המחלקה להנדסת מערכות תקשורת
Communication System Engineering

אוניברסיטת בן-גוריון בנגב
Ben-Gurion University of the Negev

SPANNER DESIGN FOR SWARM OF UAVs

Final Report

31/8/2020

Prof. Michael Segal

Tslil Greenberg 313347841
(tslilgr@post.bgu.ac.il)

Hila Gurevich 204434096
hilagur@post.bgu.ac.il

Project no. 37120011

Student 1 signature:_____

Student 1 signature:_____

Instructor signature:_____

TABLE OF CONTENTS

1. TABLE OF FIGURES

2. TABLE OF EQUATIONS

3. TABLE OF TABLES

4. TABLE OF GRAPHS

5. ACRONYM & ABBREVIATION

5.1 ACRONYMS

Table 1: Acronyms

Glossary	Meaning
UAV	Unhuman Aerial Vehicle
MANETs	Mobile Ad-hoc Networks
FANET	Flying Ad-hoc Network
VANET	Vehicular Ad hoc Network
CC	Control Center

5.2 KEY WORDS

Wireless network, Distributed network, T-spanner, NP hard, Voronoi diagram, Unit disk, Congestion control, Packet forwarding protocols.

6. INTRODUCTION

6.1 BACKGROUND AND MOTIVATION

6.1.1 Motivation

In this project we will build a new distributed algorithm that improve the communication between UAV's (unmanned aerial vehicle) swarm, to control center which are located far away from all UAV's.

The purpose of the algorithm is to get into an optimal state, so that the number of links (edges) is minimized and still avoid congestions in the network. Congestion is defined by several parameters, each one of them have different impotence and therefore receive different weight, and the edge weight will be the calculation of total variables.

6.1.2 Background

This is a continue project. In the last project (1) an Ad-Hoc network, which simulate UAV (the Ad-Hoc network) and units (mobile hosts) that are covered by the drones, was created. The swarm of UAV stayed connected to a command center which controlled the UAV supply. They divided the drones into 2 main types:

Covering Drone – drones which will take care of covering the units.

Linker Drone – drones which will only provide linking between the covering drones and the base-station or between two groups of covering drones.

The main goal of our project is to assimilate distributed routing algorithm that optimize minimum congestion in the network.

6.1.3 Previous knowledge

The knowledge received before starting the project includes knowledge in graph theory, network protocol and distributed algorithms.

It is important to understand the basic limitations of distributed algorithms, and how to use their benefits in the network.

Furthermore, from computer networks courses we know how to deal with congestions in the network and by that implement our algorithm. We also

learned about routing protocols, packet forwarding that will help us to build our algorithm.

6.2 PROJECT GOALS

- Optimal distributed algorithm which will be used on the UAVs of Israeli army.
- Simulation which will be illustrate the UAVs swarm behaver in the field. The simulation will show the way massages are route in the dynamic network by the algorithm.

7. THEORETICAL BACKGROUND

7.1 UNIT DISK

In mathematics, there are two definition for unit disk:

1) Open unit disk around point in the plane P , is the set of points which are located in less than one unit distance from P whose distance from P is less than one distance unit:

$$D_1(P) = \{Q : |P - Q| < 1\}$$

Equation 1 open unit disk definition

2) Closed unit disk around P is the set of points which are located in less than one or equal to unit distance from P:

$$\overline{D}_1(P) = \{Q : |P - Q| \leq 1\}$$

Equation 2 closed unit disk definition

Unit disks are special cases of disks and unit balls; as such, they contain the interior of the unit circle and, in the case of the closed unit disk, the unit circle itself.

since it's a continuous project, they have been using the definition of closed unit disk. we will proceed this use.

7.2 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. In our project every UAV seed.

We can see in Figure 1: Voronoi diagram that there is a line connecting every two points in the middle

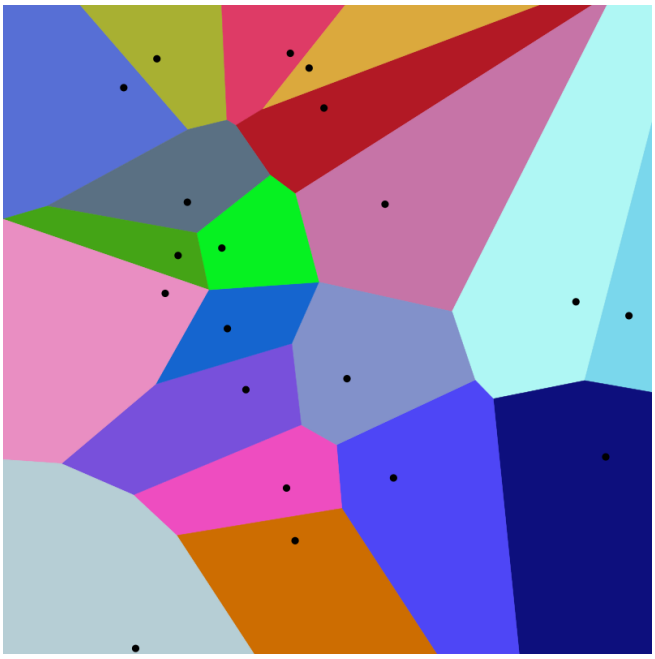


Figure 1: Voronoi diagram

7.3 GEOGRAPHIC ROUTING

The graph topology in our project is based on UAV 's location above earth surface, this definition and two features that we described below gives us the opportunity to use geographic routing (also known as directional, location-based, position-based, geometric routing).

In geographic routing the two important features are:

1) Etch node knows its own position and position of neighbors.
2) Source knows the position of the destination.

those features exist in our network, therefor we will be able to use geographic routing knowledge if needed, to get the best algorithm we can.

7.4 AD-HOC NETWORK

The nodes in a mobile ad hoc network intercommunicate via single-hop and multi-hop paths in a peer-to-peer fashion. Ad hoc networks do not use any fixed infrastructure.

more specifically, our network can use Flying Ad-hoc Network (FANET), The main advantage of Mobile Ad-hoc Networks (MANETs) is their portability or mobility. The widespread applications of MANETs has enabled subcategories of ad-hoc networking technologies, such as Vehicular Ad hoc Networks (VANETs) and Flying Ad hoc Networks (FANETs).

In single UAV architectures, the UAVs are connected to either base station located in the ground or connected with a satellite station for communication in star topology manners in Figure 2: Ad Hoc network with one leader we can see a star topology with one leader. In contrast to single UAV system, multi-UAVs systems such as our system in the project, have more than one UAV, therefore, multi-UAV can work in multi-hop scenario, and there is no need of all the UAVs to connect directly to the earth station or satellite station.

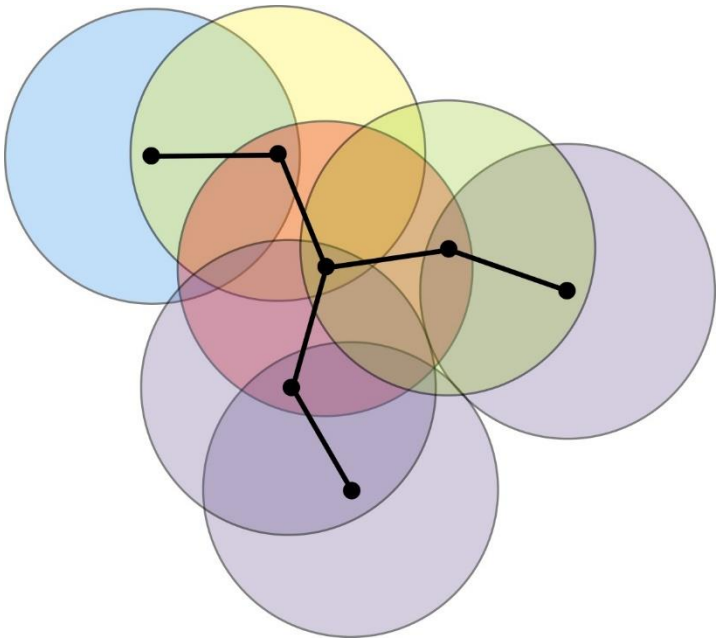


Figure 2: Ad Hoc network with one leader

7.5 PROACTIVE ROUTING PROTOCOLS

Proactive routing protocols manage all the tables of specific area and also gather routing information in a network. In FANET, there are different driven protocols which are not similar to each other. Nodes updates routing tables according to the change in topology. The routing protocol carries the latest information of nodes that is why there is no need to wait and select the path between sender and receiver. When the bandwidth is not used effectively (a lot of traffic between nodes) then this will not be recommended for large communication networks. Other than that, the protocol seems to be slow when topology is changed, or failure occurs. (2). Proactive routing protocol is described in Figure 3 Proactive Protocol

