FINAL PROJECT – BATMAN PROTOCOL

CONCURRENT AND DISTRIBUTED PROGRAMMING

Erlang

Amit Nagar- Halevy and Tal Kapelnik
204210306 204117089

Dr. Yehuda Ben-Shimol
Mr. David Leon

BGU
COMPUTER ENGEENIRING

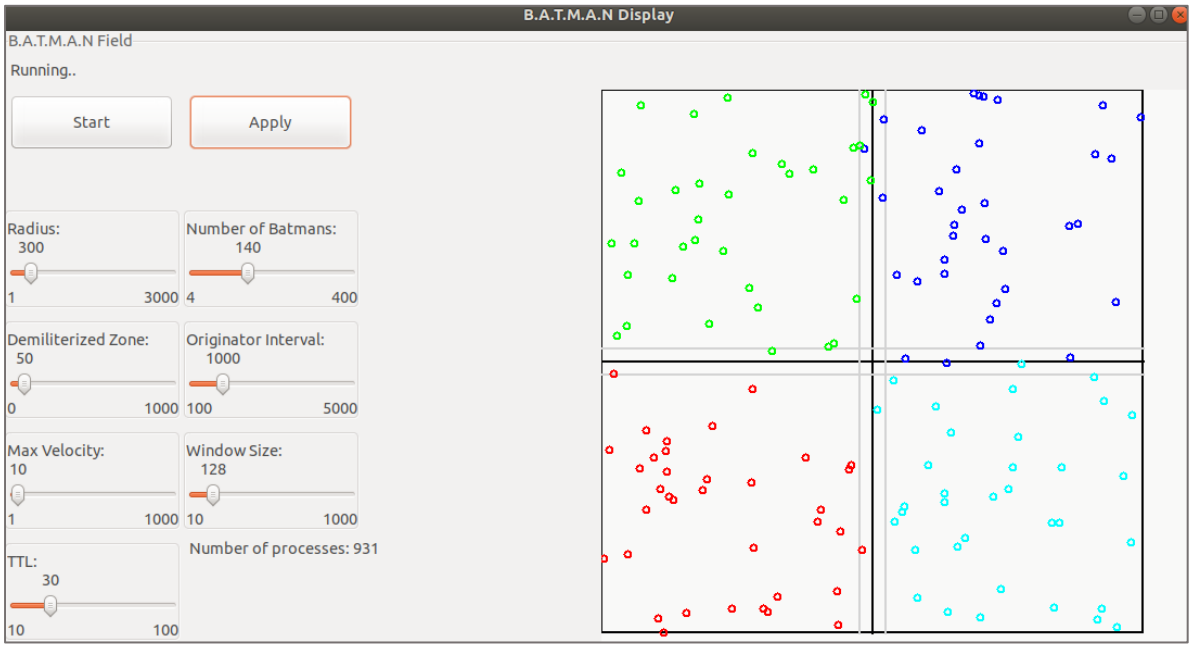
Contents

Assignment Description:

this project is simulating an envierment of bi-directional links that transfer messages from one to another via BATMAN protocol. it can work on 2-5 different nodes (computers).

B.A.T.M.A.N. Protocol:
B.A.T.M.A.N. is a proactive routing protocol for Wireless Ad-hoc Mesh Networks. The protocol proactively maintains information about the existence of all nodes in the mesh that are accessible. The strategy of B.A.T.M.A.N. is to determine for each destination in the mesh one single-hop neighbor, which can be utilized as best gateway to communicate with the destination node.
To learn about the best next-hop for each destination is all that the B.A.T.M.A.N. algorithm cares about. There is no need to find out or calculate the complete route, which makes a very fast and efficient implementation possible.

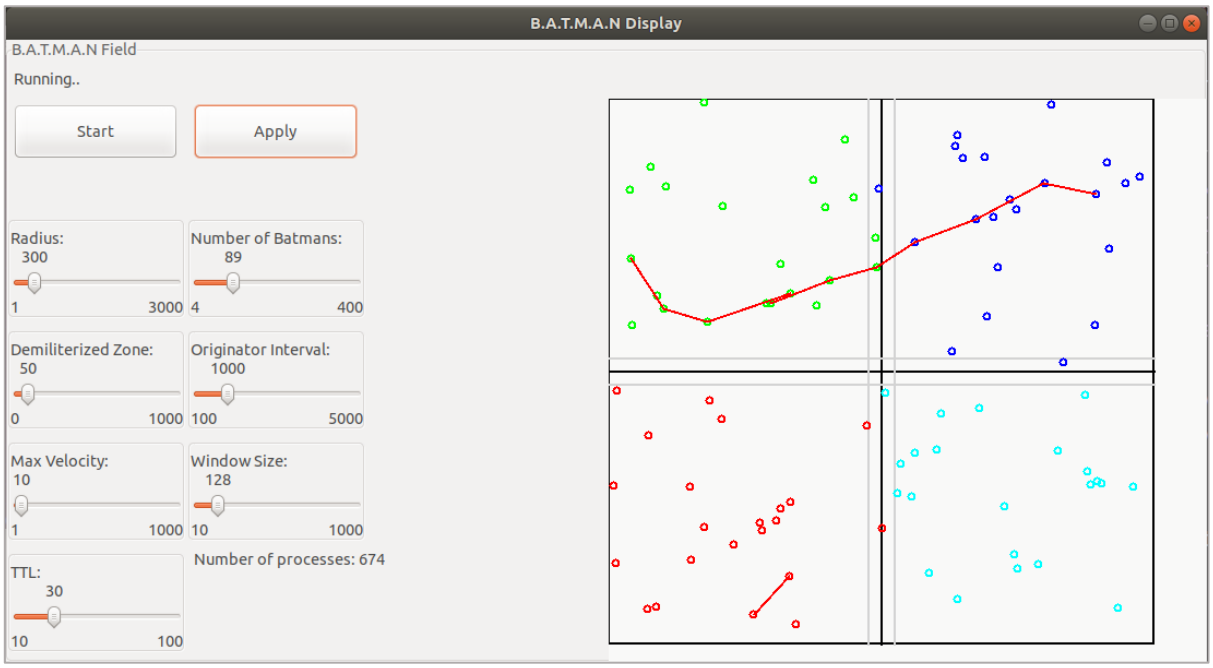
The User Interface:



each colored Circle represents a bi-directional link, that is moving in an open space of 2000 by 2000 meters. Each color represents a different computer node.
Each link (circle), as will be called from now “Robin”, maintained by two servers:

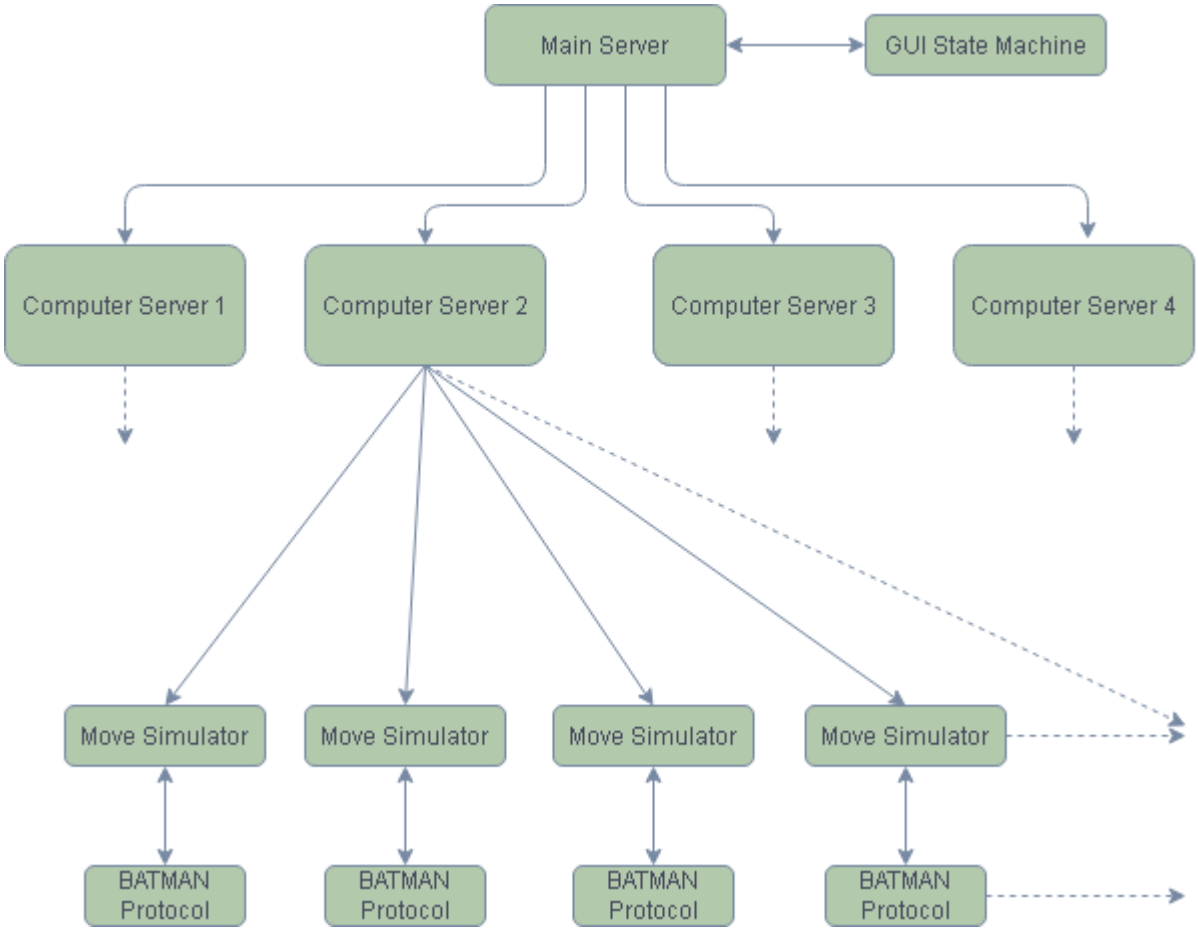
1. gen_server - “MoveSimulator”, this server is responsible for movement and simulation of communication. He is responsible for simulating random movements in different speeds an directions. He is responsible for sending all kind of messages between different Robins.
2. gen_server – “BatmanProtocol”, this server is responsible on the implementation of the B.A.T.M.A.N protocol.

Every 2 seconds the Main Server pick 2 random Robins and send a message from one to the other using the BATMAN protocol, these messages path is represented in red lines as follow:



The black lines represent the boarders.
The grey lines represent a Delimitation zone. In this zone processes from different computers can live without before moving to the other computer. This zone prevents frequent movement of a Robin between computers.

