Final Report

31.8.16

Maintenance of fully-distributed UAV network

Instructor:

Professor Michael Segal

E-Mail: segal@bgu.ac.il

Students name:

Tamir Melamed

ID: 301223863

E-Mail: tamirmel@post.bgu.ac.il

Adi Meir

ID: 201413507

E-Mail: adime@post.bgu.ac.il

Table of content

Introduction & Motivation

In this project we desire to implement an Ad-Hoc network which simulates drones (the Ad-Hoc network) and units (mobile hosts) which are meant to be covered by the drones.
The drones will be launched from a base-station.
The drones will also know to avoid from entering a preflight-programmed No-Fly-Zones by themselves and know how to go around them while still covering as many mobiles hosts as possible.

We wish to develop a distributed-independent algorithm to achieve these goals. If this goal will be achieved and will be implemented on real-life drones this could save human resources – less human operators to fly the drones, less control from the base station to keep the connectivity of the network etc.

In the initial state there will be only one base-station, while we strive to achieve multiple base-stations as the project progress.
The drones are divided into 2 main types:

1. Covering Drone, this actively will cover the mobile hosts and will communicate via Wi-Fi with its adjacent drones and base-station.
2. Linker Drone, which will only provide linking between the covering drones and the base-station, implementing a tree-graph.

Main issues and problems in the project

Related problems:

- The metric **k-center** problem:

The metric k-center or metric facility location problem is a combinatorial optimization problem.

In graph theory this means finding a set of k vertices for which the largest distance of any point to its closest vertex in the k-set is minimum. The vertices must be in a metric space, or in other words a complete graph that satisfies the triangle inequality.

Formal definition:

Given a complete undirected graph $G = (V, E)$ with distances $d(v_i, v_j) \in N$ satisfying the triangle inequality, find a subset $S \subseteq V$ with $|S| = k$ while minimizing: $\max_{v \in V} \min_{s \in S} d(v, s)$

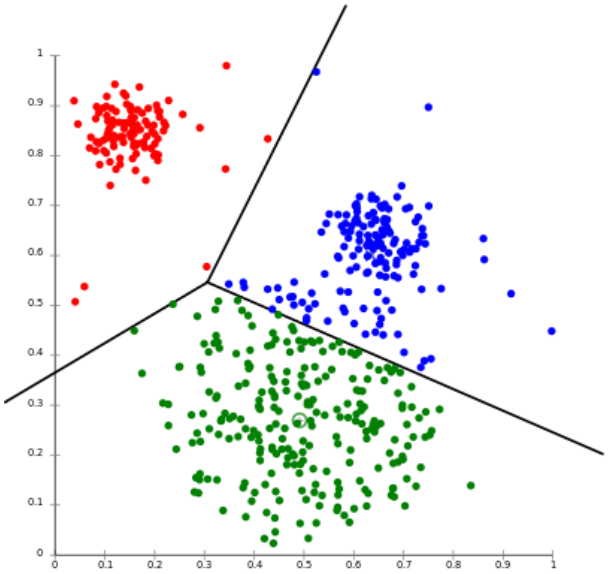
- The **clustering** problem:

Clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense or another) to each other than to those in other groups (clusters).

In our project we will refer the mobile hosts as cluster, and we'll want each cluster to be covered by a minimum number of drones.

In **Figure 2** we can see clusters divided to Voronoi cells, each clusters in its own cell, distinguished by colors.

Figure 2:



We will use clustering algorithm in order to minimize the maximum inter-cluster distances.

Project goals

- Optimizing resources (Using the smallest amount of drones possible to cover the maximum number of mobile hosts).
- Overcome the problem of limited number of drones (rank the mobile hosts and cover the highest rank sum as possible).
- Overcome the No-Fly-Zones (find the optimal path to surround the obstacles).
- Use multiple base stations implementing the Voronoi algorithm, each station will be responsible on its Voronoi cell.

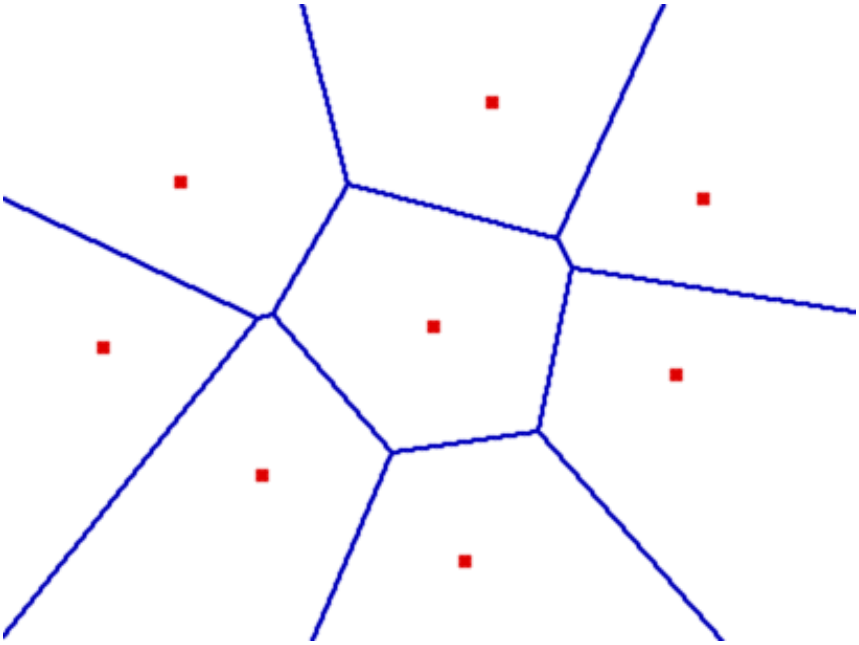
Theoretical Background

In order to implement our network, and decide which base station will be responsible for each area, we need to use a method which will divide the area into sub regions, while each region will be controlled by a specific base station. A known method is Voronoi Diagram.

Voronoi Diagram:

A Voronoi diagram is a partitioning of a plane into regions based on distance to points in a specific subset of the plane. That set of points (called seeds, sites, or generators) is specified beforehand, and for each site there is a corresponding region consisting of all points closer to that site than to any other. These regions are called Voronoi cells. The Voronoi diagram of a set of points is dual to its Delaunay triangulation.

Figure 1



We can see in the figure above that there is a line connecting every two points in the middle.

