DATA COLLECTION AND ANALYSIS ON VANET

HİLAL KARATOY
B.S., Information Technologies, Işık University, 2005

Submitted to the Graduate School of Science and Engineering
in partial fulfillment of the requirements for the degree of
Master of Science
in
Computer Engineering

IŞIK UNIVERSITY
2009

IŞIK UNIVERSITY

GRADUATE SCHOOL OF SCIENCE AND ENGINEERING

DATA COLLECTION AND ANALYSIS ON VANET

HİLAL KARATOY

APPROVED BY:

Assoc Prof. Ercan SOLAK (Thesis Supervisor)	(Işık University)	_____
Assist. Prof. O. Taner YILDIZ	(Işık University)	_____
Assist. Prof. Tankut ATAN	(Işık University)	_____

APPROVAL DATE: 02/01/2009

DATA COLLECTION AND ANALYSIS ON VANET

Abstract

The communication between devices is provided by wireless networks and it is growing rapidly, dependent on the needs. Wireless networks strike the attention of people throughout the world, and the ad hoc network is part of this attention. Ad hoc networks do not need structural mechanisms and are mostly used by the mobile nodes. In ad hoc networks the nodes act as a router and communicate with the other nodes. Ad hoc networks can be used in various places such as campus, shopping centers, buildings and vehicles.

Ad hoc network systems without any infrastructure have two kinds of nodes which are mobile and fixed. Actually, node mobility is not taken into account.
The aim of this thesis is to find the cheapest way to provide the practical statistical data that will be saved after communication between vehicles depend on road constructing, renewing, development and time schedule of the drivers etc. The data used in the analysis were the route and schedule information of the local city buses. Istanbul Metropolitan Municipality provided the coordinates of the bus stops and time tables for the data collection part. We used MATLAB to edit the data in data analysis part. We used 802.11 standards for the communication of the buses and C++ to create the protocol of data transmission. We wrote Google Earth scripts to observe the data correctness and coded TCL scripts for simulation in NS2 to see and analyze the data transfer between nodes, i.e., the buses. Different ranges of communication are used and analyzed, and their results are shown in this project. To collect more data, thousands of nodes are used, which represent more than 40 bus lines and their paths from different continents including Asia and Europe, which constitute the two sides of Istanbul.

VANET ÜZERİNDE VERİ TOPLAMA VE ANALİZ ETME

Özet

Günümüzde araçlar arasındaki iletişim kablosuz ağlar sayesinde sağlanmaktadır ve ihtiyaçlara bağlı olarak da kullanımı hızla artmaktadır. Kablosuz ağlar tüm dünyanın ilgi odağı haline gelmiş durumdadır ve özellikle tasarsız ağlar bunlardan biridir. Tasarsız ağlar çok adımlı iletişim sağlayan, altyapısız ve çoğunlukla hareketli düğümler tarafından kullanılan mekanizmalardır. Bu ağlarda düğümler hem yönlendirici görevi görür hemde diğer düğümler ile iletişim halinde olur. Tasarsız ağların kullanımı farklı yerlerde mümkündür bunlar üniversite kampüsleri, alışveriş merkezleri, ve hatta araçlar olarak örneklendirilebilinir.Tasarsız ağlarda düğümlerin hareketliliği önemsenmez. Düğümler hareketli ve hareketsiz olmak üzere ikiye ayrılır.

Bu tezin amacı, araçlar arasında gerçekleşen veri alışverişi sonucu elde edilen istatistiki bilgilerin, en ucuz şekilde yol yapılandırmalarında, şoförlerin iş saatlerine riayetinde, saatlere göre yol yoğunluğunun hesaplanmasında kullanılmasını sağlanmaktır. Kullanılacak araçlar belediye otobüsleri ve analiz edilecek veriler de otobüslerin yol bilgileri ve hareket saatleri olarak belirlendi. Veri toplama kısmında, otobüs duraklarının koordinatlarını ve ana durağa varış-ayrılış saatlerini Istanbul Büyükşehir Belediyesi verdi. Veri analizi için Matlab aracı kullanıldı. Araçlar arası bağlantı 802.11 standardıyla sağlandı ve veri alışverişi icin yeni bir protokol C++ dilinde yazıldı. Google Earth aracı koordinatların ve mesafelerin doğruluğunu görmek için kullanıldı. Otobüsler arasındaki veri alışverişini görmek ve simulasyonları gerçeklemek için ns-2 aracı kullanıldı. Farklı veri iletimi aralıkları kullanılarak sonuçlar grafik ortama taşındı ve gün içinde İstanbulun hangi noktaları hakkında ne kadar veri sahibi olunduğu analiz edildi. Verinin yayılımı için gerekli yayın alanı değerleri değiştirilerek, 40 otobüs hattından fazla, binlerce hareketli ve onlarca hareketsiz düğüm kullanılarak daha fazla veri elde edildi ve analizleri yapıldı. Istanbul'un avrupa ve asya yakalarına geçişlerindeki veri alışverişleride analizler sonuncunda görüldü.

Acknowledgements

I am deeply grateful to Assoc. Prof. Dr. Ercan Solak, my major professor and dissertation supervisor, who has done more than just supervising my thesis. Having the opportunity to work with him over the years has been intellectually rewarding and fulfilling.

I also thank Taner Eskil for his insightful suggestions and expertise.

My special thanks go to my closest friends Kristin Benli, Gülşen Aydoğan, Rina Barbut, Fethiye Erdem. Another special thanks for Cahit Çokal, Ahmet Soylu, Ömer Karataş, Turhan Daybelge, Ferhat Ural, Yalçın Şanlı, Ece Cansun, Burak Çizmeci, Burcu Adıgüzelli, Rüştü Derici, and Murat Anlı. They always share their knowledge with me and many times we had brainstorming. I'm also thankful to Neslihan Arslan and register office personnel whose friendship I deeply value. I never felt the absence of their support and presence.