Assignment Plan:



We have a Main Server, he opens 4 (or less) Computer Servers, and a GUI state machine.
Each Computer Server opens Move Simulators. and each of those opens a Batman Protocol Server.
Move Simulator Server and a Batman Protocol Server are represented with one colored circle, a Robin.
We will explain each server job and responsibilities.

Batman Protocol server:

This server is the hearth of the project. This server implements the BATMAN protocol. The main mission of this protocol is to give us the Best Link, the most promising next hop towards a given Robin.

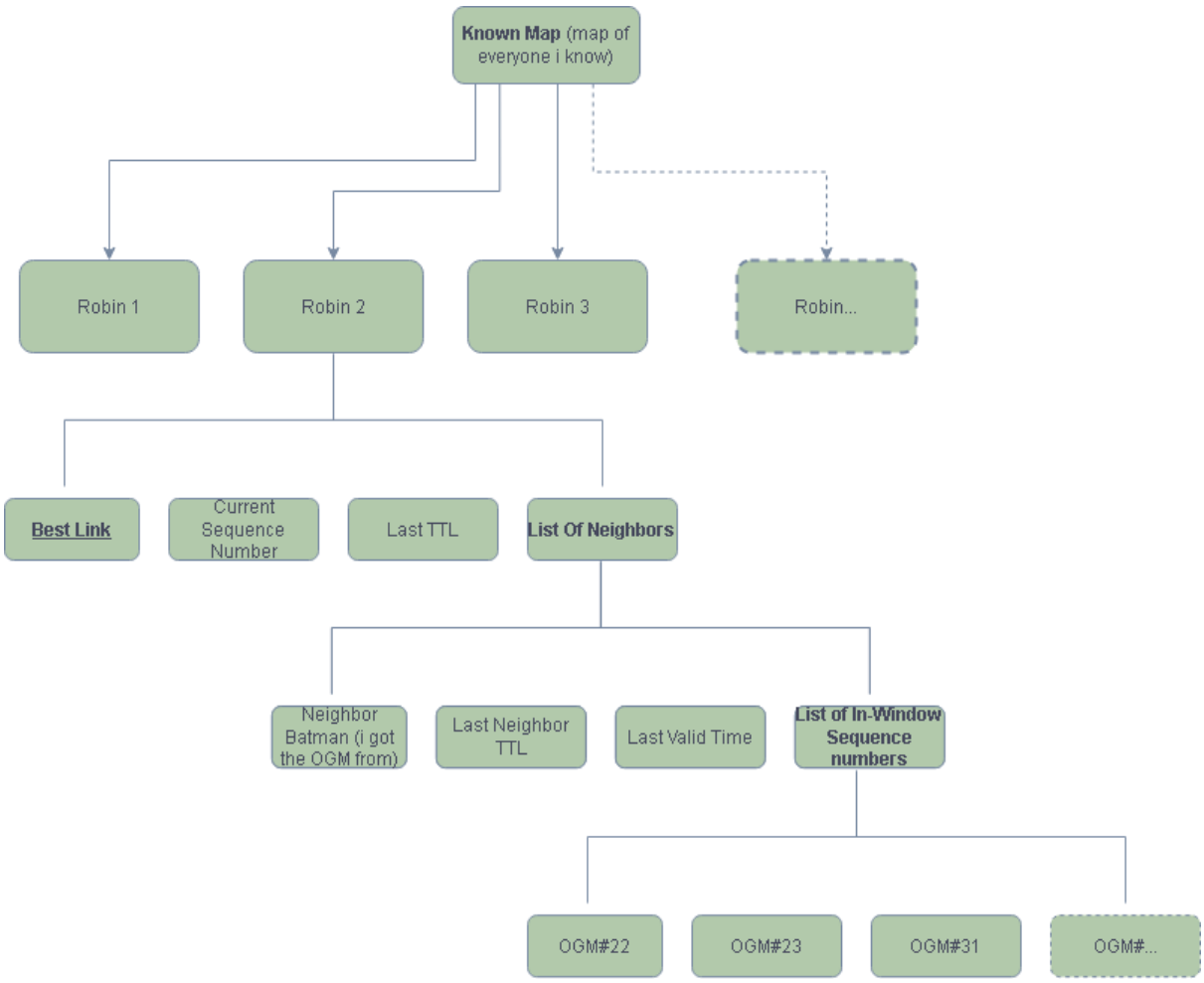
Every Originator Interval, it sends an Originator Message (OGM) to all of his neighbors (the other Robins in the given Radius).

The OGM contains:
Sequence Number- a number that increases every OGM, it makes the OGM unique.
TTL- Time To Live, how many Robins can pass the OGM until it dies.
Robin Address- who the original sender of the OGM.

Receiving OGM:
Each OGM it receives from other Robin, it will decide if it needs processing and/or rebroadcast it to my neighbors.

Rebroadcasting an OGM- sending the OGM to the neighbors with TTL changed to TTL-1. If the TTL now is 0 it will not rebroadcast.