Algorithms

Reducing the number of active drones:

1. When a drone is serving as a linker drone, it takes into consideration all its adjacent drones, so when one of its adjacent can be responsible for its area, the linker drone can be free for another responsibility.
2. When a drone is functioning as linker and it is located at the end of the tree (is a leaf), the drone knows that it can be turned off.
3. When a drone is functioning as cover drone, and its neighbors can cover its hosts by fewer drones. – not only we can reduce the number of drones , but also there are less calls to the CC in order to get more covered areas.

Methods for reducing number of calls to CC in order to get additional drones:

1. The drones save their last requests from CC to get extra drones, so when they find themselves in the second critical state and when possible, they can use the same drones from previous calls and not to call another new drone.
2. Neighbors of the drones that can help to reduce number of calls to CC by cover by themselves the current drone's hosts.

Extensions:

There are two main extensions in the algorithm above:

1. In some scenarios, there are areas which defined as forbidden areas, so no drone can circle around it.
2. The number of drones is limited.

Forbidden areas:

We differ between two situations:

- When an additional drone is enter to a No-Fly-Zone.
- When during the algorithm there are drones which enter to No-Fly-Zone.

Dealing with these two cases:

- For dealing with the first case, we locate the new drone as far as possible on the vector connecting the drone with the critical situation and the intended new location.
- In order to deal with the second case, each drone will kwwp a small critical disk around its location, when the critical disk touches the boundary of the forbidden area, we should relocate the drone (so it will not enter a No-Fly-Zone).

There are two situations:

1. Linker drones
2. Covering drones.

Solutions for linker drones and covering drones:

1. If the drone is a linker drone, we should move it up or down, left or right depending on the type of the object. If we succeed – we are done. If not, we need to allow additional linker, and by that increase the freedom of movement and by that to avoid entering to No-Fly-Zone.
2. We cannot move the drone arbitrary since it may lose its covering area (and by that its hosts). We will use additional drones and distribute the responsibility of the current drone to the extra drones.

Limited number of drones:

There is another limitation which we need to consider, due to lack of sufficient amount of drones .We may encounter a problem when requesting an additional drone and we have reached to the maximum number of drones which can be supplied.

In this situation we need to adapt the algorithm to deal with this problem.

The algorithm will dismiss covering parts of the hosts and the goal is to maximize the cost of the nodes which are covered.

We assume that every host has its own cost and the drone is aware of it.

In order to do so, every drone keep list of costs of every hosts that it covers.

In order to keep the connectivity of the network, in case we need to disconnect a drone, we can only disconnect a drone which is a leaf.

We need to pick the leaf drone which covers the least cost and check whether the sum of its host costs is less than sum of the hosts which are about to leave the coverage area of the drone who requested and additional drone from the CC(coordinate center).

Project method and tools

OMNeT++

OMNeT++ is a discrete event simulation environment. It is an extensible, modular, component-based C++ simulation library and framework, primarily for building network simulators. "Network" is meant in a broader sense that includes wired and wireless communication networks, on-chip networks, queuing networks, and so on. Domain-specific functionality such as support for sensor networks, wireless ad-hoc networks, Internet protocols, performance modeling, photonic networks, etc., is provided by model frameworks, developed as independent projects. OMNeT++ offers an Eclipse-based IDE, a graphical runtime environment, and a host of other tools. There are extensions for real-time simulation, network emulation, alternative programming languages (Java, C#), database integration, SystemC integration, and several other functions. Its primary application area is the simulation of communication networks, but because of its generic and flexible architecture, is successfully used in other areas like the simulation of complex IT systems, queuing networks or hardware architectures as well. OMNeT++ provides component architecture for models. Components (modules) are programmed in C++, and then assembled into larger components and models using a high-level language (NED). Reusability of models comes for free. OMNeT++ has extensive GUI support, and due to its modular architecture, the simulation kernel (and models) can be embedded easily into the applications. Although OMNeT++ is not a network simulator itself, it is currently gaining widespread popularity as a network simulation platform in the scientific community as well as in industrial settings, and building up a large user community.

INET

INET Framework contains IPv4, IPv6, TCP, SCTP, UDP protocol implementations, and several application models. The framework also includes an MPLS model with RSVP-TE and LDP signaling. Link-layer models are PPP, Ethernet and 802.11. Static routing can be setup using network auto configurations, or one can use routing protocol implementations.

The INET Framework supports wireless and mobile simulations as well. Support for mobility and wireless communication has been derived from the Mobility Framework.

The INET Framework builds upon OMNeT++, and uses the same concept: modules that communicate by message passing. Hosts, routers, switches and other network devices are represented by OMNeT++ compound modules. These compound modules are assembled from simple modules that represent protocols, applications, and other functional units. A network is again an OMNeT++ compound module that contains host, router and other modules. The external interfaces of modules are described in NED files. NED files describe the parameters and gates (i.e. ports or connectors) of modules, and also the sub modules and connections

(i.e. netlist) of compound modules.

Simulation procedure

Our simulation starts with setting the number of hosts and drones according to the tested scenario, among other parameters like drones speed and transmission range. The amount of hosts and drones can be influenced by various factors and limitations, i.e. the amount of available drones vs. the size of hosts on the ground, according to the area size of the current simulation scenario, or even according to the importance we give to the current simulation run (when it's more important, we can assign more drones, and hence increase the coverage and connectivity of the network). When the simulation starts we have all the hosts in one assembly point, under cover from one drone, which is within connectivity range from the CC. Hosts move randomly within the defined area, where the drones supply to the CC a full real time picture of the hosts.
The simulation ends when all hosts are farther than $radius \cdot |Drones| + radius = radius \cdot (|Drones| + 1)$ – 'chain' of drones.
In this case, the ability to cover the drones is no longer available because of the drones' distance from the CC and the number of drones which is limited to a constant value.

