can be integrated and extended in the future.

AODV routing protocol is used for routing data across Wireless Mesh Networks.

In the aodv.pc file the following algorithms are implemented:

1. Minimum communication cost clustering

2. Proximity aware reallocation

Proximity-Aware Balanced Clustering is integrated into aodv.pc of GloMoSim. The following functions have been added to aodv.h and aodv.pc:

void BroadcastHelloMessages(GlomoNode *node, NODE_ADDR destAddr);

void RoutingAodvRelayHello(GlomoNode *node, Message *msg, int ttl);

void RoutingAodvHandleHelloRequest(GlomoNode *node, Message
*msg, int ttl);
void RoutingAodvInitiateHelloReply(GlomoNode *node, Message *msg);
void RoutingAodvHandleHelloReply(GlomoNode *node, Message *msg,
NODE_ADDR srcAddr, NODE_ADDR destAddr);
double CalculateCostOfCommunication(GlomoNode *node1, GlomoNode *node2);
void RoutingAodvInform(GlomoNode *node, Message *msg, NODE_ADDR
srcAddr, NODE_ADDR destAddr);
void MinimumCostClustering(GlomoNode *node);
void CalculateCommEnergyPerCluster(GlomoNode *node);
void ProximityAwareClustering(GlomoNode *node);
void CalculateVarianceInLoad(GlomoNode *node);

6.2 Java Graphic User Interface

We have integrated clustering with the existing UI of glomosim, we gave a separate menu option in the existing options which would simply display a static position of nodes and boundary of the cluster. The clustering algorithm will generate coordinate points in UNTITLED.TRACE file. The 'UNTITLED.TRACE' file is processed to generate the coordinates of nodes in the cluster and an output file entitled 'Plot.txt' is generated. This file plot.txt is read by finalPlot java class to draw the the cluster boundary. There is added functionality which shows viewgraph and clustering in GloMoSim.

New Files added:

- ProcessFile.java
- Plot_final.java
- QuickHull3D.java
- Face.java
• Facelist.java

• Halfedge.java

• Point3d.java

**Modified files:**

• GlomoConfigDialog.java

• GlomoMenubar.java

The configuration parameters are set in `config.in` file for the simulation.

• Simulation-Time = 15 ms

• Seed = 1

• Terrain-Dimensions = 2000 x 2000

• Number of Nodes = 11 to 101

• Node-Placement = Random

• Mobility = None

• Propagation-limit = -111.0

• Propagation-pathloss = Two-ray

• Noise Figure = 10.0

• Temperature = 290.0
• Radio-type = Radio-Acc Noise (Standard Radio Model)

• Radio-frequency = 2.4e9

• Radio-Bandwidth = 2000000

• MAC-Protocol = 802.11

• Network Protocol = IP

• Routing-Protocol = aodv

• App-Config file = app.conf

• APPLICATION-STATISTICS YES

• TCP-STATISTICS NO UDP-STATISTICS NO

• ROUTING-STATISTICS NO NETWORK-LAYER-STATISTICS YES

• MAC-LAYER-STATISTICS NO RADIO-LAYER-STATISTICS NO

• CHANNEL-LAYER-STATISTICS NO MOBILITY-STATISTICS NO

• GUI-OPTION YES

7 Results

The effectiveness of proximity aware clustering scheme is validated through simulation. Testing was performed on simulations with diverse number of nodes placed in 2000 x 2000 GloMoSim Terrain Area (100 x 100 square meter area).

The results are compared with performance of the minimum cost clustering algorithm (based upon the shortest distance). Number of sensors is varied from 21 to 101 (21,41,61,81,101) and gateways 2 to 6.
7.1 Average Communication Energy per cluster

We measured total energy required to communicate between gateway and all the sensors in its cluster. If the clusters are formed based on minimum cost of communication (similar to shortest distance) the average energy consumed will be minimum but the load will not be balanced. Using the proximity aware algorithm, we tried to minimize the average communication energy comparable to minimum communication energy. The simulation results are shown in Figure 4.

![Figure 4: Average Communication Energy per cluster](image)

7.2 Standard Deviation of Load per cluster

Simulations were run to measure load on each cluster after clustering. We measured the deviation in load by varying the number of gateways from 2 to 6. Same simulations were run on minimum-cost clustering algorithm and the results are compared with proximity-aware algorithm. Figure 5 shows that for all distributions proximity-aware algorithm performs better than the minimum-cost clustering algorithm.
Figure 5: Standard Deviation of Load with increase in sensor’s density

8 Snapshots of GloMoSim Clustering GUI
Figure 6: GloMoSim GUI with Clustering menu option
Figure 7: Clustering algorithm running and generation of Untitled.trace
Figure 8: Min-cost clustering with 21 Nodes
Figure 9: Proximity-aware clustering with 21 Nodes
Figure 10: Min-cost clustering with 41 Nodes
Figure 11: Proximity-aware clustering with 41 Nodes
Figure 12: Min-cost clustering with 61 Nodes
Figure 13: Proximity-aware clustering with 61 Nodes
Figure 14: Min-cost clustering with 81 Nodes
Figure 15: Proximity-aware clustering with 81 Nodes
9 Future Work

The learning curve for GloMoSim was high and within the given time frame of the semester project, time was a big constraint to integrate both the algorithms from paper [2]. In future integration of greedy clustering algorithm in GloMoSim environment can be done. More features can also be added to java GUI.

References