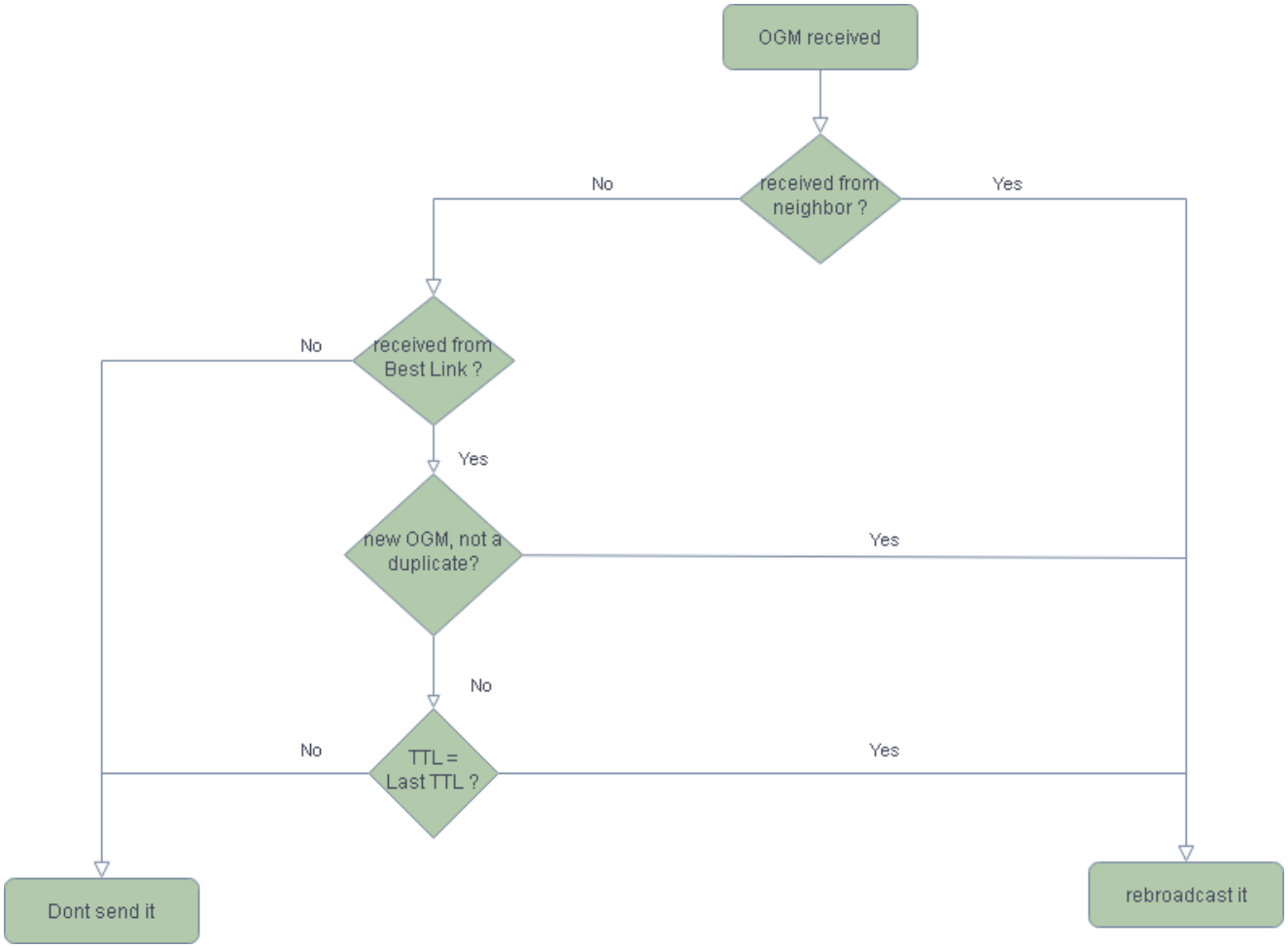
if the OGM needs processing we will update this map, Known Map, that each Robin has.