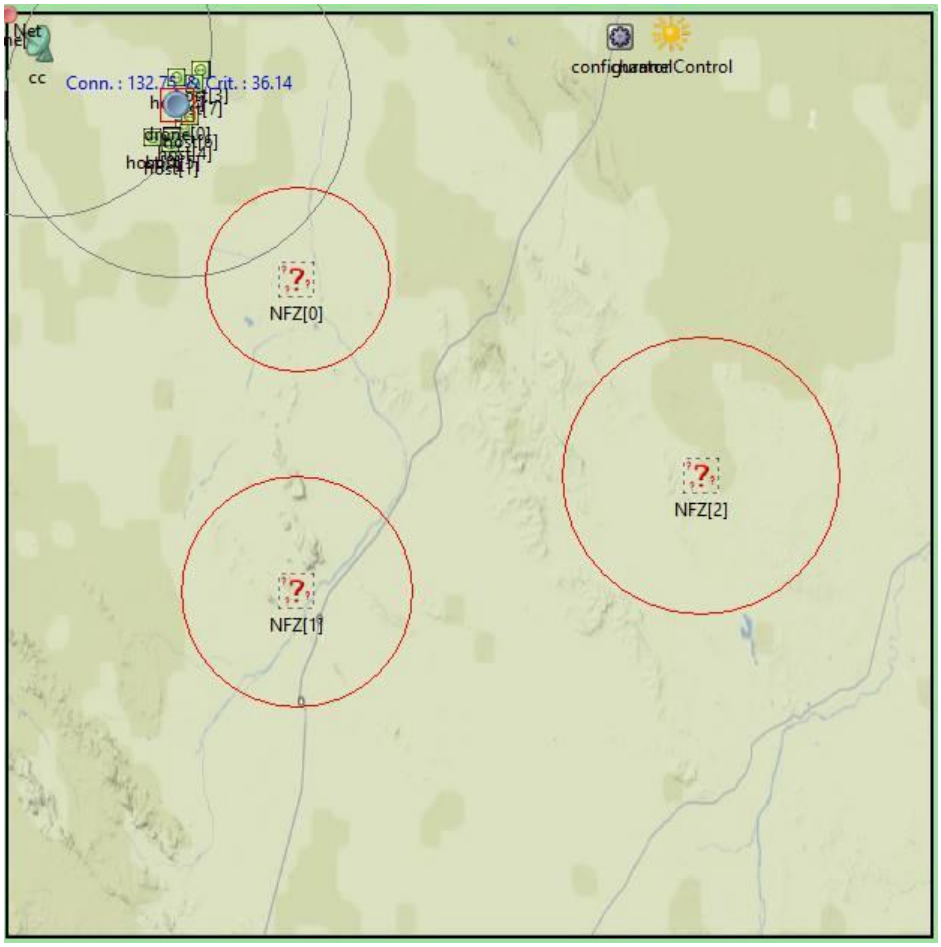
*assumption: all drones have the same cover and transmission radiuses.

Achieved progression (semester I):

We added a new NED module which implements a No-Fly-Zone (NFZ):

```
module NFZ{
    parameters:
        int x;
        int y;
        int z = default(0);
        int radius;
        @display("p=$x,$y;i=logo/no-fly-zone;r=$radius,,red");
}
```

We added an array of NFZs to the network; number of NFZ is defined at the ini file or chosen by the user when running the simulation. The definition includes (in addition to the number of NFZs):
x – longitude location on the network. y – latitude location on the network.
radius – radius of NFZ (can be different for each NFZ as shown in the figure below):



In the figure above we can see 3 different NFZs. Notice
to different radiuses.
We also implemented a set of functions which help to detect when a drone about to enter a NFZ.

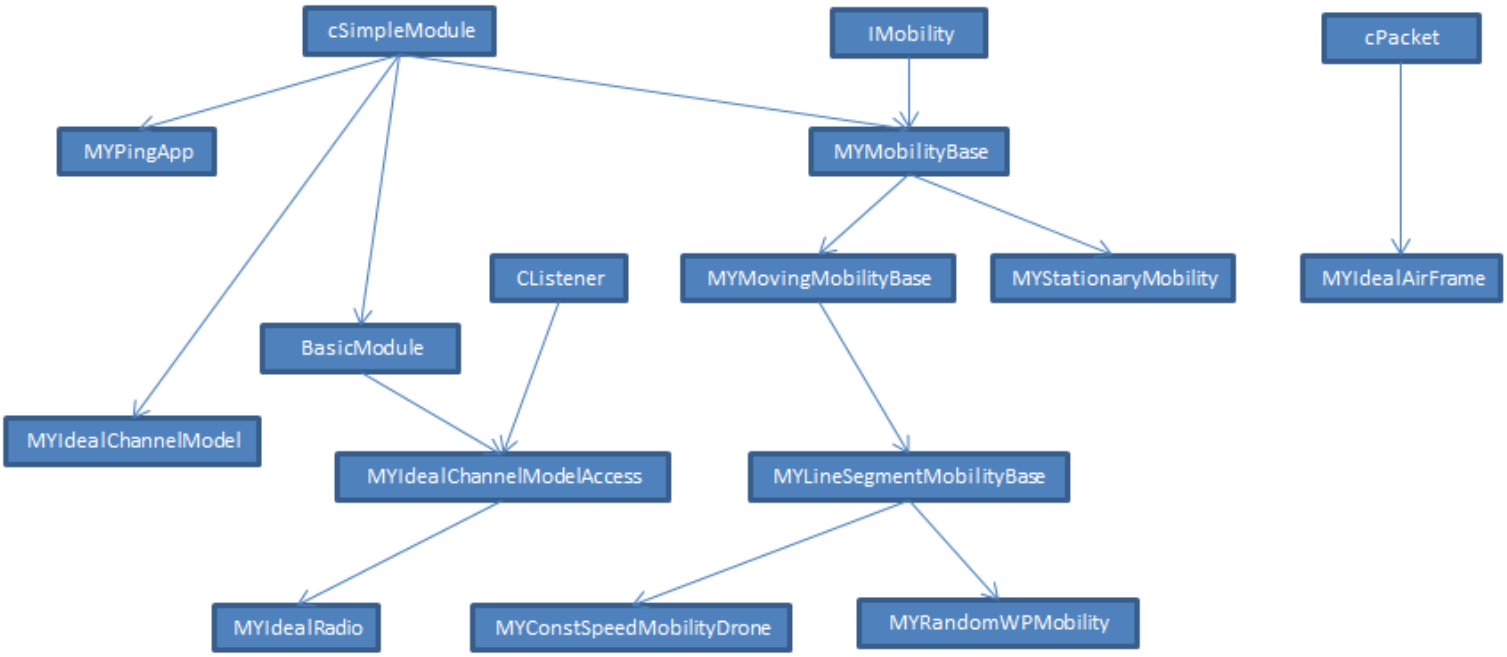
Debug

By investigating the previous project code which we inherited, and understating some of the logic and algorithms, we added a few changes to the simulation by adding the definitions which was mentioned before and vulnerable functions.
We tried to run the simulation on a newer version of OMNeT++ (4.6). We discovered that the simulation cannot run on that version and exits with an exception error.
We rolled back to version 4.3.1 as recommended on the instruction and it worked.