The last words of thanks go to my family. I thank my parents Fethiye Karatoy and my little brother Bahadır Karatoy.

I also thank the Istanbul Metropolitan Municipality for providing us the raw data.

To Kerime ŞENGÜL

Table of Contents

List of Figures

List of Algorithms

List of Tables

List of Abbreviations

CBR	Constant Bit Rate
DSR	Dynamic Source Routing
GSM	Global System for Mobile Communication
IEEE	Institute of Electrical and Electronics Engineers
IETT	Istanbul Elektrikli Toplu Taşıma
KML	Keyhole Markup Language
MAC	Media Access Control
NAM	Network Animator
NS-2	Network Simulator version 2
OTcl	Object Tool Command Language
TCL	Tool Command Language
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VANET	Vehicle Ad Hoc Network

Chapter 1
Introduction

The need for wireless communication has been growing, which has also expanded its use. Primarily, wireless communication has been widely used by the military. Besides, as a result of the growing dependence on technological improvement, it has been used by different areas in different applications since 1980.

Users can reach the information independently through the wireless communication. Wireless networks has two main aspects; namely structured and ad hoc networks. Bus stations provide an example of the structured networks, in which communication is never cut when the communication device moves from one station range to go to another. On the other hand, in ad hoc networks mobile nodes can act as a router and communicate with the other nodes sending the packets through the others.

With the increasing use of wireless networks, different topologies and multi-hop systems started to be applied. However, its usage has created some problems. For this reason, MANET (Mobile ad hoc networks) was founded by IETF (Internet Engineering Task Force) to research the ad hoc networks and provide solutions for related problems. Since then, various routing protocols have been built up such as Destination Sequenced Distance Vector (DSDV), Wireless Routing Protocol (WRP), Temporally Ordered Routing Algorithm (TORA), Zone Routing Protocol (ZRP), Dynamic Source Routing (DSR), and Ad doc On Demand Distance Vector (AODV), which are among the most widely used protocols [1, 2, 3].

Beside all these routing protocols, *broadcast service* is important as much as the other protocols for all kinds of networks. When a message needs to be sent to all nodes across the network, or if the destination location is unknown at the time of

sending the message, a broadcast is necessary. In our project, we used broadcasting when the nodes send packets to the network. This data sending method is proper for our new protocol.

A VANET (Vehicular Ad Hoc Network) is a wireless network that is formed between vehicles as needed. VANEI nth last decade, research on VANET has improved rapidly. Thus, the attention to communication in car networking has grown with the improvements in wireless communication. In this networking system, vehicles must be equipped with the necessary devices, which include wireless transceivers and programmed software, to act as a mobile node as part of VANET.

The range of communication between vehicles may be limited to a few hundred meters, which could be a drawback for data exchanging. In such cases, to provide the end-to-end communication across a larger distance of communication, the roadside units can be used and operated as fixed nodes. This method extends the wireless communication because when the mobile nodes cannot move, the roadside units can be used in emergent situations or at other times, and this method can also be analyzed. These permanent units can be useful for various services of vehicular networks such as a drop point for the data about the traffic situation. For example, the roadside units serve as a gateway to the internet, or warn the driver as a road guide in a Mercedes. In this project, we used the fixed nodes peron to drop the collected data.

A VANET is similar to a MANET (Mobile Ad Hoc Network), but it has its own properties that differentiate it from a MANET. Vehicles are mobile just as mobile nodes in a MANET, yet they move at higher speed level than nodes in a MANET, and their communication can be interrupted by the surrounding buildings. In this project, we ignored the buildings and the weather conditions. In general, speeds of the vehicles moving in the same direction are similar, and their data communication rate is higher than the opposite direction. In our research, buses move at similar speed levels depending on the time of the day and the traffic situation on the roads because public transportation vehicles have to obey traffic rules. Their trip duration is scheduled, so their speed is calculated considering the road time.

VANET has a wide range of applications serving the interests of consumers, businesses, governments, law enforcement agencies, and emergency services. Beside this, the aim of our project is to collect city data and use it in obtaining profiles for the whole city [4].

The introduction part describes the motivation and the objectives of this research and then explains the outline of the thesis.

1.1 Research Motivation

The traffic density differs from urban to rural areas. In urban daytime, vehicles are densely packed, so packet sending is not as big a problem as it is in a rural highway or even in cities at night when fewer vehicles are moving and the response of the broadcasting may not be possible. In urban scenarios, sparse networks can be prevalent due to the sensitivity of the information exchange.

VANET provides communications between vehicles. In VANET, nodes interact without using fixed infrastructure or centralized system. It can be used by taxis, as well as public transportation, and the information can be used in many places to create statistical reports by gathering useful data. Examples include the arrival time to the bus stops, the density of the city traffic in specific hours, emerging situations, driving assistance, air pollution, and wasted time. The city bus system has been chosen for the project because buses use all the main roads of the city, are used by most of the citizens, have stable schedules, and are available for a wide time range. Istanbul is chosen as the experimental city since it is one of the three metropolitans that exist in Turkey. In Istanbul, there are approximately 2000 public buses, which start to give service at 05:30 am and work until 02:00am. If these vehicles communicate with each other while they are traveling in their path, the collected city information can be broadcast over the bus VANET.

"*One existing application is observed in the VMesh Demand-Response project where utility pricing information is exchanged from roaming utility vehicles for utility usage information from consumers' homes*." [5].

This is another reason for the choice of busses and the innovation of the new ideas. For example, this project can be used for the commercial part of the market. The buses can communicate with markets, shopping centers and other places, which are areas of commerce. Furthermore, when the traffic density is getting larger, commercial possibilities are also increasing.

Installing immobile infrastructure on roads incurs great expense, so vehicle-to-vehicle communication will be necessary to extend the range of network vehicles, and more effective for the commerce and collection of the data. Also, when a vehicle dumps its data, the robustness due to the storage will be obstructed [5].