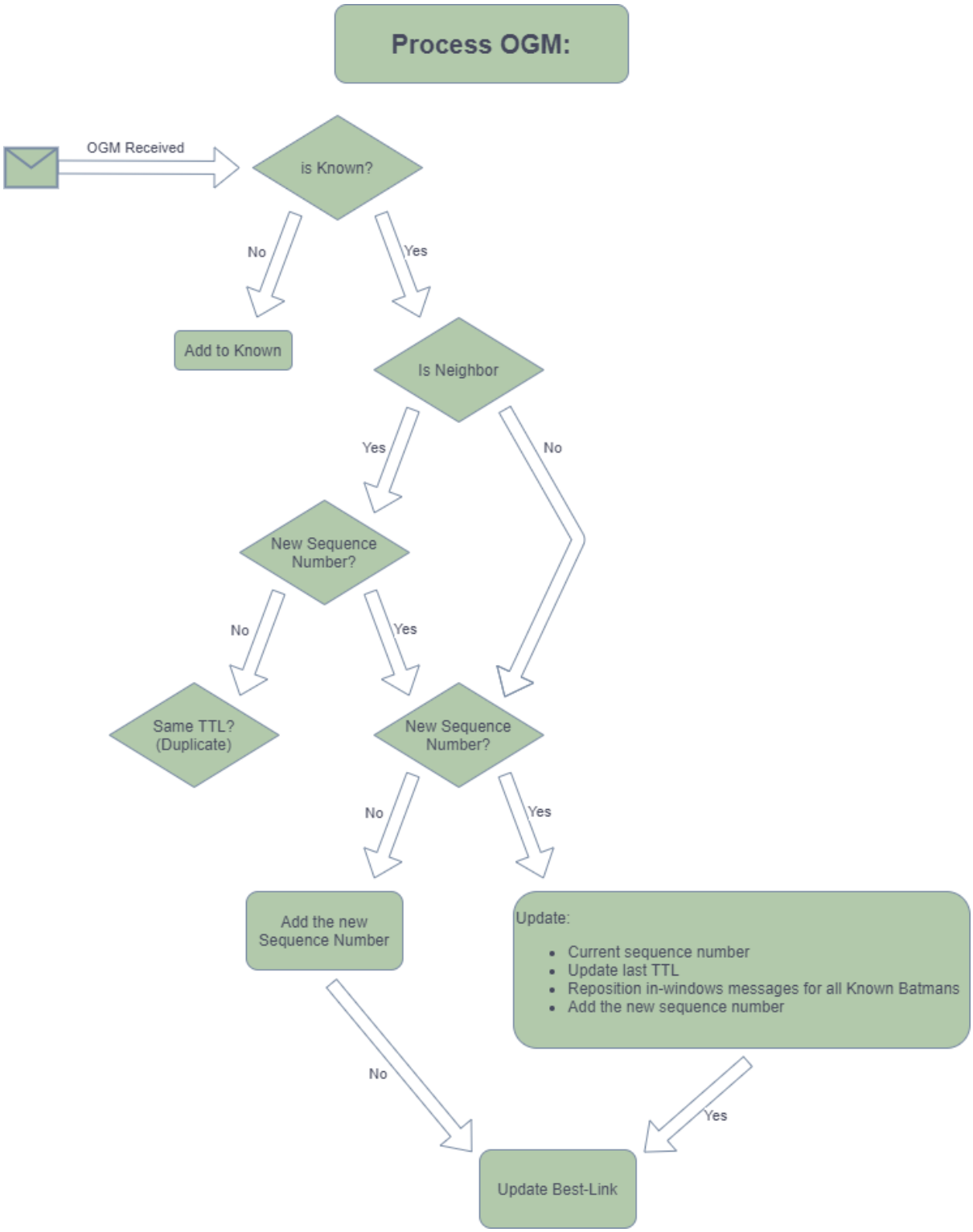
The Known map contains all the Robins this Robin got an OGM from.
Each Known Robin in the map, contain the Best Link, this is the best neighbor to send a message to this Known Robin. The Best Link is decided by the neighbor in the List Of Neighbors that has the longest List Of In-Window Sequence Number.

List Of In-Window Sequence Number-
represents the latest sequence numbers. They include the Current Sequence Number from this Robin and the WindowSize - 1 Sequence Numbers below it.

should an OGM should be rebroadcast:



Processing an OGM:



Batman Server		
Message Name	**Type**	
findBestLink, To	**Call**	finds the best link and returns it
getKnownFrom	**Call**	Returns a random pid from Known map
sendOGM	**Cast**	Every OGM_INTERVAL seconds, an OGM is sent to all nearby Pids in radius
ogm,{SeqNum, TTL,OriginatorAddress},OriginatorAddress	**Cast**	OGM recieved from direct neighbor 1. process OGM – update all necessary in Known map 2. rebroadcast OGM – send to all nearby neighbors in radius
ogm,{SeqNum, TTL,OriginatorAddress},FromAddress	**Cast**	OGM recieved not from direct neighbor 1. process OGM – update all necessary in Known map 2. rebroadcast OGM ONLY if the protocol demands. If so – forwards the OGM to all nearby neighbors in radius
deleteNeighbor, ToDelete,AddressFrom,Msg	**Cast**	Msg failed: delete neighbor from List of Neighbors
tokillonshutdown,Pids	**Cast**	Pid = All the pids of helping processes
stop	**Cast**	Stopps the server and shut down all connected processes

Move Simulator Server:

This server responsible on the movement and communication with other Robins.

Movement:

Every random time (within a range) it will generate new Direction and Velocity.
it will update the EtsX and EtsY accordingly (will be explained further in Computer Server).

Communication:

When an *OGM* need to be sent to the neighbors it will calculate the neighbors in my radius.
For the Robins on my Computer it will send them the OGM directly,
if my location indicates that I may have neighbors on other computers (close to the boarder),
then it will send to my Computer Server the OGM to handle it.

When a *Message* needs to be sent it will ask for the best link and do the same,
If the Message didn't receive by my Best Link, it will delete him from List Of Neighbors and send it to the Next Best Link.

MoveSimulator Server		
Message Name	**Type**	
receiveMsg,To,Msg,MoveSimFrom	**Call**	Receive a message, forward it if its not addressed to me.
getKnownFrom	**Call**	Returns a pid from batmans Known map
sendMsg,To,To,_Msg	**Cast**	send MSG to someone using the BATMAN Protocol
sendMsg,To,From,Msg	**Cast**	Sends a message to To, first look for a best link, if there was a problem with the best link: 1.delete the neighbor 2.send the msg again to the next best link
updateBorders,NewStartX,NewEndX,NewStartY,NewEndY	**Cast**	Updates the borders in State record.
updateMovementVector,CurrTime,Velocity,Direction	**Cast**	Update movement specs in State record.
updateEts	**Cast**	updateEts updates the location of my PID in the etsX and etsY. also checks if the borders are ok or should i terminate and move to another nearby PC
sendToNeighbors,OGM	**Cast**	sends OGM to all neighbor pids in radius
ogm,OGM,{Pid,Node}	**Cast**	OGM received, send it to the Batman
tokillonshutdown,Pids	**Cast**	Pid = All the pids of helping processes
stop	**Cast**	Stopps the server and shut down all connected processes

Computer Server:

The Computer Server is the main server of each node (except of the Main Server node himself).
It spawns and monitors all the Robins in his area.
It is responsible for communication with the other Computer Servers.
It will update the Main Server on the Location of his Robins, and the Messages that was send.
It maintains 2 Ets's:
EtsX and EtsY- each contains all the Robins Addresses at a particular X or Y.
It supports changing his boarders when other node crashes.