The code was implemented using multiple OMNeT++ and C modules.

Here presented the inheritance tree of the main project modules.

Inheritance Tree



*This tree is helpful for understanding the structure and logic of the whole simulation code.

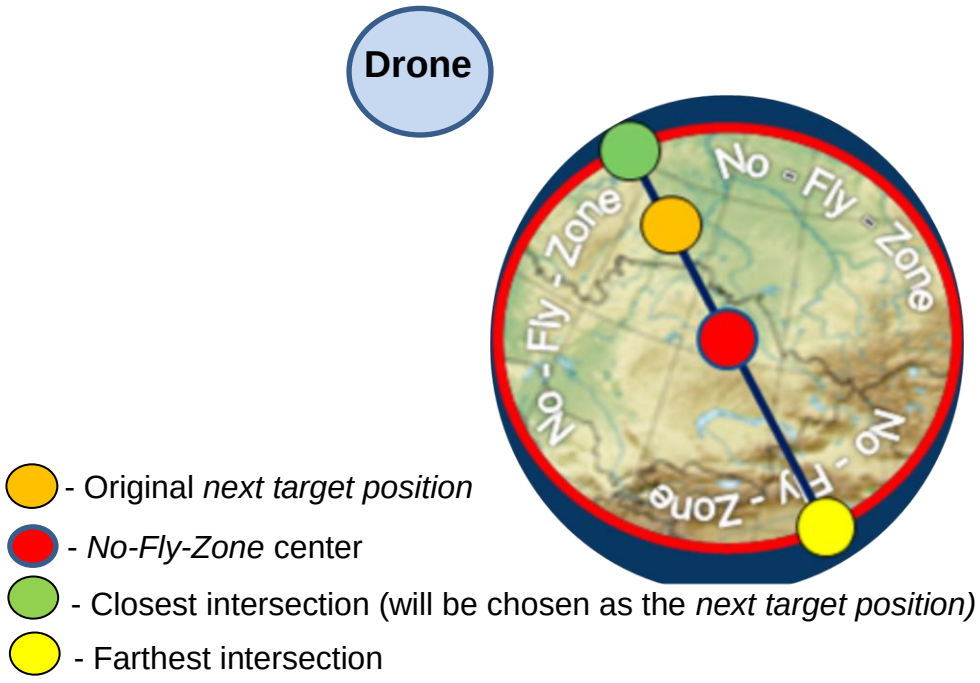
No-Fly-Zones (NFZ)

During the work on the evasion algorithm, (which was called as forbidden areas before) we combined all types of UAVs, and did not make a distinction whether it is a UAV which represent a linker or a cover UAV for our algorithm.
For each UAV the algorithm is the same and implement as follows:

Evasion Algorithm

Calculate next target position (covered hosts mass center).
If the next target position is not at NFZ, continue normally.
Else, calculate a new target position as follows:
Build a 0]using 2 Coordination points:
Coordinate A, Coordinate B.
Coordinate A is defined to be the center of the NFZ which the originally next target position was calculated inside of it.
Coordinate B is defined as the calculated next target position which was calculated originally.
The vector and the NFZ circumference will meet in two points.
We will take the new next target position to be the one which is closer to the originally calculated next target position.

Example in the Figure below:



```
Coord MYLineSegmentsMobilityBase::calculate_TP_Around_NFZ(int numOfNFZ, Coord badPosition){
   Coord nextTargetPos;
   nextTargetPos.z=0;
   int NFZx = getParentModule()→getParentModule()→getSubmodule("NFZ",numOfNFZ)→par("x");
   int NFZy = getParentModule()→getParentModule()→getSubmodule("NFZ",numOfNFZ)→par("y");
   int NFZradius=
getParentModule()→getParentModule()→getSubmodule("NFZ",numOfNFZ)→par("radius");

   double Ax = badPosition.x, Ay = badPosition.y, Bx = NFZx, By = NFZy, R = NFZradius;
   double Cx = Bx, Cy = By;

   double LAB = sqrt( pow(Bx-Ax,2)+pow(By-Ay,2) );

   // compute the direction vector D from A to B
   double Dx = (Bx-Ax)/LAB;
   double Dy = (By-Ay)/LAB;

   // Now the line equation is x = Dx*t + Ax, y = Dy*t + Ay with 0 <= t <= 1.

   // compute the value t of the closest point to the circle center (Cx, Cy)
   double t = Dx*(Cx-Ax) + Dy*(Cy-Ay);

   // This is the projection of C on the line from A to B.

   // compute the coordinates of the point E on line and closest to C
   double Ex = t*Dx+Ax;
   double Ey = t*Dy+Ay;

   // compute the euclidean distance from E to C
   double LEC = sqrt( pow(Ex-Cx,2)+pow(Ey-Cy,2) );

      // compute distance from t to circle intersection point
      double dt = sqrt( R*R - LEC*LEC);

      // compute first intersection point
      double Fx = (t-dt)*Dx + Ax;
      double Fy = (t-dt)*Dy + Ay;

      // compute second intersection point
      double Gx = (t+dt)*Dx + Ax;
      double Gy = (t+dt)*Dy + Ay;
      if (euclidean_distance(Fx,Fy,lastPosition.x,lastPosition.y) <euclidean_distance(Gx,Gy,lastPosition.x,lastPosition.y)){
            nextTargetPos.x=Fx;
            nextTargetPos.y=Fy;
      }
      else{
            nextTargetPos.x=Gx;
            nextTargetPos.y=Gy;
      }
   return nextTargetPos;
}
```

Priority algorithm

In order to overcome the situation where the number of drones which can be deployed is insufficient, and in addition, optimize the basic algorithm in a way such that the same number of drones can cover more valuable interest areas, we developed the priority algorithm.

To simulate different priorities for hosts we generated for each one a random number (uniformly distributed) between 1 and 10 (10 is Highest).

In the previous situation if a drone detects a critical situation, when a covered host is approaching the end of the drone's radius and there are no more drones which can be deployed, and no neighboring drone could help **[0]**, the simulation was terminated;

our algorithm suggests the following solution:

If a drone detects a critical situation and no neighboring drone can take ownership on the "rogue" drone, a PRIORITY REQUEST**[1]**broadcast message will be send from the drone.

When a leaf**[2]** drone will receive the message it will calculate the sum[3] of the host's priorities which are covered only by this drone.