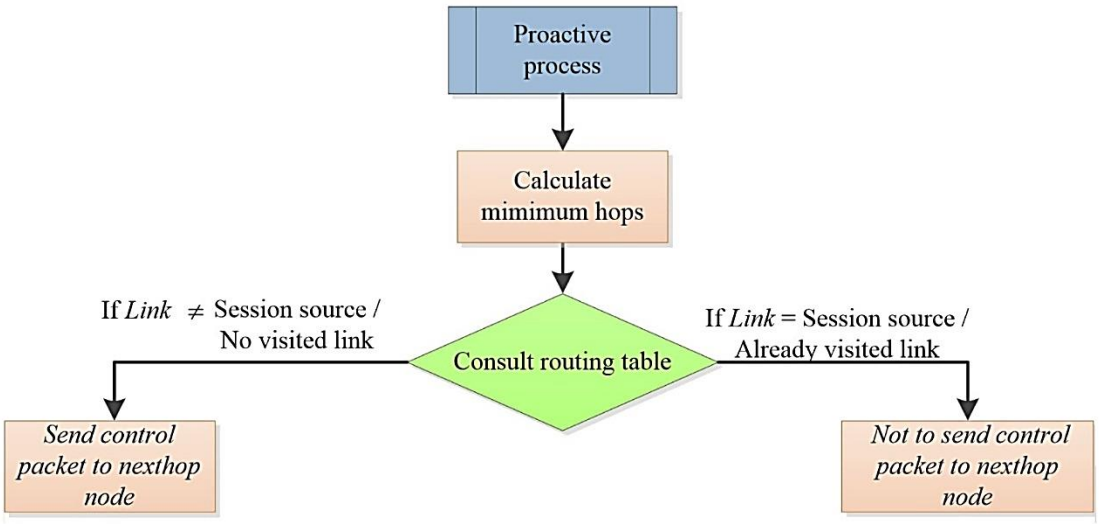


Figure 3 Proactive Protocol

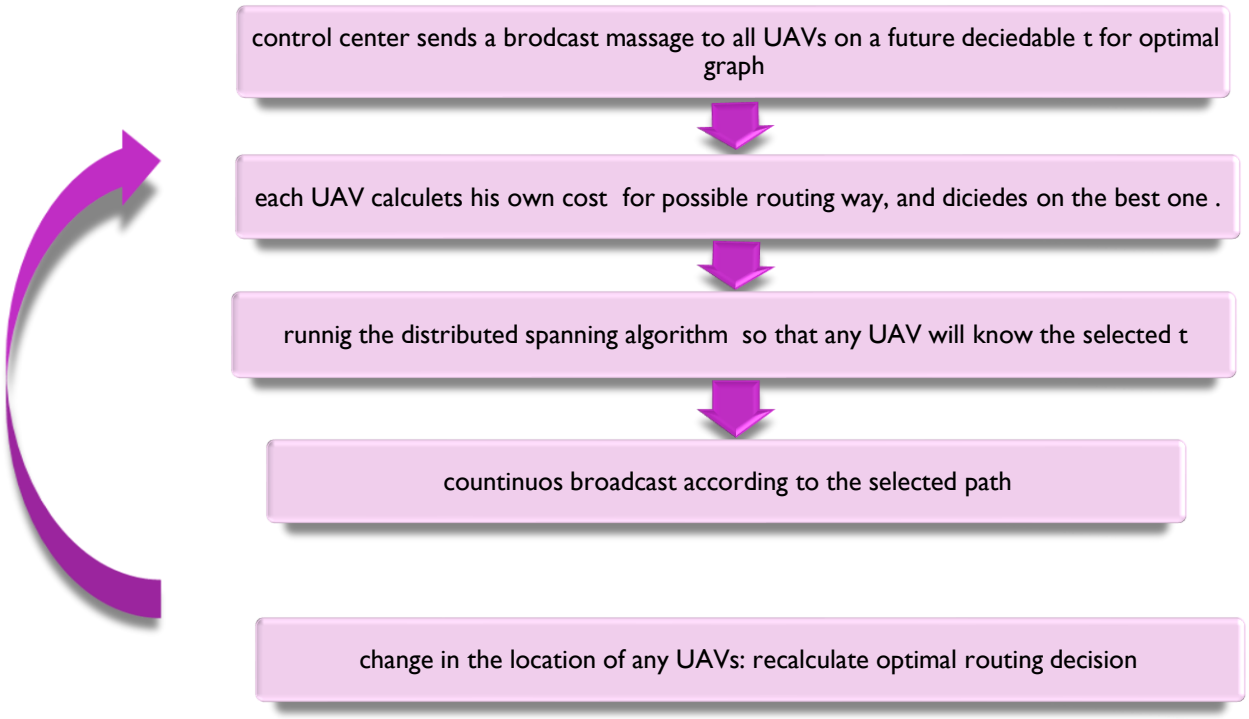
7.6 LEADER ELECTION ALGORITHM

There are a lot of leader election algorithm, but in this case the number of drones and the ID range of the drones are known, so we can easily chose a leader.

When a new drone is joining the swarm from the base station the base station announce the minimum ID of the on air drone. In that way all the on-air drones knows the leader.

8. PROJECT DESCRIPTION

8.1 ALGORITHM DESIGN



8.2 PSEODO CODE

Our algorithm would work this way:

1.We would calculate β value(Equation 3: calculate beta factor), using c- parameter governing the success probability, n- number of UAV's, k- spreading factor(given constant parameter) for which we would use in order to create randomness in the forwarding procedure .

$$\boldsymbol{\beta = \frac{\ln cn}{k}}$$

Equation 3: calculate beta factor

2. Broadcast r_{u}- $\mathrm{EXP}(\beta)$ (Equation 4: exponential distribution)value to neighbors within k distance.

$$f(x) = \begin{cases} \beta \cdot e^{-\beta x}, & x \geq 0 \\ 0, & otherwise \end{cases}$$

Equation 4: exponential distribution

3. Each vertex x that receive the r_u message will calculate and store $m_u(x)$ value as described in Equation 5: closeness random factor, and also store from whom the message was received mark as p_u, where d_G is the distance (in hops) between x and u.

$$m_u(x) = r_u - d_G(x,u)$$

Equation 5: closeness random factor

4. After receiving messages we will choose the edge which connected to the neighbor with the maximum value as described in Equation6 : maximum value/

$$m(x) = max_{u \in V}\{m_u(x)\}$$

Equation6 : maximum value

8.3 MEASURABLE PRODUCT

In our project the measurable product is an algorithm and simulation visualization that show our solution to the network optimization.

Today the exist code does not contain distributed routing algorithm that optimize minimum congestion in the network. So far, we wrote pseudo code (measurable product) and had a meeting with the project instructor. After the meeting we know that we need to modify the pseudo code by his instructions.

Our next mission is to write the pseudo code as C++ code and to assimilate it into the received past project (after modifications).

8.3.1 MYIdealAirFrame

In order to implement our part, we made changes in the message, to add more variables Figure 4: message content, in file `MYIdealAirFrame.msg`:

```
packet MYIdealAirFrame
{
    simtime_t transmissionDuration;     // Time it takes to transmit the packet, in seconds
    Coord transmissionStartPosition;    // position of sender when transmission starts
    double transmissionRange;           // max. distance from sender where reception is possible
    char DroneDatabase[2048];
    int K_parameter;
    int currK;                          //count down k parameter, for distibute R_u msg //Tslil & Hila
    double R_u;                         //Tslil & Hila
    int currDepth_lvl;
    int isFindFrame;                    // boolean- is this is a find message?
    int isReplyFrame;                   // boolean- is this is a reply message?
    int isRelocationFrame;              // boolean- is this is a relocation message?
    int initiatorID;                    // ID of the first sending drone
    int frame_ID;
    int lastSender;                     // ID of the last drone that pass the message
    bool leaf_linker;                   // if this drone is a leaf (till K_Parameter length) and its a linker then true, meening
    int initLinker;                     // linker from whom i received the org message HILA & TSLIL
    int senderLinker;                   //linker from whom i received the message HILA & TSLIL
}
```

Figure 4: message content

8.3.2 MYConstSpeedMobilityDrone

Send "msgEstablishSubGraph" signal when there was a topology change that causes a new drone to join the swarm.

```
char * droneAndLinker=strdup(myRadioRef2.linker);
strcat(droneAndLinker,myNewLinker);
cc2->setNewHostOwner(myRadioRef2.linker,curFullName);          //set curFullName(this) as the new linker of
cc2->setNewHostOwner(curFullName,myNewLinker);               //set myRadioRef2.linker to be the one that the
emitEstablishSubGraphSignal(droneAndLinker);                 //send "msgEstablishSubGraph" HILA & TSLIL
```

Figure 5: sending establish sub graph signal- connectivity drone

In Figure 5: sending establish sub graph signal- connectivity drone we can see the case of connectivity drone, in this case we send the signal after the new drone has connected and based his position at the chain, and his neighbors has been updated.

```
//Hila & Tslil
char * droneAndLinker=strdup("regular");
strcat(droneAndLinker,myNewLinker);
emitEstablishSubGraphSignal(droneAndLinker);
```

Figure 6: sending establish sub graph signal- regular drone

In Figure 6: sending establish sub graph signal- regular drone we can see case of regular drone.