1.2 Research Objective

The communication between vehicles, roadside units and fixed structures has been improving with new technologies. Recruitments are applied to the 802.11 by the IEEE 802.11 group in order to provide support for ITS (Intelligent Transportation System) application.

In existing infrastructure and ad hoc modes of the IEEE 802.11, wireless standards are being developed, which provides wireless devices with the ability to perform the short duration exchanges necessary to communicate between a high-velocity vehicle and fixed units. In this project, perons are the immobile units.

The high velocities at which vehicle move sometimes reduce the amount of time available for data exchanges. In our protocol, the delay time for the response of the message received is defined by a very little value, but it will be increased in the real time because of that reason. Apart from velocity, the direction of the vehicle in motion affects the data exchanges. Moreover, the road characteristics also constitute an influential factor. In this project, curve in a road and the waiting time on the traffic lamps and bus stations are ignored, which could be considered as the slight deficiency in the project.

Traffic density depends on the trip time of the buses moving in this project. Still, the density of the road in real time should be computed in the future [4].

Indeed, the objective of this thesis is to set up simulation environment for a bus VANET and test it with real data. To this purpose, a new protocol which analyses the information particularly produced for Istanbul has been created. To collect the information in the cheapest way, 802.11 standards are used, for it does not use the base stops and is less expensive compared to GSM [5].

1.3 Thesis Outline

Related research and previous experiments are presented in Chapter 2, in which the different localization methods in VANET are explained. Chapter 3 describes the tools which are used for data transformation, collection, and simulations. Chapter 4 contains the main development. The experimental work and the results are presented in Chapter 5. Finally, conclusion of the thesis is elaborated in Chapter 6.

Chapter 2
Literature Review

In this chapter, the background information that will be useful to understand the rest of this thesis will be given. In the following sections, the basics of ad hoc networks will be explained, and then the Geographical Position will be introduced.

2.1 Broadcast Protocol

When a message needs to be sent to all nodes across the network or if the destination location is unknown at the time of sending the message, a broadcast is necessary. A broadcast protocol is also used for routing. The basic way to implement a broadcast service, namely flooding, is to have each node broadcast the message which is sent to all of its neighbors except the node which is the message sender. For the wireless media, however, flooding is an inefficient method to send data because the bandwidth usage is increased. When the number of nodes increases on the network for one message, the usage of bandwidth also increases exponentially.

On the other hand, the traditional broadcast protocol for wireless ad hoc network has several advantages. First, the protocol is simple with its convenient implementation. Second, a node sending a message by broadcasting does not need to send an extra message when no broadcast message is generated. Third, each broadcast message will take all possible ways to reach other nodes in traditional broadcast protocols. It doesn't matter whether one of the nodes have failed; other nodes will still be able to send or receive the message.

Nevertheless, broadcast protocol has one crucial disadvantage depending on the conditions and needs. If the transmission radius is defined by a large value, or if the

number of the nodes is high, the transmission area of any node will cover many nodes. As it has already been explained, this is a disadvantage depending on the conditions. In our project, however, this particular disadvantage will not create an undesirable condition as data will not be collected from current node's data. Instead, different transmission ranges are applied, and the analyzed their effect on the data collection as well as communication density of the nodes will be analyzed.

2.2 Ad hoc Networks

Ad hoc networks do not use any infrastructure and preassigned routers to provide a communication between nodes. Instead, every node acts as a router. Nodes recognize other nodes which are in the same range. In Figure 2.1, B forwards the D or C's data to the node A because C and D can not reach node A directly.

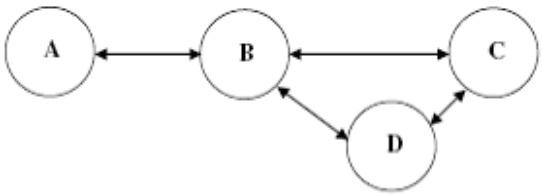


Figure 2.1 Example of an Ad Hoc Network

This kind of network doesn't need an expensive infrastructure, it can be setup as an independent network or it can be connected to the wired network using a hybrid structure with fixed nodes. This thesis is about VANET. Ad hoc networks made up of mobile vehicle nodes.

2.3 Geographical Positions

Latitudes and longitudes create a grid system on the Earth's surface and every point on Earth can be expressed with a unique set of latitude and longitude coordinates.

Latitude also known as parallels starts from earth's equator goes to south and north geographical poles respectively measure 90° north and 90° south from the equator which is 0° latitude. Each half has a 90 latitudes means that 180 latitudes divide the earth and one degree latitude equals to 111km.Latitude lines start at 0° on equator

and +90 on North Pole, -90 on South Pole. Longitude starts from Greenwich with 0° and span the world with 180° to the west and 180° to the east [7]. The distance between lines of longitude varies as we move from the equator to the north or south poles. Latitude coordinates are specified as **N**orth and **S**outh and longitude as **E**ast and **W**est.

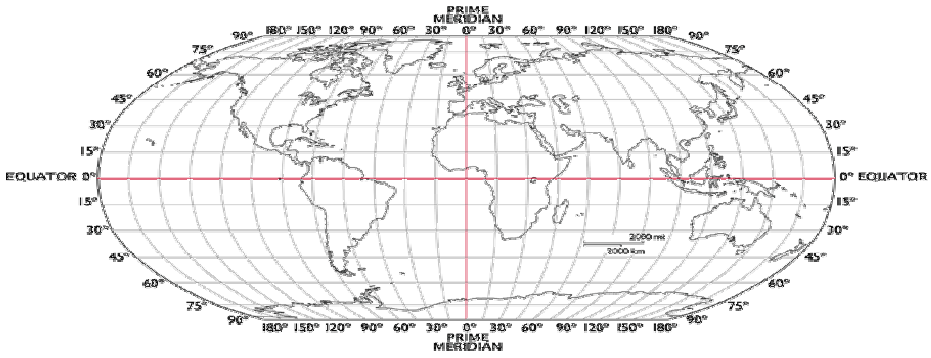


Figure 2.2 World Map

Latitude of Turkey is 36º north of the Equator and longitude of Turkey 26º East of Greenwich. *Turkey's* geographical *coordinates* are latitude *36-42*° N, longitude 26-45° E.