When a leaf drone receives a PRIORITY REQUEST message it calculated the sum**[3]** and if it less than the sum received in the REQUEST message it answers with a PRIORITY REPLY[4] message, addressed to the source of the REQUEST message.

The PRIORITY REQUEST message source drone is waiting long enough to receive messages from all the leaves in the tree (worst case: the graph's diameter).

Then, if more than one message received (meaning more than one leaf had sum of less than the source's sum), the drone will choose the drone with the min sum.

If two, or more, drones will have the same min sum then the source drone will choose the one who's closest to it (first message received) and will send it a PRIORITY MOVE[5] message.

The leaf drone which will receive the MOVE message will change ownership of all the hosts which are not covered only by him to neighboring drones, then, move to new location (hosts' center mass) and take ownership on the hosts which entered the critical strip.

[1] PRIORITY REQUEST message is of type *MYIdealAirFrame* with the addition of the following parameters:
   a. Sum – the sum of all the host's priorities which are in the critical strip.
   b. NewLocation – the calculated next target position (center mass of all hosts in critical strip).
   c. ID – field which identifies the message as a PRIORITY REQUEST message.
   d. Source – the index from the source drone (who sent the message).

[2] Because of the insufficient drones number we would like to disconnect a cover leaf drone (if a linker is a leaf then it's redundant and will be sent back to CC). The reason for choosing a leaf and no other type of drone is that our graph's structure is a connected tree; we can only disconnect a leaf without jeopardizing the connectivity of the tree.

[3] The sum is calculated only on the hosts which are covered by a singular drone because all the others can be transferred (change ownership) to other neighboring cover drones.

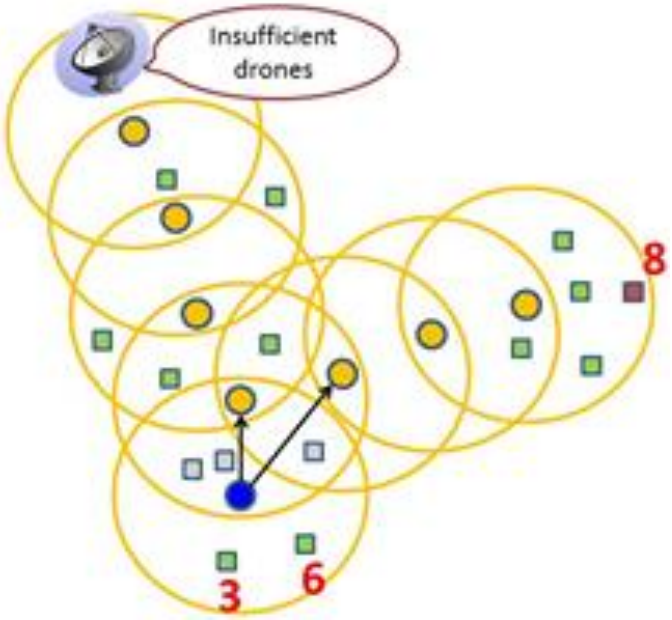
[4] PRIORITY REPLY message is of type *MYIdealAirFrame* with the addition of the following parameters:
   a. Sum – the sum of all the hosts' priorities, only those who're covered by this singular drone.
   b. ID - field which identifies the message as a PRIORITY REPLY message.
   c. Source – same as in PRIORITY REQUEST.
   d. Destination – the index of the drone the message is addressed for.

[5] PRIORITY MOVE message is of type *MYIdealAirFrame* with the addition of the following parameters:
   a. Destination – the index of the drone the message is addressed for.
   b. NewLocation – the calculated next target position (center mass of all hosts in critical strip).

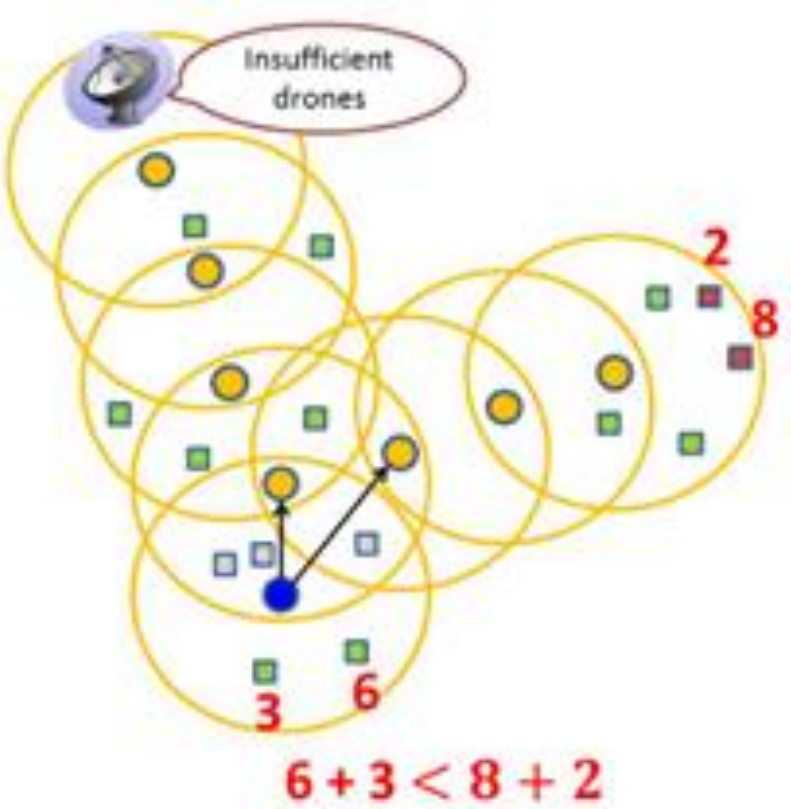
c. ID - field which identifies the message as a PRIORITY MOVE message.