8.3.3 MYIdealRadio

Handle received signal "FindMsgForEstablishSubGraph"

```
void MYIdealRadio::handleMYUpperDroneMsg(const char * droneAndLinker,int signalID)
{
    int hostID, linkerID;
    Coord place;
    Enter_Method_Silent();
    EV_INFO<<"HHHHHHHHHH handleMYUpperDroneMsg TTTTTTTTTTTTTTTT"<<endl;
    // build the initiating msg from drone (comes from signal)
    char name[32],targetHost[32];

        EV_INFO << " --------------------  HANDLING CRITICAL ALERT CASE ---------------------------- MYIdealRadio " << endl;
        sprintf(targetHost, "establishFrame");

    cPacket *frame = new cPacket();
    frame->setName(targetHost);
    int initiatorDroneId = detachNumFromFullName();
    frame->setKind(initiatorDroneId);           //SHOULD BE THE CURR DRONE INITIATOR - BUT !! only if not used in any other place

    cMessage *msg2 = frame;
    frame = check_and_cast<cPacket *>(msg2);

    delete frame->removeControlInfo();

    MYIdealAirFrame *airframe = createAirFrame();

    if(signalID == (int)establishSubGraph){
        EV_INFO<<"HHHHHHHHHHHHHHHHHHHHH receive signal send msg TTTTTTTTTTTTTTTTTTTT"<<endl;
        int k,i;
        std::string initator1= std:: string(droneAndLinker,0,8);
        std::string linker1 = std::string(droneAndLinker,8,8);
        const char * initiator=initator1.c_str();
        const char * linker=linker1.c_str();
        double R_u=culc_R_u();
        sprintf(name, "FindMsgForEstablishSubGraph");
        airframe->setCurrDepth_lvl(0);
        k = this->getParentModule()->getParentModule()->getParentModule()->par("K_parameter");
        airframe->setK_parameter(k);
        airframe->setCurrK(k);
        airframe->setR_u(R_u);
        airframe->setIsFindFrame(1);
        airframe->setIsRelocationFrame(0);
        airframe->setIsReplyFrame(0);
        airframe->setInitiatorID(detachNumFromFullName(initiator));
        airframe->setLastSender(detachNumFromFullName());
        airframe->setFrame_ID(frameID_counter++);
        airframe->setInitLinker(detachNumFromFullName(linker));
        airframe->setSenderLinker(detachNumFromFullName(getLinker(myRadioRef->linker)));
        for(i=0 ;i< 2048; i++)
            airframe->setDroneDatabase(i,0);
```

Figure 7: handle received signal

In Figure 7: handle received signal we updated the relevant message value for this signal type.

8.3.4 MYIdealChannelModelAccess

Handle "*msgEstablishSubGraph*" message.

```
//Tslil & Hila- forward message to all neighbors
else if(!strcmp(airframe->getName(),"FindMsgForEstablishSubGraph")) { //handle recived message
    simtime_t time=simTime();
    std::ofstream point;
    point.open("C:/Users/Hila urevich/Documents/FINAL_PROJECT/est_msg.txt",std::ios::app);
    point<<"1,simtime is:"<<time<<std::endl;
    point.close();

    EV_INFO<<"start time"<<time<<endl;
    int k=airframe->getK_parameter();
    int currk=airframe->getCurrK();
    double R_u=airframe->getR_u();
    double newCost= R_u-(k-currk);
    int initLinker_int=airframe->getInitLinker();
    int myid=myRadioRef->ID;

    int initiatorID = airframe->getInitiatorID();
    char initLinker[9];
    AtachNumToFullName(initLinker_int,initLinker);
    const char * newInitLinker = (const char*)initLinker;

    char lastSenderName[9];
    AtachNumToFullName(lastSender,lastSenderName);
    const char* lastSenderName1 = (const char*)lastSenderName;

    //************************* Handle received message***********************
    //checking if there is a need to update the linker
    if(newCost>myRadioRef->currCost) {
        const char * linker=myRadioRef->linker;
        if(strcmp(linker,"cc")!=0) {
            if(!(myRadioRef->ID == initiatorID && strcmp(newInitLinker,lastSenderName1)==0)) { //dont connect to the original linker before the extradrone was added
                int currLinker=detachNumFromFullName(linker);
                const char * curFullName = this->getParentModule()->getParentModule()->getFullName();
                const char * newLinker = cc->lookupNameByName(lastSenderName1);
                cc->setNewHostOwner(curFullName,newLinker);
                if(check_cycles(curFullName) == true) { // ok to update the linker
                    this->getParentModule()->getParentModule()->bubble(" I have changed my linker ");
                    myRadioRef->currCost = newCost;
                }
                else {          //change back to org linker
                    char orgLinker[9];
                    AtachNumToFullName(currLinker,orgLinker);
                    const char * newOrgLinker = (const char*)orgLinker;
                    const char * newCurrLinker = cc->lookupNameByName(newOrgLinker);
                    cc->setNewHostOwner(curFullName,newCurrLinker);//set myRadioRef2.linker to be the one that the connectivity gap we are closing
                }
            }
        }
    }          //end Handle received message
```

Figure 8: handle received message- update the linker

In Figure 8: handle received message- update the linker we checked if this is the maximal cost received so far.

↳We checked if the current drone linker isn't the "cc".

↳ We checked if the drone we want to link to isn't the original linker before the extra drone was added.

↳ We made sure that the drone I want to connect to does not cause cycles.

- The check cycle function is implemented as shown in Figure 9: Check cycle

```
//HILA & TSLIL return true- if no cycle found, false- there is a cycle
bool MYIdealChannelModelAccess::check_cycles(const char * start) {
    const char* next = getLinker(start);
    while (strcmp(next, start) != 0) {
        if (strcmp(next, "cc") == 0)
            return true;
        next = getLinker(next);
    }
    return false;
}
```

Figure 9: Check cycle

- Initialize the message parameter before broadcast as described in Equation 3: calculate beta factor and Equation 4: exponential distribution implemented in Figure 10: calculate beta and generating exponantial value

```
//TSLIL & HILA: calculate beta value for exponential probability
double MYIdealChannelModelAccess::culc_beta(){
    double c_par=this->getParentModule()->getParentModule()->getParentModule()->par("C_parameter");
    int number_Of_Hosts = sizeof(this->final_List)/sizeof(DronesList);
    int k_par=this->getParentModule()->getParentModule()->getParentModule()->par("K_parameter");

    return log(c_par*number_Of_Hosts)/k_par;
}

double MYIdealChannelModelAccess::culc_R_u(){
    double r_u= exponential(1/culc_beta());
    return r_u;
}
```

Figure 10: calculate beta and generating exponantial value

```
//*******broadcast the message
if(myRadioRef->isFirstEstablish==false && currk==k){//if its your first sent message generate your random number
    double R_u=culc_R_u();
    airframe->setR_u(R_u);
    myRadioRef->isFirstEstablish=true;
}
if(currk>1){//forward the message
    airframe->setSenderLinker(detachNumFromFullName(myRadioRef->linker));
    airframe->setIsFindFrame(1);
    currk=currk-1;
    airframe->setCurrK(currk);
    airframe->setIsReplyFrame(0);
    airframe->setLastSender(myRadioRef->ID);
    //airframe->setInitLinker(myRadioRef->linker);
    cc->send_Find_Message(myRadioRef, airframe,lastSender); // Broadcast the message to all neighbors
}
else{
    myRadioRef->isFirstEstablish=false;
    airframe->setIsFindFrame(0);
}
```

Figure 11: handle recived message- broadcast the message

In case it's the drone first sent message, he needs to generate a random number.

Update the message with the relevant values before distribution to neighbors as shown in Figure 11: handle recived message- broadcast the message

8.3.5 MYIdealChannelModel

Broadcast the message to all drones inside my transmission range as show in Figure 12: broadcast the message.

```
else{ //Hila & Tslil- if establish message send to all! neighbors including last sender
    fullname = r->radioModule->getParentModule()->getParentModule()->getFullName();
    if( !(strcmp(fullname,srcfullname) == 0 ))  // don't consider yourself
    {
        bool inRange = srcRadio->pos.sqrdist(r->pos) <= sqrTransmissionRange -50;//////////////////////////hila & tslil
        if (inRange)   //include only if inRange and not self
        {
            if(((strcmp(r->radioModule->getParentModule()->getParentModule()->getName(),"drone") == 0) && (r->myisActive ) )) //also check if myisActiv
            {
                ccRadio = lookupRadioByName("cc");
                if(ccRadio.pos.sqrdist(r->pos) > sqrTransmissionRange/3)       // patch for avoiding to send messages to drones that still are in CC bu
                {
                    // account for propagation delay, based on distance in meters
                    // Over 300m, dt=1us=10 bit times @ 10Mbps
                    simtime_t delay = sqrt(srcRadio->pos.sqrdist(r->pos)) / SPEED_OF_LIGHT;
                    EV <<""<<endl;
                    EV << " ---------------------- SEND DIRECT from: "<< srcfullname << " to: "<< fullname <<endl;
                    EV <<""<<endl;
                    messageSent++;
                    noNeihgbors = false;
                    check_and_cast<cSimpleModule*>(srcRadio->radioModule)->sendDirect(airFrame->dup(), delay, airFrame->getDuration(), r->radioInGate);
                }
```

Figure 12: broadcast the message

8.4 ADJUSTMENTS IN THE CODE

During the QA test part for our changes in the project, we ran into an important bug that caused drones to return to the CC while they were covering host. We had to fix this bug since it prevented us to achieve bigger topologies, in order to check and proof that we actually improved the existing application. We found that while trying to move the drone several tests has occurred, and the default coordinate that was set to {0,0,0} (go back to the CC). We have changed the default coordinate to be the current passion of the drone.