Computer Server		
Message Name	**Type**	
sendLocations	**Call**	Sends etsX and etsY to the MainServer
sendMsg,To, {FromNeighborPid,FromNeighborNode},Msg,MoveSimFrom	**Call**	Sends a message to another robin, waiting for a reply 'received'
updateMyMonitor,Mymonitor	**Call**	Update the monitors pid in state
receiveMsg,To, {FromNeighborPid,_FromNeighborNode},Msg,MoveSimFrom	**Call**	recieved the msg from **neighbor Computer** and send it to the right pid
updateBorders, {X,Y,Dir,Vel}	**Call**	Move simulator wants to know the new borders, if he is out of the area, he should terminate and a new move simulator should be created in the computer, else the new borders will be sent
getKnownFrom, PidFrom	**Call**	Returns {PidTo,NodeTo} from batmans known(PidFrom)
getProcesses	**Call**	Returns the number of processes on my node
createBatman,{X,Y,Dir,Vel}	**Cast**	spawn a Robin and monitor it, we add the DemilitarizedZone, so the moveSimulator will know it
sendOGMtoNeighbors,MyX,MyY,OGM,{PidFrom,NodeFrom}	**Cast**	Sending an OGM to nearby robins, also sends the ComputerServer a cast to forword it to the nearby PC if im close to the border.
ogmFromNeighbor,MyX,MyY,OGM,{PidFrom,NodeFrom}	**Cast**	Receiving an OGM from other robin. Need to forword it to BATMAN to process
monitorMe,From	**Cast**	Sends a addRobin message to myMonitor to add to monitored processes
removeRobin,Pid	**Cast**	removes Robin from ETS and cast mainServer to do so
msgSent,From,To	**Cast**	When receiving a message, notify the MainServer

generateRobin	**Cast**	Robin crashed so we generate a new one
newBoarders,NewComputerNodes,NewComputerAreas	**Cast**	someone died, update Boarders, if my boarder updated change it too
newRobinsAtXY,ListOfXY	**Cast**	spawn new Robins at each {X,Y} from ListOfXY
tokillonshutdown,Pids	**Cast**	Pid = All the pids of helping processes
stop	**Cast**	Stopps the server and shut down all connected processes

GUI State Machine:

When Starting, Initiating WX environment, creating a Frame, a canvas to paint on, and all the widgets needed. updating all necessary in guiStateM and then start the machine with first state waiting. Every RefreshTime defined as 14 fps, a do_refresh function is called and a wxBufferedPaintDC is created. We picked this DC to eliminate flickering causes from destroying and creating a ClientDC. In this way, a bitmap is created in the background, and the next frame is getting ready BEFORE the current frame is destroyed, this method smooths the frame transitions.

GUI State Machine				
Message Name	**Type**	**State**	**Next State**	
paintnow	**cast**	**Waiting**	**paint**	Waiting for the user to hit a button to start.
sendnewStats, Env	**cast**	**Waiting**	**paint**	when the user hits apply, start drawing.
refresh,ETS	**cast**	**paint**	**paint**	paint all ETSrobin on the screen and update the spec lable
refreshProcesses,NumOfProcesses	**cast**	**paint**	**paint**	update numberofProcesses in the guiState
sendnewStats,Env	**cast**	**paint**	**Waiting**	Read the sliders new info, and send the new specs to the Main Server.

Main Server:

The Main Server, on start initiating all the Computer Servers in the right boarders.
Generating a GUI state machine for printing to the screen.
He has an EtsRobins- where all the Robins and their locations are located, he updates it regularly.
In addition, he has an EtsMessages for marking in red line the messages in the GUI state machine.
He is in charge on monitoring the crashes of Computer Servers and telling other Computer Servers to take the falling node area.