Priority message example

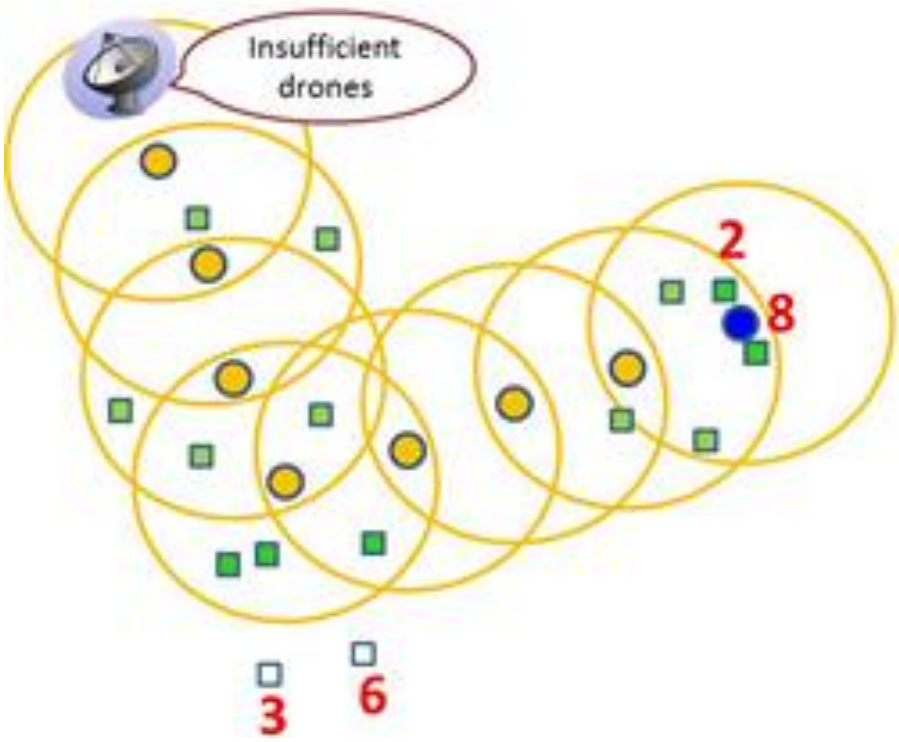
Right (red) host enters critical strip → its' cover drone sends request for a new cover drone from CC → insufficient drones → drone sends PRIORITY REQUEST message with sum=8 leaf drone (blue) receive the REQUEST message, sums the corresponding priorities (green 6+3) and decides not to answer because: $6+3>8$



A new drone with priority 2 enters critical strip as well → its cover drone send another REQUEST message with a larger sum this time (2+8=10) → the leaf sums again Vs the received sum and this time decides to answer with a PRIORITY REPLY message



The initial drone sends to the leaf a PRIORITY MOVE message → the leaf changes ownership of the overlapping hosts to its neighbors, moves to the new location (hosts in critical strip's center mass) and take ownership of them.



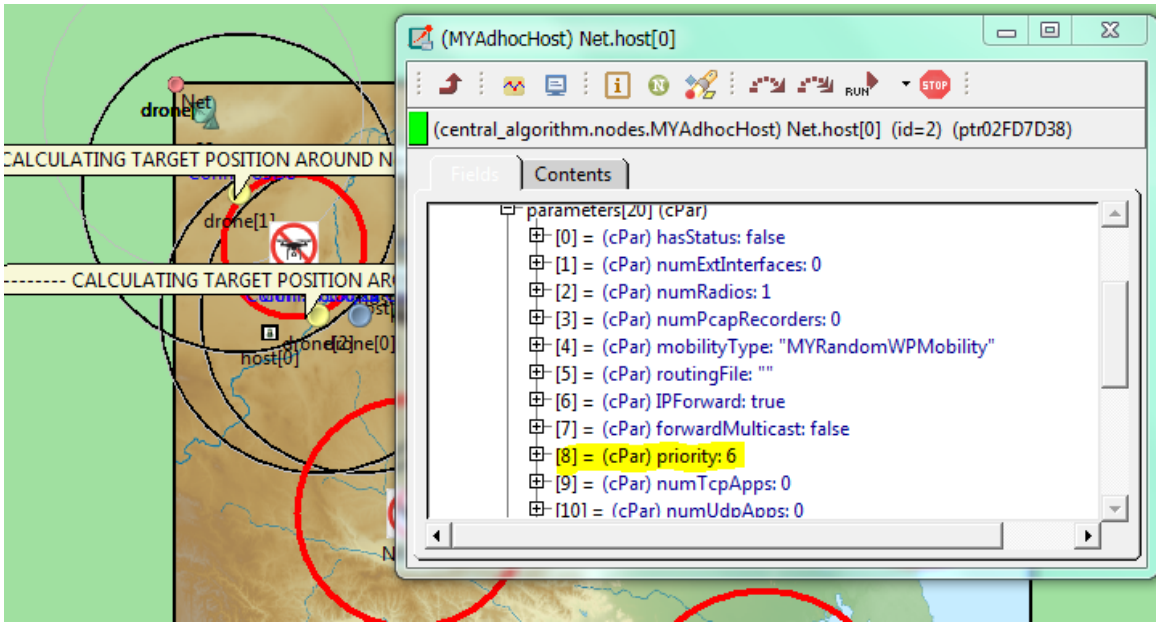
The priority parameter for each host is defined in its NED file:

```
module MYNodeBase
{
    parameters:
        @display("bgb=,448");
        @node;
        @labels(node,ethernet-node,wireless-node);
        bool hasStatus = default(false);
        int numExtInterfaces = default(0);
        int numRadios = default(0);                // the number of radios in the rou
        int numPcapRecorders = default(0); // no of PcapRecorders.
        string mobilityType = default(numRadios > 0 ? "MYStationaryMobility" : "");
        string routingFile = default("");
        bool IPForward = default(true);
        bool forwardMulticast = default(false);
        int priority = default(0);
```

Using the ini file this way:

```
*.host[*].priority =intuniform(1, 11)
```

We can see at the example below that host 0 got priority of 6:



The implementation of a priority message is by adding the following features to *MYIdealAirFrame* class:

```
class MYIdealAirFrame : public ::cPacket
{
 protected:
   simtime_t transmissionDuration_var;
   Coord transmissionStartPosition_var;
   double transmissionRange_var;
   char DroneDatabase_var[2048];
   int K_parameter_var;
```

```
  int currDepth_lvl_var;
  int isFindFrame_var;
  int isReplyFrame_var;
  int isRelocationFrame_var;
  int initiatorID_var;
  int frame_ID_var;
  int lastSender_var;

 public:
  int priority_sum;
  int priority_message_type;
  int src_sent_drone_id;
  int dst_sent_drone_id;
}
```