The change we made in the code is attach in Figure 13: Coordination fix.

```
Coord MYConstSpeedMobilityDrone::calcSECPosition()
{
    double RminPair = 77777, RminTriplet = 77777, TripletSECX, TripletSECY,ax=0,ay=0,bx=0,by=0;
    double criticalRange = 0, Xnew=0, Ynew=0;
    MYIdealChannelModel::myRadioList pairSEC;
    MYIdealChannelModel::myRadioList tripletsSEC;
    int size = (int)hostNeigborsList.size(),count=0;
    Coord p, PairSECCoord, TripletSECCoord,currPos;

    //calc the position according to the hostNeigborsList, droneNeigborsList and ccRadioEntry (need to check size cuz not always set)
    //MOSHE: find best position for more then 2 hosts set pairSECCord as the middle of the two best pairs that cover all neighbour hosts or to zero
    if(size >= 2){
        //get BEST PAIR can be only if isCoveringAll
        pairSEC = checkPairs();

        for (MYIdealChannelModel::myRadioList::iterator it0 = pairSEC.begin(); it0!=pairSEC.end() ; it0++){
            switch(count){
            case 0:{
                RminPair = it0->pos.z;
                it0->pos.z = 0;
                ax = it0->pos.x;
                ay = it0->pos.y;
                break;
            }
            case 1:{
                bx = it0->pos.x;
                by = it0->pos.y;
                break;
            }
            }
            count++;
        }
        PairSECCoord = {(ax + bx)/2, (ay + by)/2, 0};
    }
    //Hila & Tslil
    if((int)pairSEC.size() == 0){       // when (int)PairSEC.size() == 0 so no PairSECCoord to set
            char lastSenderName[9];
        AtachNumToFullName(myRadioRef2.ID,lastSenderName);
        const char* lastSenderName1 = (const char*)lastSenderName;
        currPos= cc2->getHostPosition(lastSenderName1);
        PairSECCoord.x=currPos.x;
        PairSECCoord.y=currPos.y;
        PairSECCoord.z=currPos.z;
    }
```

Figure 13: Coordination fix

9. GENERAL

9.1 MILESTONES

Table 2: Millstones

No.	Outcome	Estimated completion date	Measurable product
1	Writing the pre report	12/12/2019	Pre report
2	Overview past project	17/1/2020	Underspending the project
3	Writing the progress report	24/1/2020	progress report
4	Design an algorithm to the problem	30/3/2020	algorithm
5	Code writing	30/4/2020	simulation
6	Debugging the code	10/6/2020	Working simulation
7	Presenting the project in the presentation's day	21/6/2020	Presentation and poster
8	Writing the final report	21/6/2020	Final report

9.2 TASKS

Table 3: Tasks

No.	Outcome	Estimated completion date	Days to complete	Measurable product
1	Receive theoretical background for algorithm	20/11/2019	20	Knowledge to start
2	Writing the pre report	12/12/2019	10	Pre report
3	Receive past project	13/12/2019	3	Code to work with
4	Finding and understanding the previous route solution	19/1/2020	36	Finding code parts that we need to modify
5	Writing the progress report	24/1/2020	5	Progress report
6	Design an algorithm	30/3/2020	65	Algorithm
7	Meeting with the project instructor	7/4/2020	7	Approval
8	Adjust the previous code to our algorithm	30/4/2020	20	Simulation
9	Debugging the code	20/5/2020	10	Working simulation
10	QA tests	31/5/2020	11	Verify edge cases
11	Adjust the code by the QA results	10/6/2020	10	Perfect simulation
12	Design a poster and presentation	15/6/2020	5	Poster & presentation
13	Writing the final report	21/6/2020	7	Final report

9.3 TOOLS AND PROGRAMS

9.3.1 Omnet++

OMNeT++ 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, queueing 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, database integration, SystemC integration, and several other functions. (3)

9.3.2 INET Framework

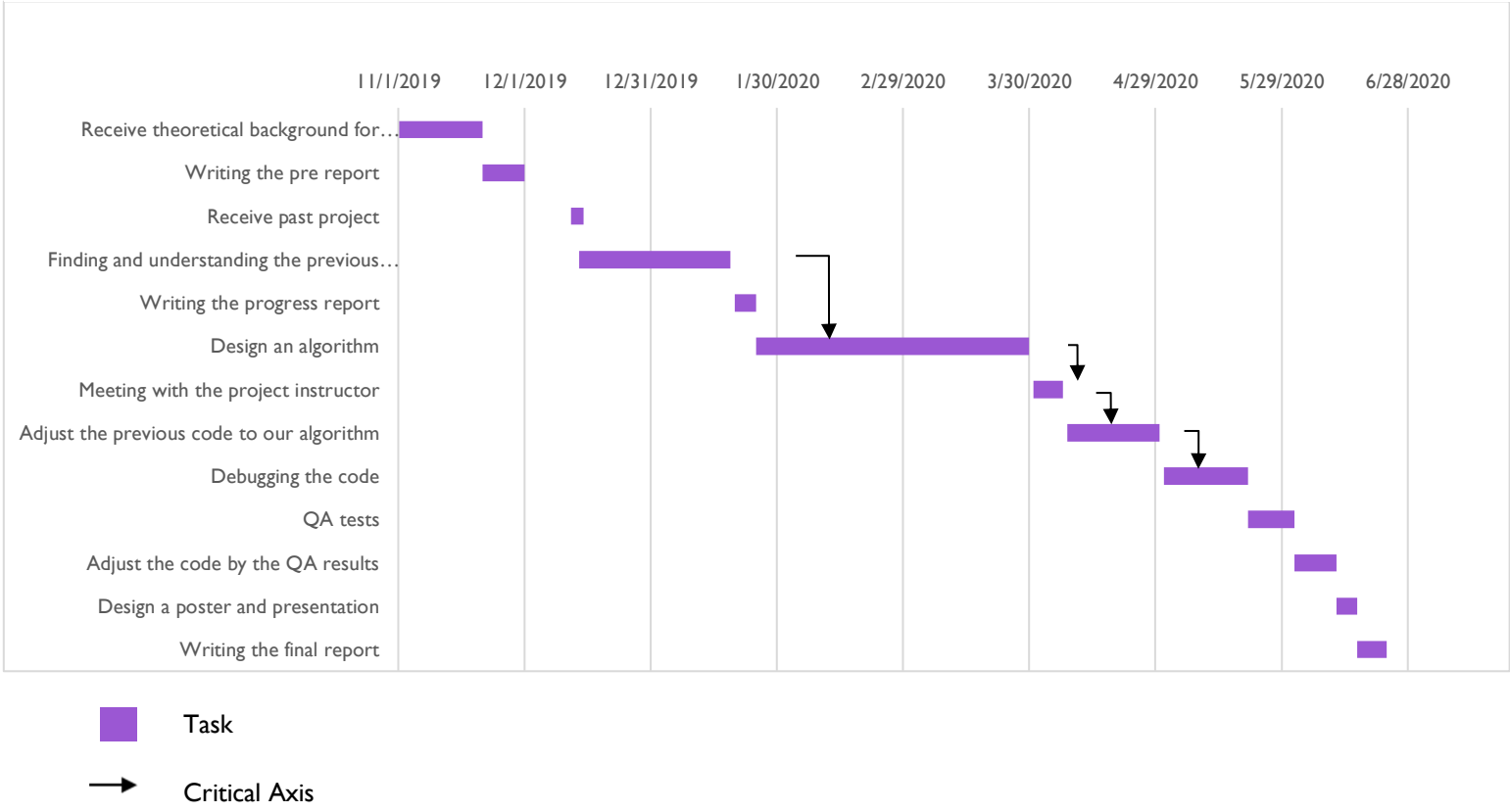
INET Framework is an open-source model library for the OMNeT++ simulation environment. It provides protocols, agents and other models for researchers and students working with communication networks. INET is especially useful when designing and validating new protocols, or exploring new or exotic scenarios.

INET contains models for the Internet stack (TCP, UDP, IPv4, IPv6, OSPF, BGP, etc.), wired and wireless link layer protocols (Ethernet, PPP, IEEE 802.11, etc), support for mobility, MANET protocols, DiffServ, MPLS with LDP and RSVP-TE signaling, several application models, and many other protocols and components.

Several other simulation frameworks take INET as a base, and extend it into specific directions, such as vehicular networks, overlay/peer-to-peer networks, or LTE. (4)

9.4 GANTT TABLE

Table 4: Gantt table



10. RESULTS

10.1 SIMULATION SCREENSHOTS

10.1.1 FindMsgForEstablishSubGraph- Connectivity

In Figure 14: establish sub graph message and in Figure 15 : establish sub graph message we can see how the message diffuses in the network. Those cases are extra connectivity drone.