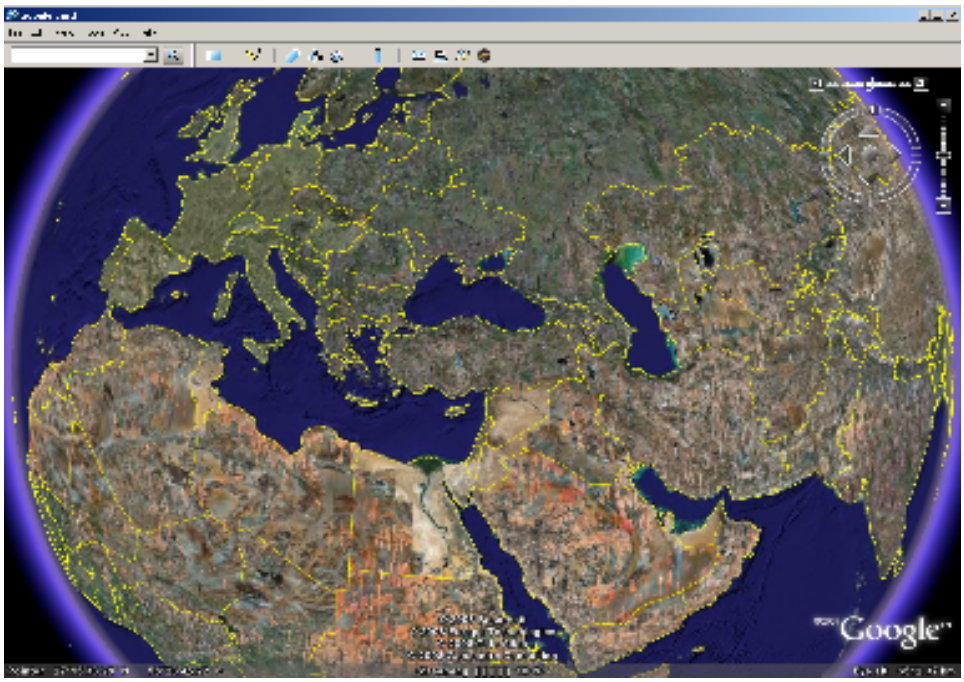


Figure 2.3 Turkey Map

Istanbul' geographical *coordinates* are latitude 40-41° N, longitude 28-29° E.

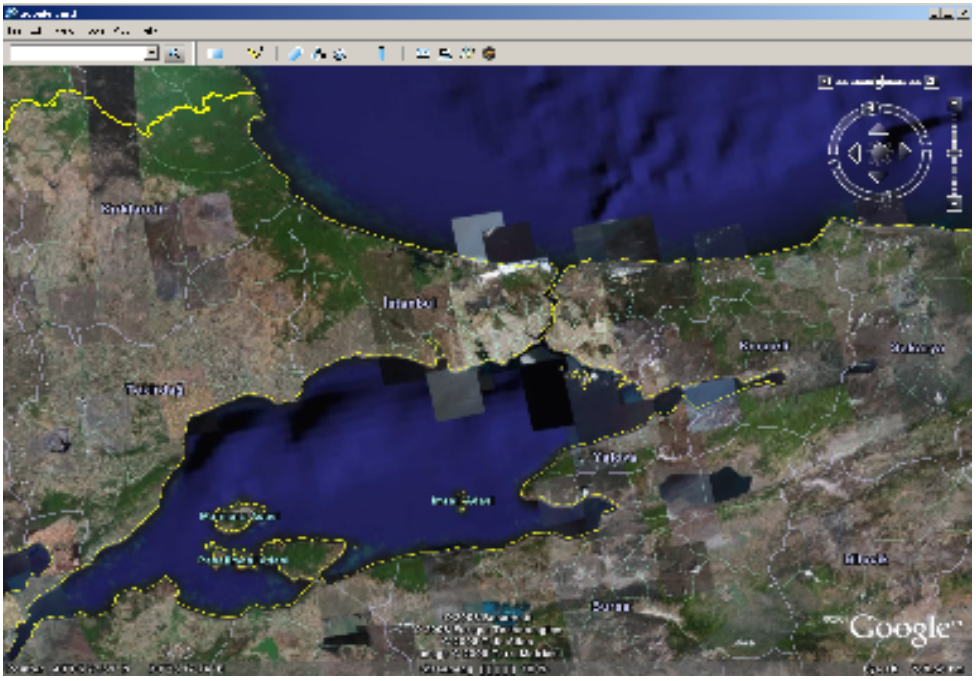


Figure 2.4 Istanbul Map

Chapter 3
Tools

In Chapter 3, Tools which are used in the project will be explained. The used tools are MATLAB, Google Earth, and ns-2. MATLAB is used for the functions which are used for analysis and simulation scripts. Google Earth is used for time and distance verifications and ns-2 used for verifications, movement and broadcasting simulations.

3.1 MATLAB

In MATLAB, users perform computational tasks faster than traditional programming languages [8].

MATLAB is the main tool in this project. It is used for raw data transformation, data collection, data analysis, visualizing data, and creates different types of files. Different types of MATLAB functionalities are used for the implementations.

In this project, the raw data type is string and they are converted to floating point data types and they are used in calculations. For all jobs the efficiently coded functions are used. The functions and operations will be shown below.

Structure data type is used to collect data. MATLAB String library is used to compare strings and convert strings to integers. Sorting functions are used for enumerating nodes across time.