While the implementation for identifying and accepting the message is at MYIdealChannelModelAccess.cc file.

```
  //check if this is a priority type message. if yes, which type? , if no, continue
    if(priority_type != OTHER){
      //get my id number
                int myid=detachNumFromFullName3(getParentModule()->getParentModule()-
                >getParentModule()->getFullName());
      int sum=0; // sum of cur priorities
      ////************************* REQUEST MESSAGE **************
      if(priority_type==PRIORITY_REQUEST){  //check if im a leaf . if yes - sum all covered prio
hosts by me
        if(IsLeafDrone()){
          int num_of_covered_hosts=par("num_Of_Hosts_Cover");
          for(int i=0;i<num_of_covered_hosts;i++){
                        sum+=(int)getParentModule()->getParentModule()->getSubmodule
                              ("MYNodeBase",i)->par("priority");
          }
          if(airframe->priority_sum > sum){
            airframe->priority_message_type=PRIORITY_REPLY;
            airframe->priority_sum=sum;
          airframe->dst_sent_drone_id=airframe->src_sent_drone_id;
            airframe->src_sent_drone_id=myid;
            //send priority reply message
          }
          //  airframe->priority_sum=sum;

        }

        // if cur drone is not a leaf, forward the message
      }

      ////************************* REPLY MESSAGE **************
      else if(priority_type==PRIORITY_REPLY && airframe->dst_sent_drone_id==myid){    //
update DB with min sum and and source drone
 priority_sums[airframe->src_sent_drone_id]=airframe->priority_sum;
        //wait until all reply messages arrived
        if( (simTime() - getParentModule()->getParentModule()-
>getSubmodule("MYIdealChannelModel")->par("REQUEST_message_time_sent")) > 2){
// wait until all drones had enough time to answer
  /* send_MOVE_message to the drone with the min sum  */
            int min_sum=9999,min_idx;
            for(int i=0;i<20;i++){
```

```
                if(min_sum>priority_sums[i] && priority_sums[i]){
                    min_sum=priority_sums[i];
                    min_idx=i;
                }
            }

           //send Move message with drone id to move
            airframe->priority_message_type=PRIORITY_MOVE;
            airframe->dst_sent_drone_id=min_idx;
            airframe->src_sent_drone_id=myid;
            sendmove()
          }
         }

    //**********************MOVE MESSAGE*****************************
        else if(priority_type==PRIORITY_MOVE){      // change ownership of overlapping hosts and
move to new location
           // change ownership and maybe move to new location
           // cc2=getParentModule()->getParentModule()->getSubmodule("MYMobilityBase")-
>getSubmodule("MYMovingMobilityBase")->getSubmodule("MYLineSegmentMobilityBase")-
>getSubmodule("MYConstSpeedMobilityDrone")->par();
//if you are the drone who need to move, change ownership and move
           if(myid == airframe->dst_sent_drone_id){
    //needs to be current drone (a leaf which is the one who need to move
             //Drone_Info *d;
             //for  change ownership
             char* newdroneFullName=(char*)malloc(sizeof(char)*8);
             char* hostFullName=(char*)malloc(sizeof(char)*8);
             // sprintf(newdroneFullName,"host[%d]",d->drone_ID);
             for (HostList::iterator it=hostList.begin(); it !=hostList.end(); ++it){
                Host_Info *host = &*it;
                it->coverDrone_ID;
             //    sprintf(newdroneFullName,"host[%d]",);
                sprintf(hostFullName,"host[%d]",it->host_ID);
                cc2->setNewHostOwner((const char*)newdroneFullName,(const
char*)hostFullName);
             }
             //MYIdealRadio *r;
             MYLineSegmentsMobilityBase * drone;
             //this->priority_move_case=true;
             //update that we are in priority case
             getParentModule()->getParentModule()->getParentModule()-
>par("priority_case").setBoolValue(true);
 //update the position which the chosen drone needs to be located
             getParentModule()->getParentModule()->getParentModule()-
>par("xPosition").setDoubleValue(airframe->xLocation);
             getParentModule()->getParentModule()->getParentModule()-
>par("yPosition").setDoubleValue(airframe->yLocation);
             //move the drone to the new chosen location
             drone->move();
  //update the priority case is done . now we are not in priority case
             getParentModule()->getParentModule()->getParentModule()-
>par("priority_case").setBoolValue(false);
           }
        }



     }
```

Helping neighbor computation

When testing our neighbors for help with cover we use the following pseudo-code algorithm:

- Loop all drone neighbors.

- Find neighbor with smallest distance to any host from the critical set, which invoked the critical state and then the helping neighbor test.

- Calculate RneighborHelpStrip according to:

RneighborHelpStrip = transmissionRange – criticalStrip-RneighborhelpstripDelta

- If (RneighborHelpStrip > smallest distance found)
  neighbor can help.
- else
  neighbor can't help.

And according to the result decide if to send a request for extra drone to help with cover or connectivity.

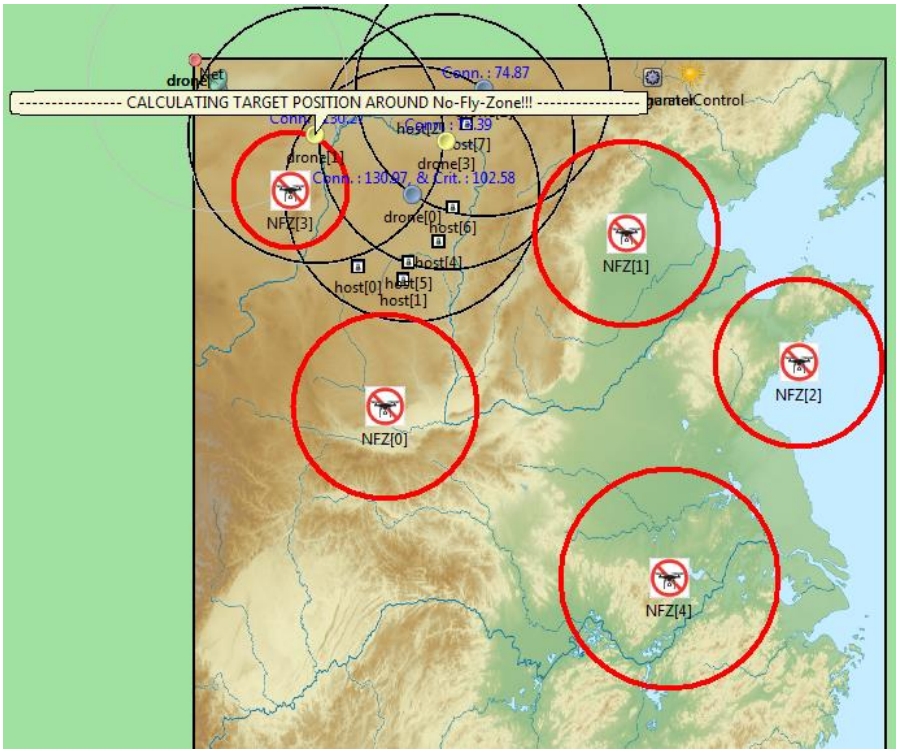
Final Results

In the figure below is a print screen from the running simulation.