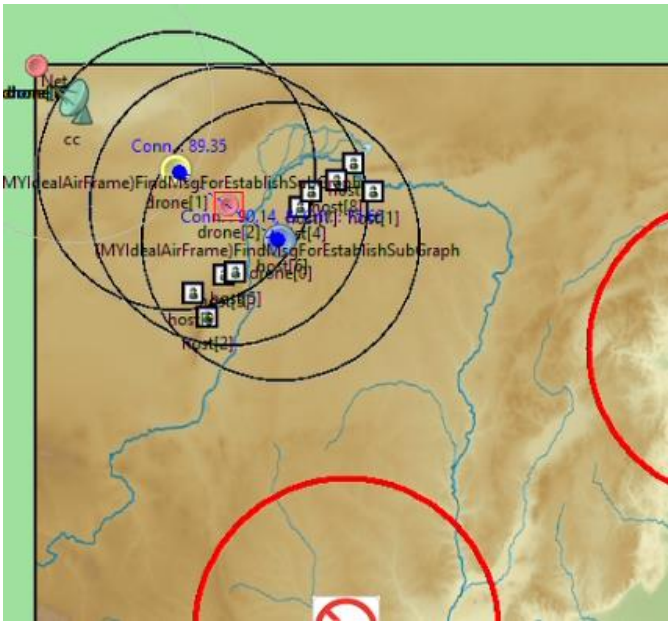


Figure 14: establish sub graph message

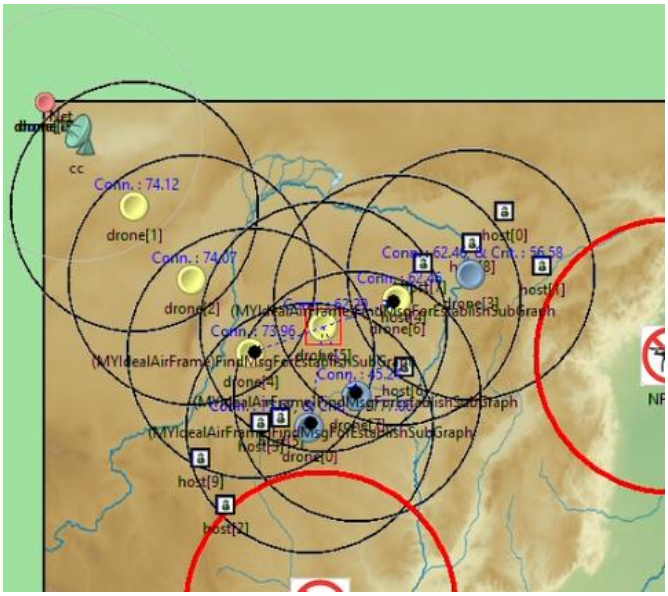


Figure 15 : establish sub graph message

10.1.2 FindMsgForEstablishSubGraph- Regular

In Figure 16: establish sub graph message and in Figure 17: establish sub graph message we can see how the message diffuses in the network. Those cases are extra connectivity drone.

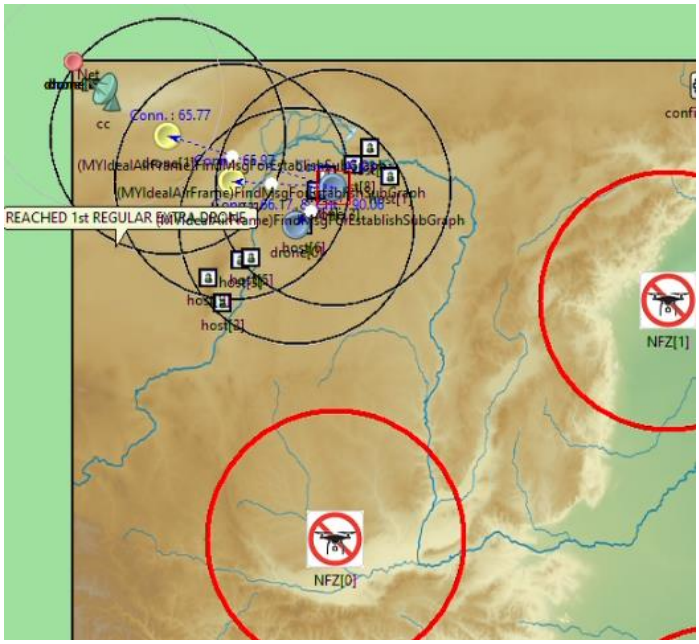


Figure 16: establish sub graph message

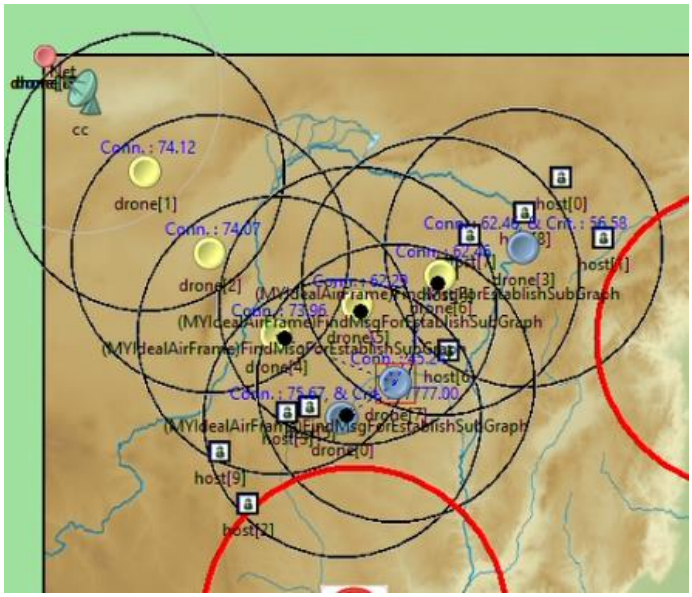


Figure 17: establish sub graph message

10.2 STATISTICAL RESULTS

10.2.1 Hop Count

The time needed for a message to be passed from a soldier (host) to control center (CC) is depend on the number of drones that the message needs to pass to reach the CC. Therefore, the hop count measurement is critical and we tried to decrease it. We calculated the number of hosts covered by drones multiplied by the number of hops between the drone that covers them to the CC. As we can see in Graph 1: Hop count and in Graph 5: Hop Count, which have a different seed (random seed is a number used to initialize a pseudorandom number generator),the graphs show a comparison between simulation with the functionality of establishing sub graph, and without this functionality. In the graph we also added the average hop count, so we see that while the times that the topology is stable the functionality of the sub graph improves the average hop count from host to the CC.

In Table 4: Drone-Host cover without establish sub graph and Table 6: Drone-Host cover with establish sub graph there is the original data used for building Graph 1: Hop count and Graph 5: Hop Count

10.2.2 Number of Drones

During the simulation we saw that while using the functionality of establish sub graph the total number of drones needed for keeping connectivity in the network was smaller or even to the number of drone without this functionality, as shown in Graph 2: Number of drones and in Graph6 : Number of drones. Obviously, the less drones you use it is more economic, and you exploit your resources better.

10.2.3 Convergence time

As in most engineering problems we met, there is a trade-of. In our case we saw improvement as we shown above, but it cost in time that it takes for the network to determine the new sub graph. In Graph 3: Convergence time, Table 5: Simulation times for Convergence , and in Graph 7: Convergence time Table 7: Simulation times for Convergence we show the times that took for the network to converge.

The total simulation time was 80.02. The overall time that the establishing sub graph added for the simulation was 6.41E-06, which is 8.0125e-8% of the simulation time.

The total simulation time was 80.02. The overall time that the establishing sub graph added for the simulation was 4.63E-06, which is 5.78605349e-8% of the simulation time.

In both shown simulation results the added time is minor.

10.2.4 Message count

Another tradeoff of our functionality is redundancy in administration messages. As we can see in Graph 4: Message count and in Graph 8: Message count , the load of our messages is about 40% of all administration messages, but since the convergence time is still negligible the added overhead improvement in the network.