3.2 Google Earth

Google Earth is a computer program which is developed by Keyhole and bought by Google. It basically shows satellite pictures and has different resolutions. When x

and y coordinates are given to the program the location is shown by Google Earth. Satellite imaginary, maps, terrain, 3D buildings are shown in map [9].

3.2.1 KML

Keyhole Markup Language (KML) is a file format used to display geographic data in an Earth browser such as Google Earth which is used for this project. KML uses a tag-based structure with nested elements, attributes and is based on the XML standard. In this project some KML tags are used to show the stops and paths, such as place marks, lines, polygons. KML is a case sensitive language.
Algorithms 3.1, 3.2 and 3.3 show KML scripts [10, 11].

Algorithm 3.1 Line drawn and Icon insertion on map

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.1">
                    <Document>
                              <name>1A.kml</name>
                              </Placemark>
                                        //Line properties
                              < Placemark >

                              < Placemark >
                                        //Icon coordinates
                              </Placemark >
                    </Document>
</kml>
```

Figure 3.1 Line and Icon on map

Algorithm 3.2 Line drawing on map

```
<Placemark>
        <name>1A_1</name>
                <styleUrl>#blueLine</styleUrl>
                        <LineString><altitudeMode>relative</altitudeMode>
                <coordinates>
                                                29.052830,41.023990,0.0
                                                29.042600,41.023500,0.0
                                                29.036300,41.024500,0.0
                                                29.033500,41.026800,0.0
                                                //x,y,z coordinates
                                        </coordinates>
                                </LineString>
</Placemark>
```

Figure 3.2 Line on map

Algorithm 3.3 Icon insertion on map

```
<Placemark>
        <name>1A_1_6</name>
                <Point>
                  <coordinates>29.052830,41.023990,0.0</coordinates>
                </Point>
</Placemark>
```

Figure 3.3 Icon on map

3.3 NS-2

Network Simulator (version 2) is an object-oriented network simulation tool written in C++ and simulation interface in OTcl (object tool command language). NS is useful for simulating many different network topologies [12].

3.3.1 TCL

Tool Command Language is dynamic programming language, suitable for networking usage. Tcl is an open source [13].

Tcl has a very wide range of uses but some of them are used in this project, which are to make wired and wireless connections, node movement, and broadcasting.

NAM (Network Animator) is a TCL/TK (Tool Command Language / TK GUI Toolkit) based animation tool for viewing network simulations. When a simulation has been run in ns-2 it outputs a trace file that can be opened in NAM and it's then possible to watch the nodes move and to see the simulation take place.

3.3.1.1 Simple Simulation Example (wired)

This section shows a simple NS simulation script and explains what each line does. Example is an OTcl script that creates the simple network configuration and runs the simulation scenario below. This example is for wired network and shown in Algorithm 3.4.

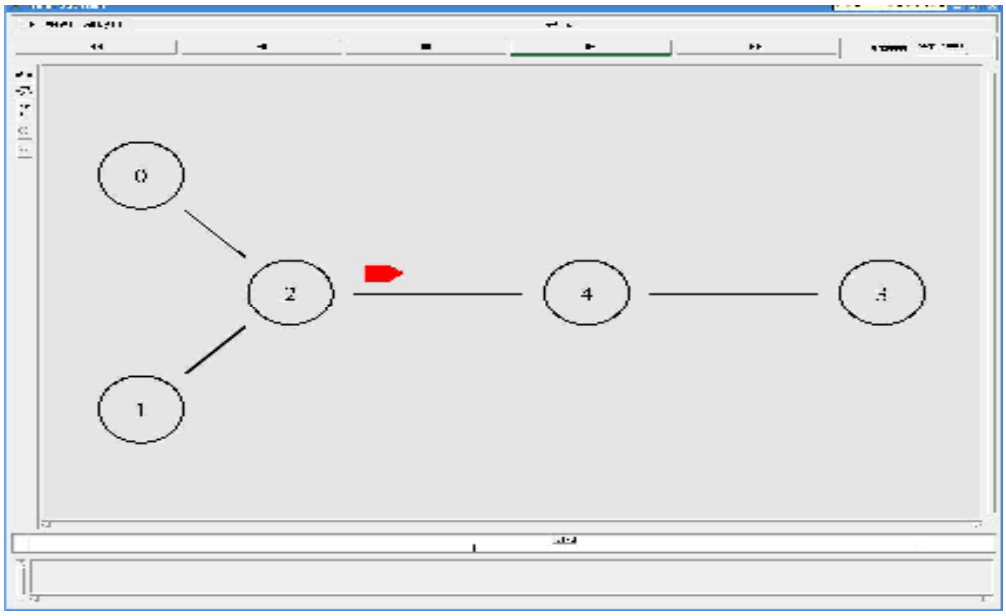


Figure 3.4 Nam for wired network

This network consists of 5 nodes (n0, n1, n2, n3, n4) as shown in Figure 3.4. The duplex links between n0 and n2, and n1 and n2 have 2 Mbps of bandwidth and 10 ms of delay. The duplex link between n2 and n4 and n4 and n3 has 1.7 Mbps of bandwidth and 20 ms of delay. An "UDP", user datagram protocol, agent is attached to n1, and a connected to a “null” agent attached to n3. A "null" agent just frees the packets received. As default, the maximum size of a packet that an "UDP" agent can generate is 1Kbyte and rate 100 bps. The "CBR", constant bit rate, is set to start at 0.1 sec and stop at 4.5 sec [17].

Algorithm 3.4 Example of wired network

```
#Create a simulator object
set ns [new Simulator]

#Open the NAM trace file
set nf [open out.nam w]
$ns namtrace-all $nf
```

Algorithm 3.5 (cont'd)

```
#Define a 'finish' procedure
proc finish {} {
        #Close the NAM trace file
        #Execute NAM on the trace file
}