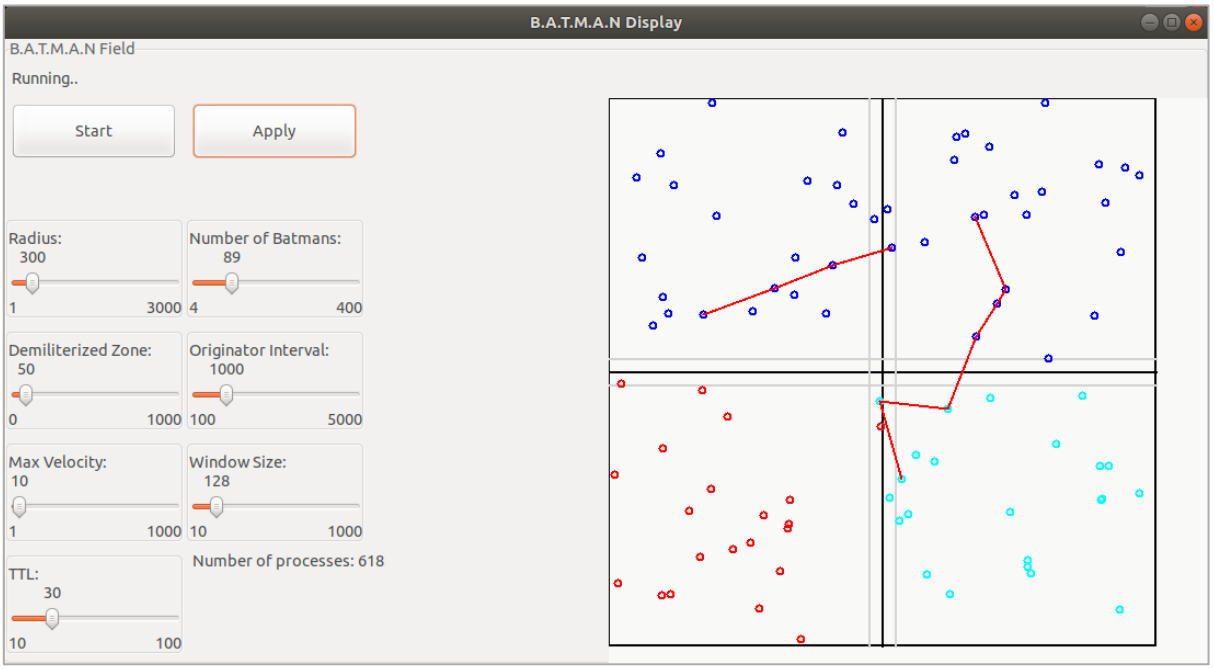
Main computer		
Message Name	**Type**	
getNumberOfProcesses	**Call**	retunrs the sum of all processes in my node + all other nodes
getNodes	**Call**	returns the list of nodes connected now
etsUpdate,FromNode,EtsX,EtsY	**Cast**	regular ETS update from Node EtsX and EtsY are lists of the original ETSs Update the main etsRobin that GUI uses
removeRobin,Pid,Node	**Cast**	Removes a Robin from the ETSRobins
addMessage,From,To	**Cast**	Adds a new message to etsMsgs for GUI to display
nodedown, MyNode	**Cast**	• Update boarders on all remaining computerServers. • Delete from EtsRobins and spawn new ones. • send the locations to transfer to the new computer o if 4 nodes remaining -> computer to the right or to the left takes Control and gets X,Y locations o if 3 nodes remaining -> and Startx = 0 **and** Endx = 0 then count the nodes o if 2 nodes remaining -> takes control on all the field and get the locations
newStats,NewSpecs	**Cast**	User inserted new Stats with the sliders on GUI: {Radius,Number of Robins,Demiliterized_Zone,ORIGINATOR_INTERVAL, MaxVelocity,WindowSize,TTL}. Restart ComputerServers with the new Stats.

Crashes Handling:

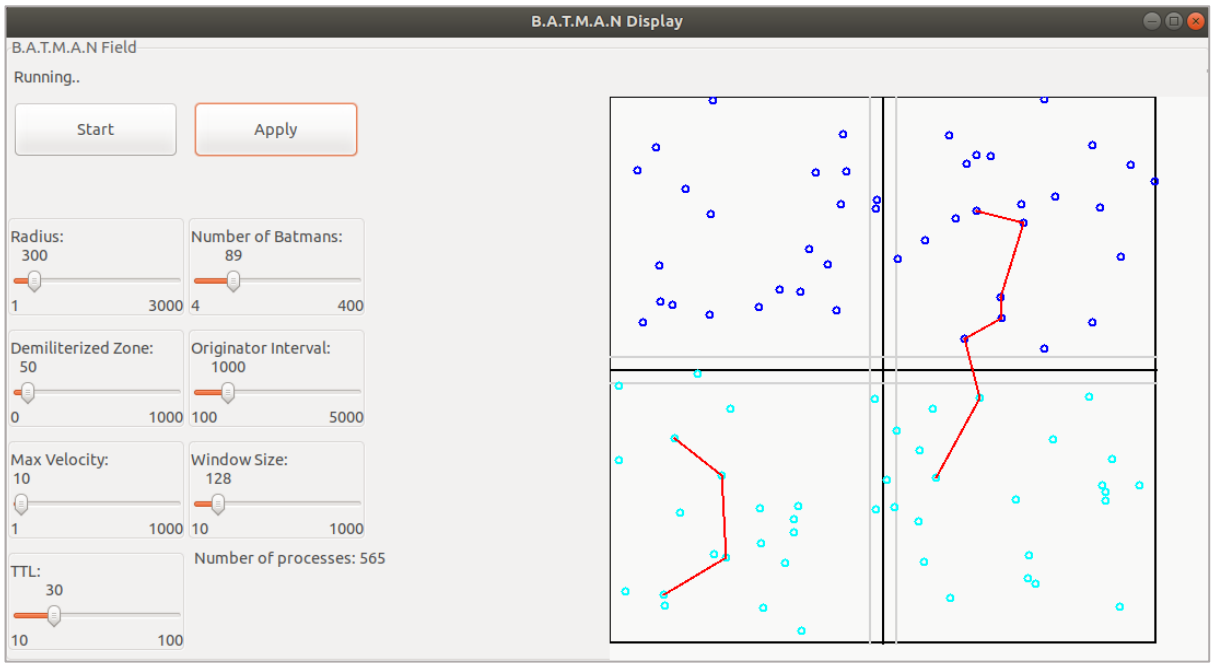
Crashing Node:

When a node crashes (one of the four computer servers), update boarders on all remaining computerServers and delete from EtsRobins all connected Robins, spawn new ones, and send the locations to transfer to the new computer -

- If there are 3 nodes remaining -> Computer to the right or to the left, of the one that fell, takes it and spawn new Robins in the same locations.



- If there are 2 nodes remaining -> Each node gets Half of the field.