10.3 ATTACHED GRAPH FOR SIMULATIONS

10.3.1 First Simulation

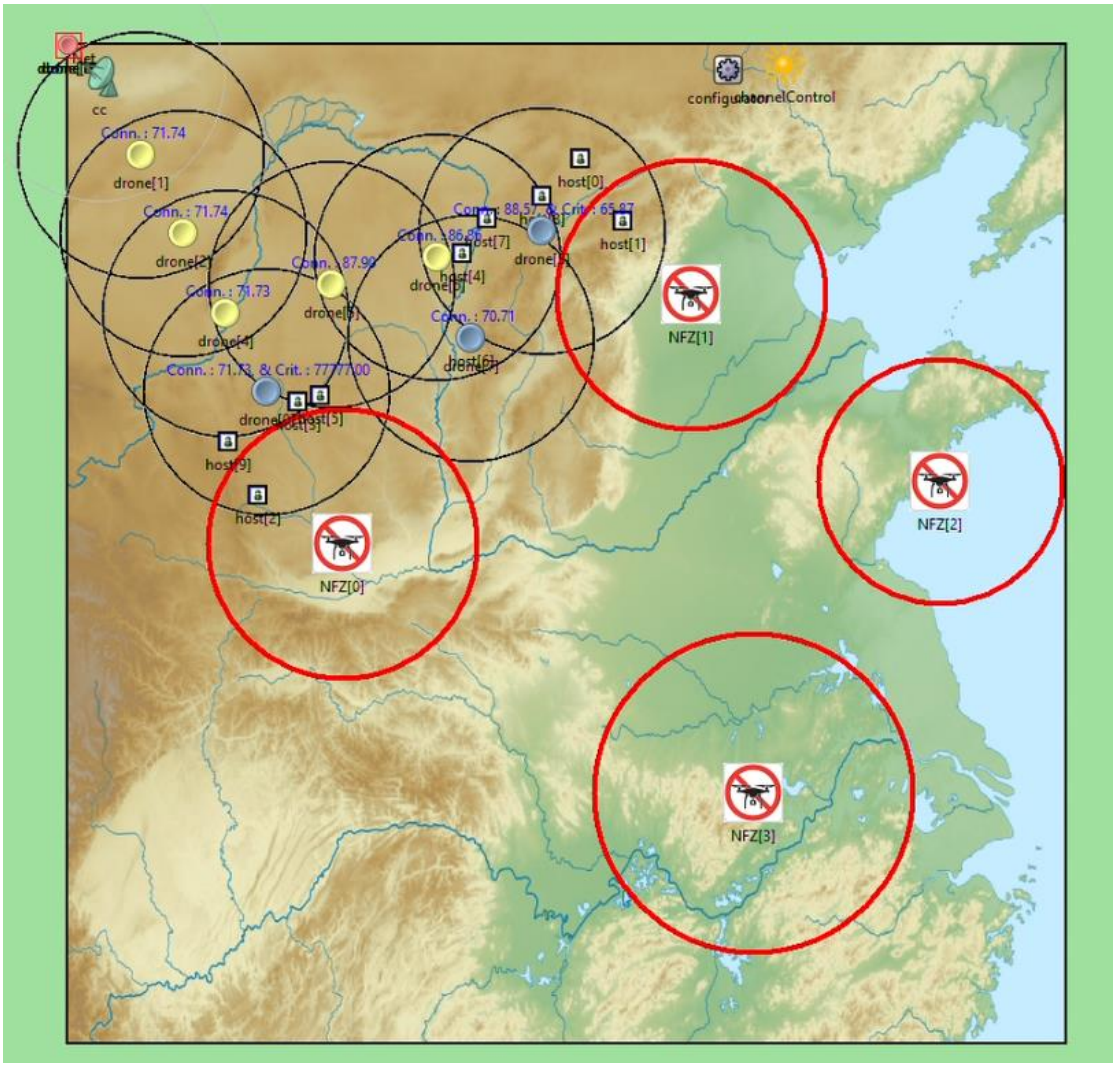


Figure 18 : Simulation topology

Topology parameters:

Number of drones: 15

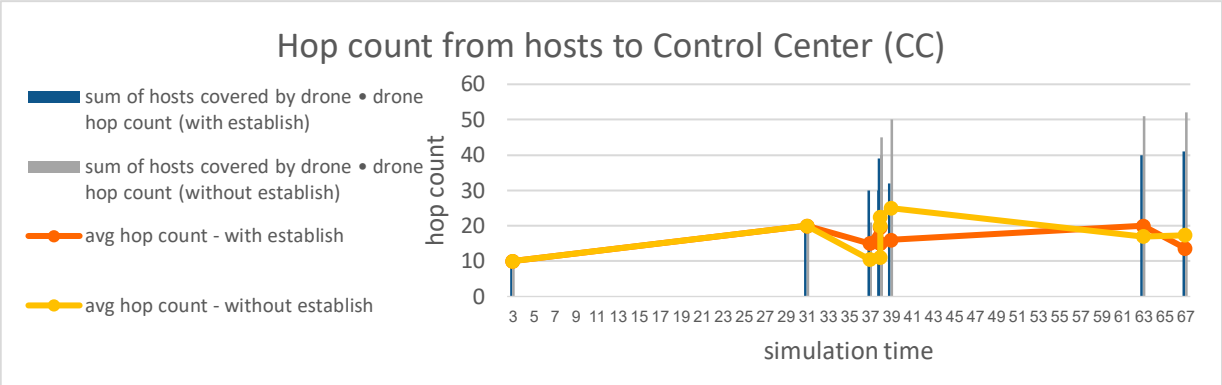
Number of hosts: 10

Seed-set: 2

Transmission range: 100m

Map size: 800X800 m^2

HOP COUNT

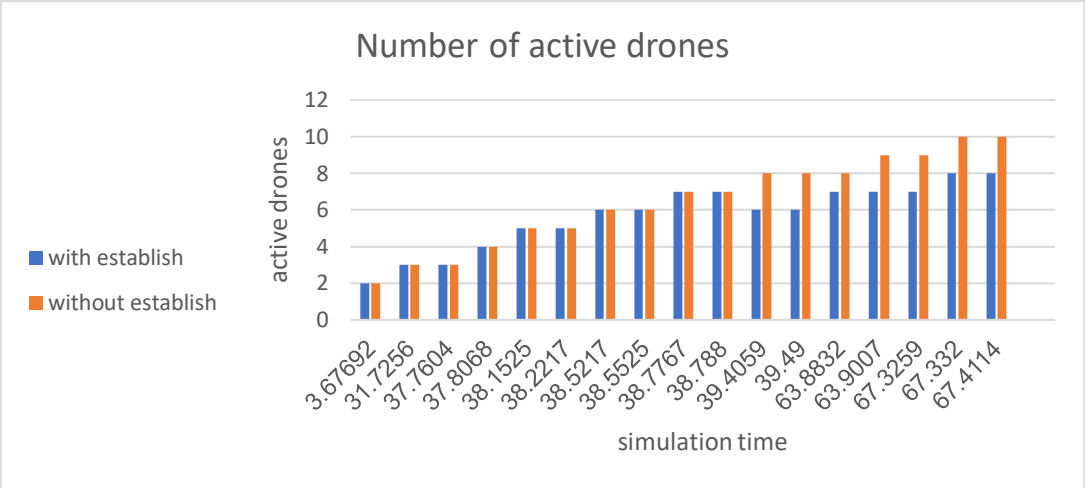


Graph 1: Hop count

Table 4: Drone-Host cover without establish sub graph

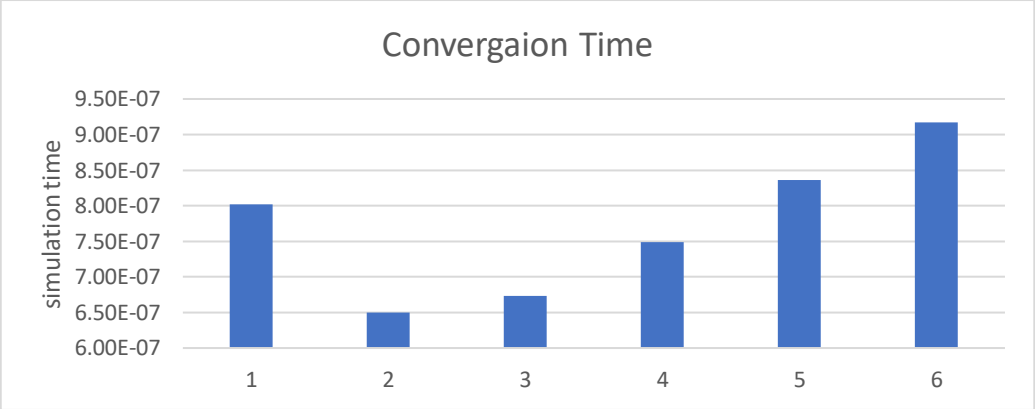
name	owner count	hop count	sim time	avg hop count	num of hosts covered by drone • drone hop count
drone[0]	10	1	3.67692	10	10
drone[0]	10	2	31.7256	20	20
drone[3]	1	3	37.7604	10.5	21
drone[0]	9	2	37.8068		
drone[0]	9	2	38.4217	11	22
drone[3]	1	4	38.4525		
drone[0]	5	3	55.9653	20	40
drone[3]	5	5	55.9714		
drone[0]	5	3	63.3119	22.5	45
drone[3]	5	6	63.4086		
drone[3]	5	7	63.5086	25	50
drone[0]	5	3	63.5119		
drone[8]	1	4	75.7579	17	51
drone[0]	4	3	75.7615		
drone[3]	5	7	75.8039		
drone[0]	4	3	76.4581	17.33333333	52
drone[8]	1	5	76.474		
drone[3]	5	7	76.5197		