#Create five nodes
set n0 [$ns node]
set n1 [$ns node]
...
#Create links between the nodes
$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
...
#Give node position (for NAM)
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
...
#Setup a UDP connection
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
set null [new Agent/Null]
$ns attach-agent $n3 $null
$ns connect $udp $null

#Setup a CBR over UDP connection
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set type_ CBR
$cbr set packet_size_ 1000
$cbr set interval_ 100
$cbr set rate_ 100

#Schedule events for the CBR
$ns at 0.1 "$cbr start"
$ns at 4.5 "$cbr stop"

#Call the finish procedure after 5 seconds of simulation time
$ns at 5.0 "finish"

#Run the simulation
$ns run
```

3.3.1.2 Simple Simulation Example (wireless)

This section shows a simple NS simulation script and explains what each line does. Example is an OTcl script that creates the simple network configuration and runs the simulation scenario below. This example is for wireless network and shown in Algorithm 3.5.

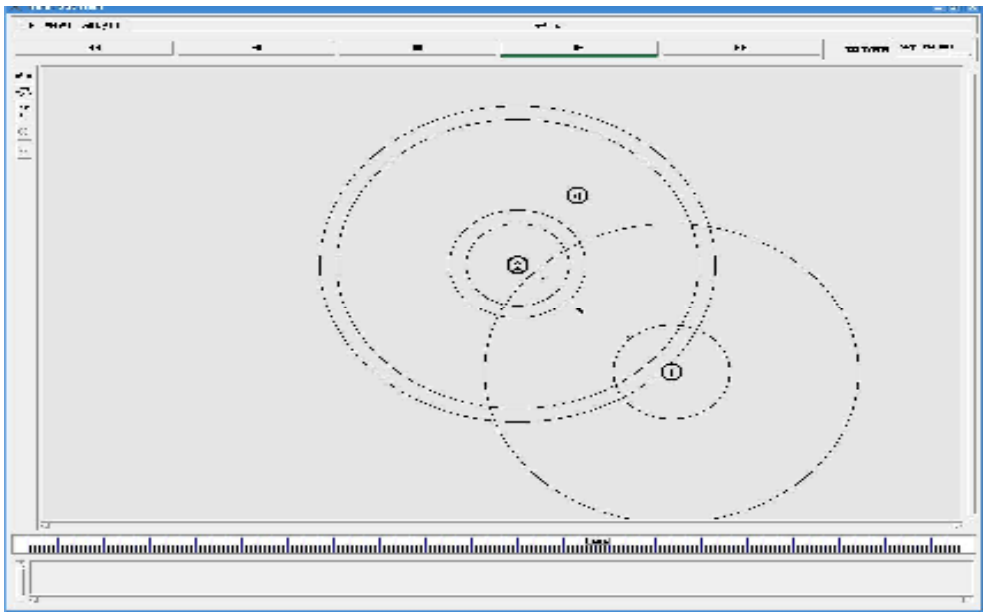


Figure 3.5 NAM for wireless network

This network consists of 3 nodes (n0, n1, n2) as shown in Figure 3.5. The wireless connection is between three nodes. An "UDP" agent is attached to n0, “CBR” attach to udp and a connected to a “null” agent attached to n1 and. A "TCP" agent is attached to n1, and a connected to a “sink” agent attached to n2. A "null" agent just frees the packets received. As default, the maximum size of a packet that an "UDP" agent can generate is 1 Kbytes and tcp packet size is 552byte. The "CBR" is set to start at 61 sec and “TCP” start at 93sec and continue to connection till simulation end.

Algorithm 3.5 Example of wireless network

```
#Define options
set val(chan)          Channel/WirelessChannel    ;# channel type
set val(netif)         Phy/WirelessPhy            ;# netw interface type
set val(rp)            AODV                       ;# routing protocol
set val(x)             500                        ;# X dimension (meter)
set val(y)             500                        ;# Y dimension (meter)
set val(time)          200.0                      ;# Simulation time (sec)

# Initialize Global Variables
set ns_        [new Simulator]
set tracefile  [open out.tr w]
$ns_ use-newtrace
set namfile [open out.nam w]

# Create God
create-god $val(nn)

#  Create the specified number of mobilenodes [$val(nn)] and "attach" them
```

Algorithm 3.5 (cont'd)

```
# to the channel.
set chan [new $val(chan)]

# configure node
$ns_ node-config -adhocRouting $val(rp) \
            -agentTrace ON \
            -routerTrace ON \
            -macTrace ON \
            -channel $chan

for {set i 0} {$i < $val(nn) } {incr i} {
      set n($i) [$ns_ node]
      $n($i) random-motion 0          ;# disable random motion
      $ns_ initial_n pos $n($i) 20
}

# Provide initial (X,Y, for now Z=0) co-ordinates for mobilenodes
# Node movement
$ns_ at 3.0 "$n(0) setdest 235.3 285.8 5.0"
# Setup traffic flow between nodes
set udp_(0) [new Agent/UDP]
$ns_ attach-agent $n(0) $udp_(0)
$ns_ at 61 "$cbr_(0) start"
# Create an FTP application from 1 to 2 at time 93 (seconds)
set tcp [new Agent/TCP]
$tcp set class_ 2
$ns_ attach-agent $n(1) $tcp
set sink [new Agent/TCPSink]
$ns_ attach-agent $n(2) $sink
$ns_ connect $tcp $sink
$tcp set fid_ 1
$tcp set packetSize_ 552
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns_ at 93 "$ftp start"
```

MobileNode parameters:

A wireless simulation is made up of some number of **MobileNode**s. Each such mobile node needs some options to be configured like routing protocol, MAC layer protocol, antenna type, channel type etc. A list of all parameters is given below for reference.

- Channel type (Channel/WirelessChannel)
- Propagation model (Propagation/TwoRayGround)
- Interface type (Phy/WirelessPhy)
- MAC layer protocol (Mac/802_11)
- Routing protocol (DSR)
- Interface Queue type (CMUPriQueue - for DSR)
- Interface Queue Length (50)
- Antenna type (Antenna/OmniAntenna)

- LL type (LL)

Energy parameters [14] **:**

- energyModel $val(engmodel)
- rxPower $val(rxPower) # transmitting power
- txPower $val(txPower) # recving power
- sensePower $val(sensePower) # sensing power
- idlePower $val(idlePower) # idle power
- initialEnergy $val(initeng) # Initial energy

Topology parameters:

These are the configuration parameters for the topology structure, like the dimensions of the grid, number of nodes present etc. A list of them is given below for reference.

- x-dimension of the topography
- y-dimension of the topography
- Number of nodes.

Other parameters are,

- Total simulation time, and
- Trace file name

3.3.1.3 Trace File

This section shows a trace analysis example. In Algorithm 3.6, trace file opening and writing traces on it are shown.

Algorithm 3.6 Cerate trace file