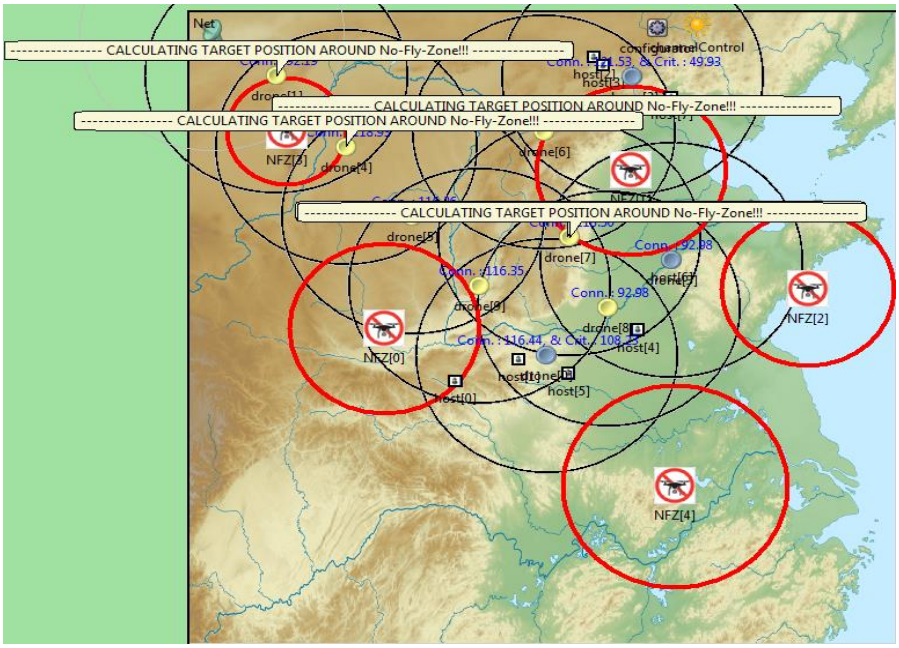
The black radius represents the drones' transmission and cover range.

Each time a drone calculates its *NextTargetPosition* to be inside a NFZ, it calculates a new position around it (as shown in the bubble).

The drones will never enter a NFZ, not cover or linker, in contrast to the hosts which can move freely.



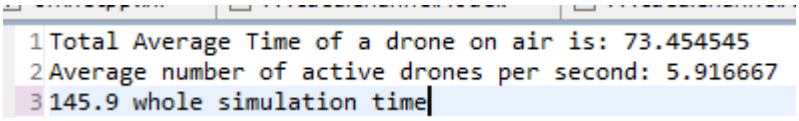
The example above is from the beginning of the simulation



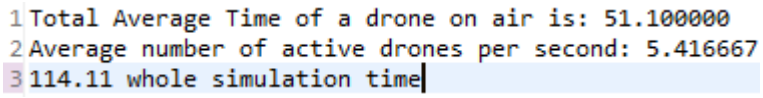
The example above is from a later time in the simulation

statistics

Case 1: The algorithm which does not relates to no fly zones



Case 2: The algorithm which relates to no fly zones



We can see that the average of the whole simulation time between the two cases is different; the time simulation for the case in which there is a consideration to forbidden areas is shorter than in original case (which does not consider forbidden areas) that's because the drones are not located in a direct way, but in new locations which make the route indirect. The more drones are used, the faster simulation ends.

We can also notice that the total average time of a drone on air in case 1 is larger than in case 2. That's because there are drones which cannot be active if ??

The parameter of average number of drones per second is pretty close so we can conclude that adding the feature for relating to forbidden areas is not only simulates a real situation but also does not require much more drones per second.

מכיוון שקייס 2 מדמה מציאות אמיתית יותר, קיבלנו שהניצול של כל מזלט בנפרד איננו מקסימלי כמו שהיה במקרה שהוא פחות דומה למציאות מכיוון שלא מתיחחס למקומות אסורים לטיסה. לכן הזמן הממוצע של מזלט באוויר במקרה של קייס 2 הוא נמוך יותר,

Project schedule

Preliminary Report – 11.12.2016

• Introduction and motivation
•Main issues and problems
•Project goals.
• Theoretical background
•Algorithms (forbidden areas and limited drones)
• Project methods and tools.
• Schedule.
• References.

Progress Report – 22.01.2016

• Accomplished work so far
• Changes and additions to the project.
• Intermediate achievements regarding stated objectives of the project.

Final Report – 31.08.2016

- Suggested Algorithms for presented problems
- 'extreme' situations
- Examples
- Final results

References

[1] Sergei Bespamyatnikh, Binay K. Bhattacharya, David G. Kirkpatrick, Michael Segal: Mobile facility location. DIAL-M 2000: 46-53.

[2] Sergey Bereg, Binay K. Bhattacharya, David G. Kirkpatrick, Michael Segal: Competitive Algorithms for Maintaining a Mobile Center. MONET 11(2): 177-186 (2006).

[3] Sergei Bespamyatnikh, Binay K. Bhattacharya, David G. Kirkpatrick, Michael Segal: Lower and Upper Bounds for Tracking Mobile Users. IFIP TCS 2002: 47-58.

[4] Nimrod Megiddo: Linear-Time Algorithms for Linear Programming in R3 and Related Problems. SIAM J. Comput. 12(4): 759-776 (1983).

[5] J. Gao, L. Guibas and A. Nguen, Deformable spanners and applications, ACM Sympos. Comp. Geom. (2004) 190–199.

[6] S. Har-Peled, Clustering motion, Discrete Comput. Geom. 31(4), (2004) 545–565.

[7] Jie Gao, Leonidas J. Guibas, An Nguyen: Distributed Proximity Maintenance in Ad Hoc Mobile Networks. DCOSS 2005: 4-19 .

[8] Teofilo F.GONZALEZ : Clustering to minimize the maximum intercluster distance. Theoretical Computer Science 38 (1985) 293-306.

[9] Michael Segal: Updated Variants Distributed Solution