- If 1 node remaining -> This node takes all and get the locations

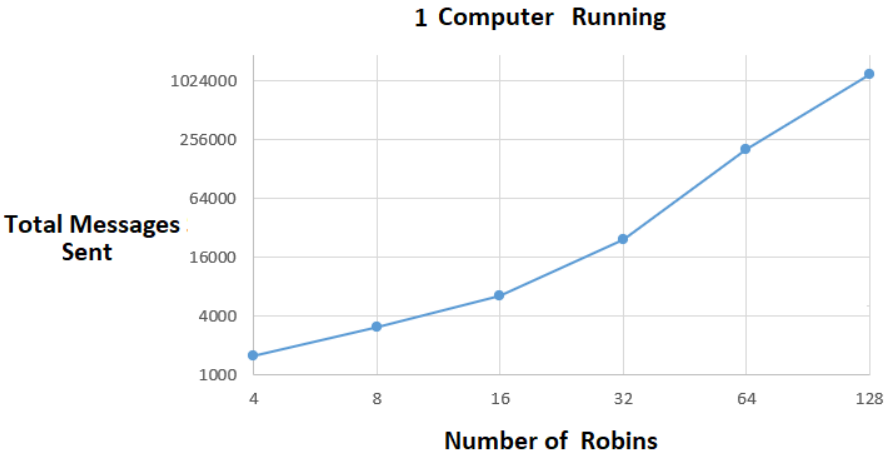
Robins Change Computer:

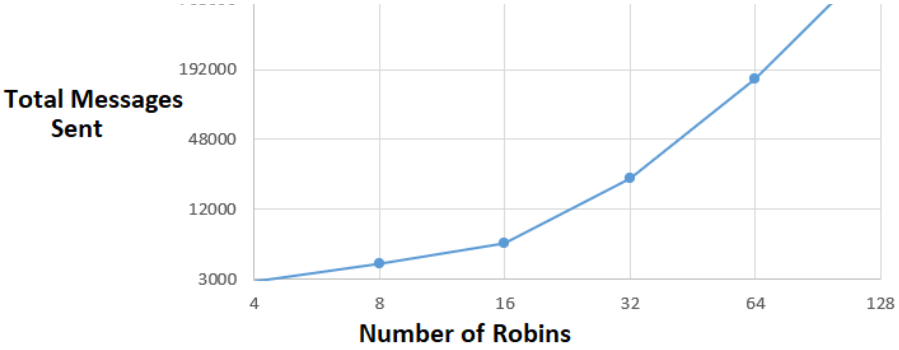
When a MoveSimulator detects an out-of-borders location, he makes a call request to his computer server, if another computer have fallen, the border might have changed, in this call the MoveSimulator will realize that and update its own borders in the state, if no computer have fallen or the borders of my computer server have not changed, MoveSimulator sends itself a stop message, and the computer server will cast a generate robin message to the computer with the right borders, including the important details about the fallen Robin.

Crashing Robin:

When a Robin crashes, a down message is sent to the Computer Server's monitor, and he will spawn a new Robin at the same location and monitor it.

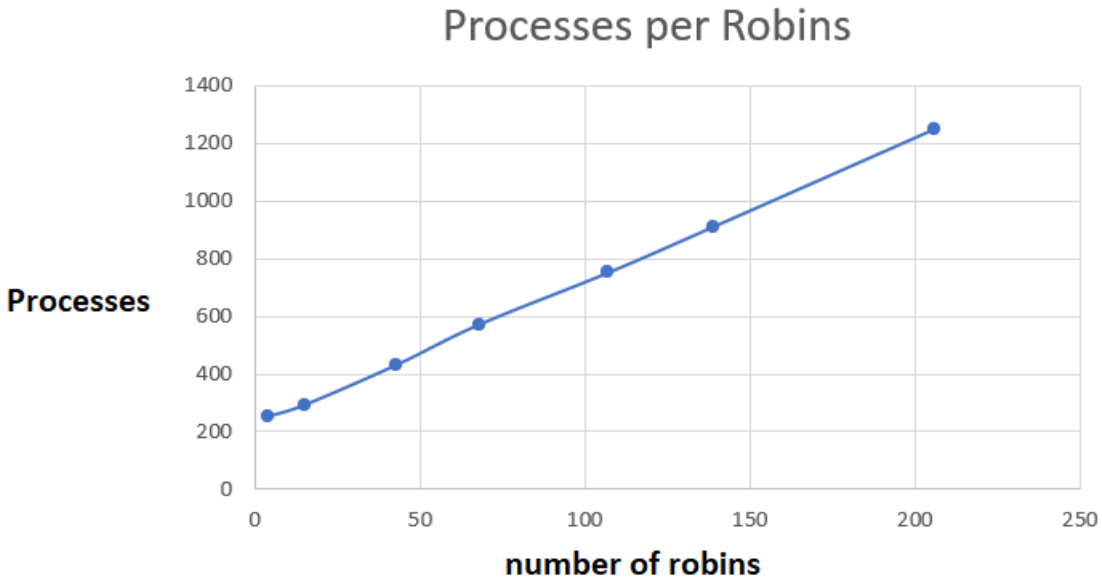
Statistics –





Here we can see the difference in the amount of total messages sent between all servers.

We can see that when 4 computers are running, a lot more messages are being sent, and the main reason will be communications and transitions between computers


Processes per Robins
Processes
1400
1200
1000
800
600
400
200
0
0
50
100
150
200
250
number of robins