```
set tracefile   [open out.tr w]
$ns_ use-newtrace
$ns_ trace-all $tracefile
```

Running the above script generates a NAM trace file that is going to be used as an input to NAM and a trace file called "out.tr" that will be used for simulation analysis. Figure 3.6 shows the trace format and example trace data from "out.tr".

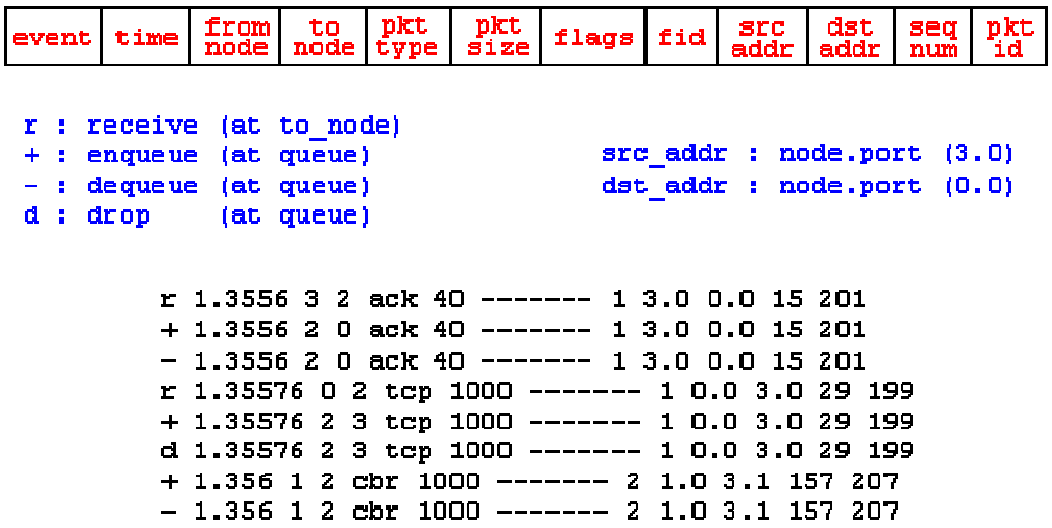


Figure 3.6 Trace Format example

Each trace line start with event descriptor which are (+,-, r, d), and followed by the simulation time (in second) to the event, and show the node numbers which are from and to nodes, which identify the link on which the event occurred. Next information is about the packet type and its size (in bytes). The next field after "---" is flow id of ipV6 that a user can set for each flow at the input OTcl scripts, flow id field is also used when specifying stream color for the NAM display. The next two fields are source and the destination address in forms of "node.port". The next field shows the network layer protocol's packet sequence number (UDP doesn't use SEQ number). The last field shows the unique id of the packet [15].

3.3.1.4 Add New Protocol

To add new protocol in ns-2 there are two points, we should obey. The first one is a packet and the second one is an agent. (A "packet" is a group of data. An "agent" is component of software and hardware and it can communicate the other agents with using those properties). The packet header must be declared, what packet header will include, such as source ip address, packet number, and destination ip address. After deciding of the packet header structure it must be introduced to the system. The final step is for the agent. What will agent do when receive the packet or send the packet. Some necessary changes will be done in ns-2 files.

The new packet name must be written in “packet.h” to create static class for the OTcl linkage and “tcl/lib/ns-packet.tcl” to enable new header in OTcl.

The new agent link with a OTcl class.

This example is going on the ping packet type and agent. The necessary changes are below.

- Copy “ping.cc” and “ping.h” as “new_.h”, “new_.cc” to the ns-2 directory. new_.cc includes the recv and send methods for the new agent.
- Register the new application header by modifying “packet.h” and “ns-packet.tcl”.
- Set default values for the newly introduced configurable parameters in "ns-default.tcl"
- Register the new application by modifying Makefile

  ping.o \ new_.o

  $(LIB_DIR)int.Vec.o $(LIB_DIR)int.RVec.o \
- Run “make” on the console and introduce the changes to the ns-2.
- To test the new protocol, write a simple tcl file which is shown in Algorithm 3.7.
- To control this tcl file write “ns new_.tcl” on the console [16].

Algorithm 3.7 Simple TCL file