NUMBER OF DRONES



Graph 2: Number of drones

CONVERGENCE TIME

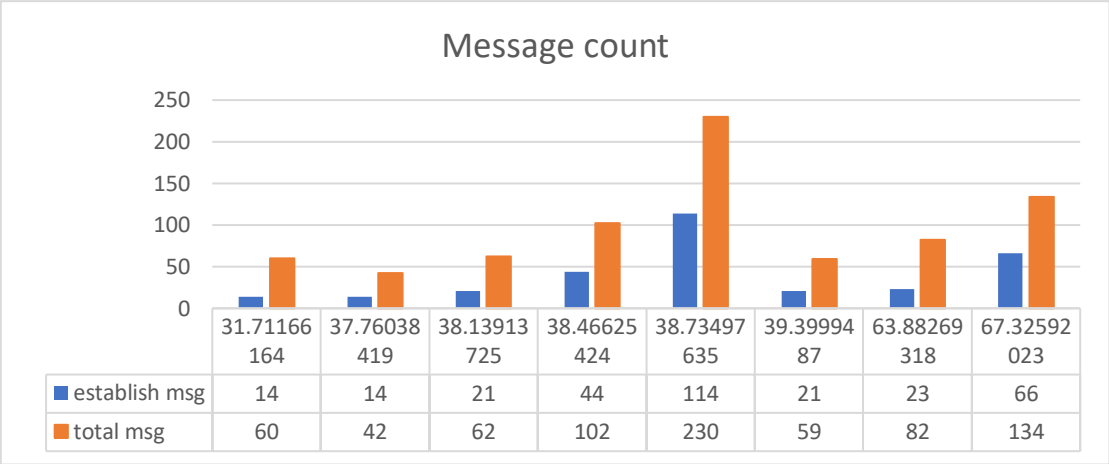


Graph 3: Convergence time

Table 5: Simulation times for Convergence

	Start	Finish	Conv. time
1	31.71166155	31.71166236	8.1E-07
2	37.76038408	37.76038477	6.9E-07
3	38.13913717	38.13913789	7.2E-07
4	38.46625422	38.46625504	8.2E-07
5	38.73497632	38.7349773	9.8E-07
6	39.39994861	39.39994935	7.4E-07
7	63.88269306	63.88269387	8.1E-07
8	67.32592013	67.32592097	8.4E-07

MESSAGE COUNT



	31.71166164	37.76038419	38.13913725	38.46625424	38.73497635	39.39994 87	63.88269318	67.32592023
establish msg	14	14	21	44	114	21	23	66
total msg	60	42	62	102	230	59	82	134

Graph 4: Message count

10.3.2 Second Simulation

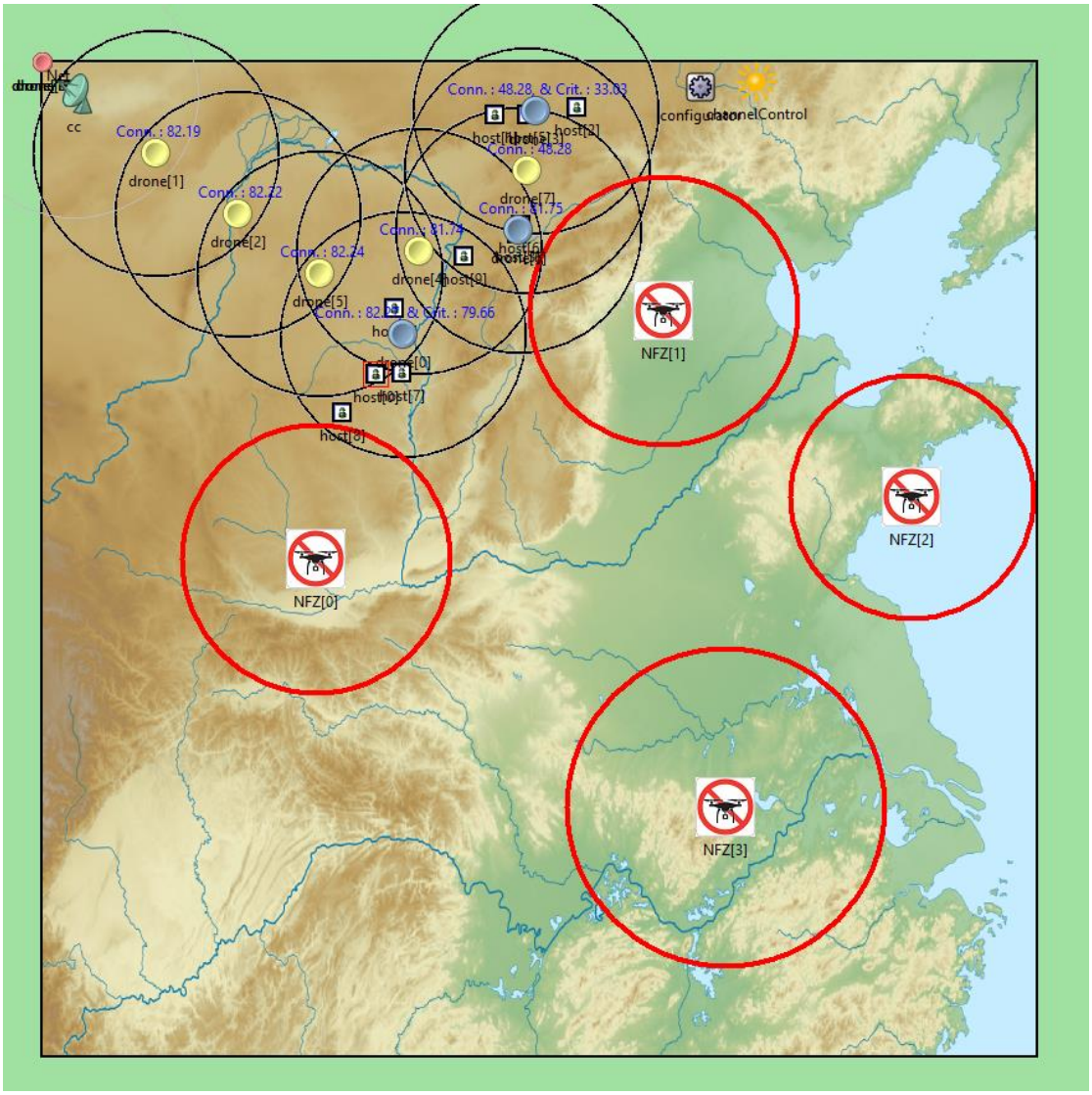


Figure 19: Simulation topology

Topology parameters:

Number of drones: 15

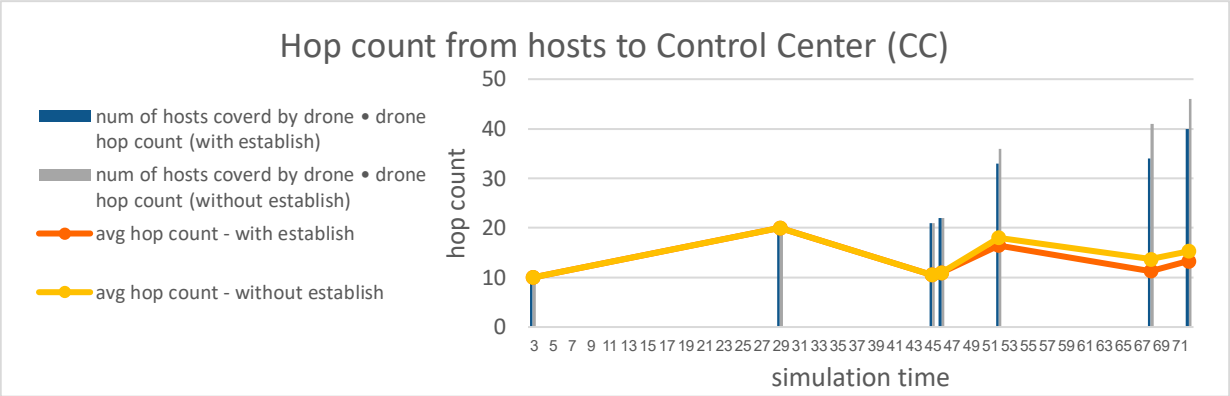
Number of hosts: 10

Seed-set: 64

Transmission range: 100m

Map size: 800X800 m^2

HOP COUNT

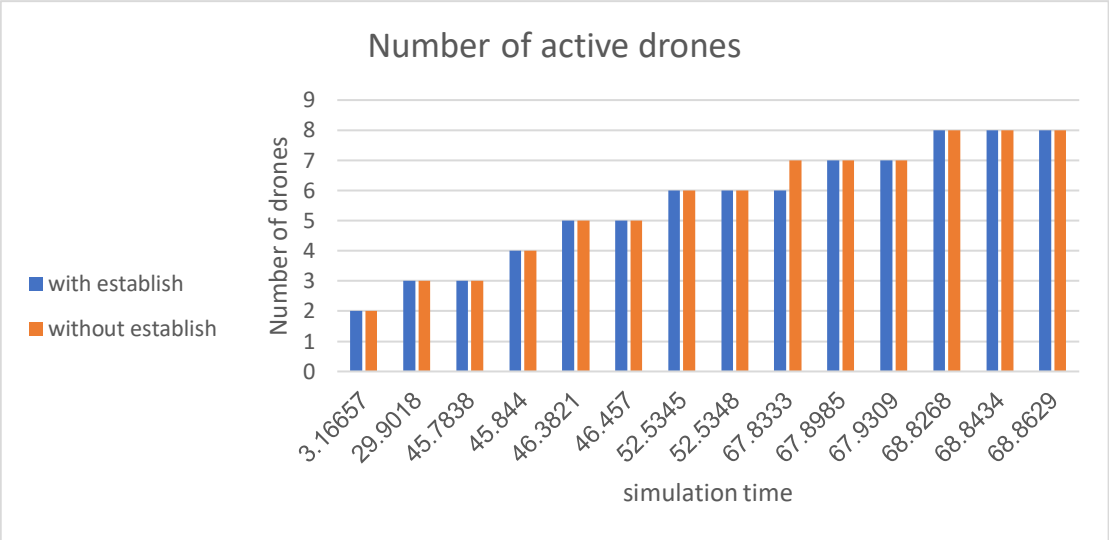


Graph 5: Hop Count

Table 6: Drone-Host cover with establish sub graph

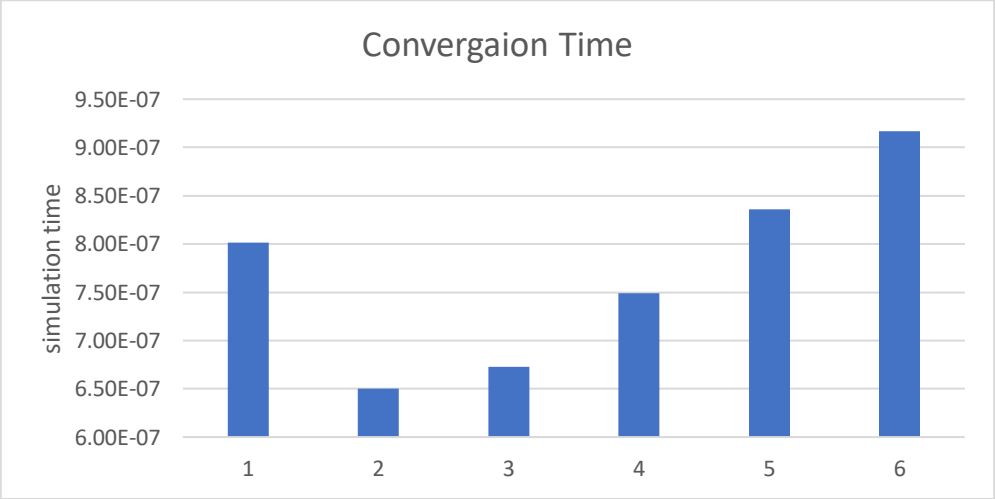
name	owner count	hop count	sim time	avg hop count	num of hosts coverd by drone • drone hop count
drone[0]	10	1	3.67692	10	10
drone[0]	10	2	31.7256	20	20
drone[3]	1	3	37.7604	15	30
drone[0]	9	3	37.8068		
drone[3]	1	3	38.1525	19.5	39
drone[0]	9	4	38.2217		
drone[0]	9	3	38.5217	15	30
drone[3]	1	3	38.5525		
drone[0]	8	3	38.7767	15	30
drone[3]	2	3	38.788		
drone[3]	2	4	39.4059	16	32
drone[0]	8	3	39.49		
drone[0]	5	3	63.8832	20	40
drone[3]	5	5	63.9007		
drone[7]	1	4	67.3259	13.66666667	41
drone[0]	4	3	67.332		
drone[3]	5	5	67.4114		

NUMBER OF DRONES



Graph6 : Number of drones

CONVERGENCE TIME

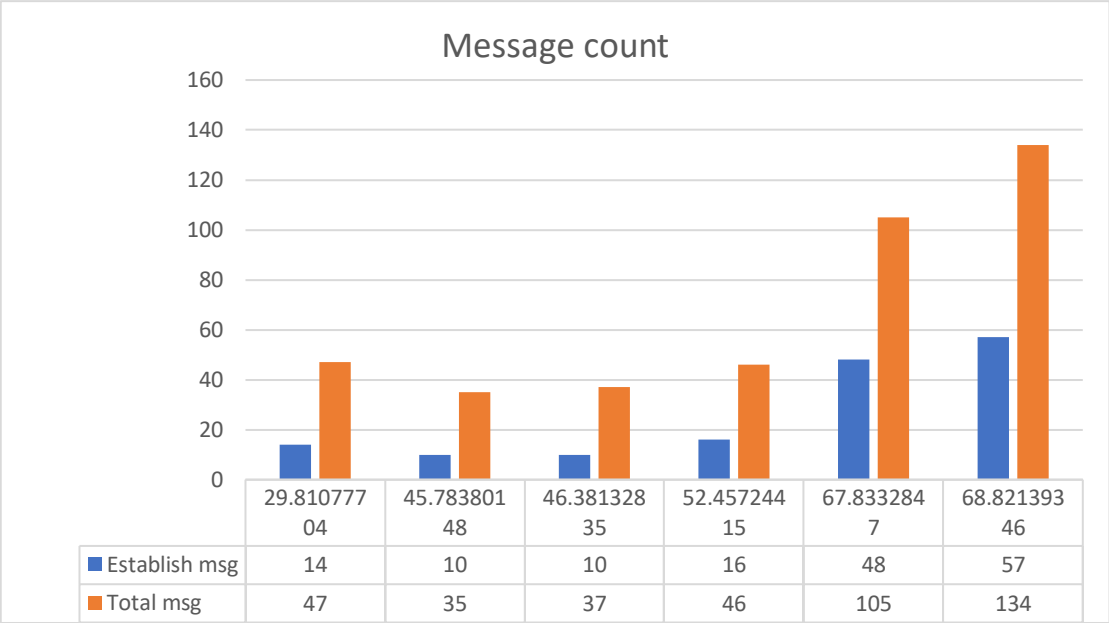


Graph 7: Convergence time

Table 7: Simulation times for Convergence

	start	finish	conv time
1	29.81077695	29.81077775	8.02E-07
2	45.78380132	45.78380197	6.50E-07
3	46.3813282	46.38132887	6.73E-07
4	52.45724404	52.45724478	7.49E-07
5	67.83328457	67.8332854	8.36E-07
6	68.82139335	68.82139426	9.17E-07

MESSAGE COUNT



	29.81077704	45.78380148	46.38132835	52.45724415	67.8332847	68.82139346
Establish msg	14	10	10	16	48	57
Total msg	47	35	37	46	105	134

Graph 8: Message count

11. SUMMARY

11.1 WORK PARTITION

While working on the project we have tried to split the work between as, but in short time we have discovered that the integration process is match more complex and takes too much time rather than work together on the code. Since the COVID19 time we have tried working together using the ZOOM application and after about a month of almost no progress we have decided to come back to face to face meetings and we saw immediately better performance.

11.2 SKILLS ACHIVED FROM CSE DEPARTMENT

While working on the project we used knowledge achieved in many of the department courses:
we wrote in C++ language that we learned in *Advanced Programing* class, we needed to use some *Data Structures* to keep the drones information. The topology is a graph that we learned how to manage in *Graph Theory*. Our project issues a communication problem and therefore we needed to use knowledge received in *Communication network 1* and *Communication network 2* we added headers to manage the network, and of course the network is a wireless network and knowledge received in *Wireless networks*. Each drone is independent from another, and therefore it distributed and the knowledge on the right way to work on such networks learn in *Distributed Algorithms*. In *Ad-Hoc Algorithms* we learned tools for handle distributed graphs and more specifically Proactive Routing Protocols. And finally, in *simulation* classes we used the OMNET++ environment and achieved the ability to work with.

As described above we achieved many different skills in the degree that was necessary to implement our project. After completing the project we have improved our programing skills, and we have a better understanding on how to implement a theoretical idea into a working simulation, while implementing an algorithm we suddenly needed to take in consideration edge cases of the system that our functionality needed to be blend in.

11.3 THE PROJECT IN NUMBERS

The time devoted during the semester was 2-3 days a week for about ten hours a day. And after the semester was over we met 6 days a week also for ten hours a day. Which is about 1000 hours in total in both semesters.

The implementation of our functionality was about 1200 code lines, that was scattered in 12 different files that each one of them was in charge of different aspect in the project. The attached codes are the main parts in the algorithm.

Our project was done by us (Tslil and Hila) but of course we used guidance from Prof. Michael Segal and from Mr. Kiril Danilchenko. Also we consulate our student colleagues.

12. REFERENCES

1. *Maintenance of fully-distributed UAV network -Final Report.* **Melamed, Tamir and Meir, Adi.** s.l. : Meir, Adi; Melamed, Tamir, 2016.

2. *A Review and Classification of Flying Ad-Hoc Network (FANET) Routing Strategies .* **Adnan Nadeem, Turki Alghamdi, Ali Yawar, Amir Mehmood, Muhammad Shoaib Siddiqui.** 2018, Journal of Basic and Applied.

3. **OMNeT++ introdaction.** ***http://omnetpp.org/intro./.*** **[Online]**

4. INET framework introdaction. ***http://inet.omnetpp.org/Introdaction.htm.*** **[Online]**

5. PRASANT MOHAPATRA, SRIKANTH V. KRISHNAMURTHY. ***AD HOC NETWORKS Technologies and Protocols.*** **University of California : Springer Science, 2005.**

6. Unit disk. ***Wikipedia.*** **[Online] https://en.wikipedia.org/wiki/Unit_disk.**

7. ***Efficient Algorithms for Constructing Very Sparse.*** **NEIMAN, MICHAEL ELKIN and OFER. s.l. : ACM Transactions on Algorithms, 2018, Vol. 15.**