```
set ns [new Simulator]
set nf [open out.nam w]
$ns namtrace-all $nf
#Create three nodes
#Connect the nodes with one link
#Create two new_ agents and attach them to those nodes
#Connect the three agents
#Schedule events
#Run the simulation
```

Chapter 4
New Protocol IETT on NS-2

In this chapter some tools that were referred in Chapter 3, process flow and project's algorithm will be explained in details. They will be detailed under the sections of Raw Data Transformation and Data Simulation. We started with four text files. Those four files had much information about the buses and bus stops. We separated those data in files and tested their accuracy by using test simulations. Then the protocol is used in data exchanging among buses.

4.1 Raw Data Transformation

The raw data is contained in four text files. Those four files had the information of buses and bus stops of Istanbul.

- **Hatlar(bus_route) :** includes the bus route' information such as bus identity number,operational zone, trip_time, and path.
- **Duraklar(bus_stops) :** includes the bus stops' information such as stops' identity number, name, coordinates, direction, construction history and other information such as ticket counters.
- **Hattin_Duraklari(stops_of_the_line) :** includes the stops of the bus' route such as bus' identity number, direction type (which is departure or arrival), bus stops' id.
- **Hareket_Saatleri_Isgunu(arrival_departure_time)** : includes the buses' movement information, how many buses will be used for different locations, and daily schedule for them and each bus movement are organized by given service number and arrival departure time.

Using MATLAB we combined using related items and coded by MATLAB. The Turkish frames of files come from their originals supplied by IETT.

4.1.1 Data Selection

In this project, the first step is to select the data from the four text files. Of course the related data must be selected. Data are stored in the structures because some data types must be different, such as integer, string and float. First of all, each text file re-separated column by column and related columns were selected which would be used in the next step of the project. Each text file will be shown below after the explanation of each item.

The four text files are shown below and their given data and selected data will be shown with examples. These files include data which was more than what we needed. For this reason, specific columns were selected.

Table 4.1 Hatlar.txt

hat_kodu	isletme_bol	sefer_muddet	ana_guzergah	**hat_adi**
10	Anadolu	130	KADIKÖY-ALTIYOL-Z.BEY-S.CEDİD-AYŞE KADIN-KÜÇÜKYALI-İDEALTEPE-MALTEPE-TUGAY YOLU CEVİZLİ	KADIKÖY-MALTEPE-CEVİZLİ
10B	Anadolu	110	KADIKÖY-ALTIYOL-ZİVERBEY-KOZYATAĞI-BÖCEKLİ CAMİİ-BAYAR CAD-ALTBOSTANCI	KADIKÖY-ALTBOSTANCI
141	İkitelli	40	CİHANGİR MAH.- AMBARLI - AVCILAR MERKEZİ - İST.ÜNV.AVCILAR KAMPÜSÜ	CİHANGİR MAH. - AVCILAR METROBÜS
142F	İkitelli	80	YEŞİLKENT-TOKAT MH.-FİRÜZKÖY-ÜNİVERSİTE MH-FABRİKALAR YOLU- AVCILAR METROBÜS	YEŞİLKENT- AVCILAR METROBÜS
150	Beyoğlu	90	SARIYER-MADEN MAHALLESİ- KOÇ ÜNİVERSİTESİ-GARİPÇE KÖYÜ-RUMELİ FENERİ	SARIYER - RUMELİ FENERİ
151	Beyoğlu	90	SARIYER-MADEN MAH.-UYUM SİTESİ-ZEKERİYA KÖY-USKUMRUKÖY-KİLYOS	SARIYER - KİLYOS
55	Topkapı	95	GAZİOSMANPAŞA-ÜÇ ŞEHİTLER - EYÜP - HALICIOĞLU - OKMEYDANI - ŞİŞLİ	GAZİOSMANPAŞA - ŞİŞLİ
55ET	Topkapı	70	EYÜPÜÇŞEHİTLER-AYVANSARAY-HALICIOĞLU-ANADOLU KAHVESİ-DOLAPDERE-TAKSİM-ALTINTEPE	EYÜP ÜÇŞEHİTLER-TAKSİM
55M	Topkapı	70	EYÜP ÜÇŞEHİTLER-AYVANSARAY- HALICIOĞLU- ÇEVREYOLU- ÇAĞLĞYAN- MECİDİYEKÖY	EYÜP ÜÇŞEHİTLER-MECİDİYEKÖY

Table 4.2 Duraklar.txt

durak_kodu	ad	yon	isletme_bol	Durak_tipi	modul_adedi	yeni_durak_kodu	yapilis_tarihi	ilcesi	x_koordinat	**y_koordinat**
A0001B	ÜSKÜDAR CAMİ ÖNÜ	1 peron (HAT 15B-15C	Üsküdar	MODERN	10	140010	02.02.2001 00:00	Üsküdar	2.901.630.000	4.102.790.000
A0001C	ÜSKÜDAR CAMİ ÖNÜ	2 peron (HAT15-15A)	Üsküdar	MODERN	8	140020	13.02.2001 00:00	Üsküdar	2.901.600.000	4.102.760.000
A0001D	ÜSKÜDAR CAMİ ÖNÜ	ÇEŞME YANI(15Ş)	Üsküdar	WALL	3	140030	16.07.1998 00:00	Üsküdar	2.901.560.000	4.102.660.000
A0002A	TEKEL DEPOSU	BEYKOZ YÖNÜ	Üsküdar	AÇIK ÜÇGEN	0	140042	01.06.1998 00:00	Üsküdar	2.902.060.000	4.103.130.000
A0002B	TEKEL DEPOSU	ÜSKÜDAR	Üsküdar	MODERN	1	140041	30.11.2003 00:00	Üsküdar	2.902.060.000	4.103.130.000
A0003A	PAŞALİMANI	BEYKOZ	Üsküdar	MODERN	1	140052	27.10.2001 00:00	Üsküdar	2.902.550.000	4.103.390.000
A0003B	PAŞALİMANI	ÜSKÜDAR	Üsküdar	WALL	1	140051	11.09.1998 00:00	Üsküdar	2.902.530.000	4.103.380.000
A0004A	KUZGUNCUK	BEYKOZ YÖNÜ	Üsküdar	AÇIK	0	140062		Üsküdar	2.902.910.000	4.103.640.000
A0004B	KUZGUNCUK	ÜSKÜDAR YÖNÜ	Üsküdar	WALL	1	140061	08.05.1998 00:00	Üsküdar	2.902.950.000	4.103.650.000

Table 4.3 Hattin_Duraklari.txt

hat_kodu	Yon	sira_no	durak_kodu
10	D	1	A2224A
10	D	2	A2003A
10	D	3	A2408A
10	G	2	A0587A
10	G	3	A0589A
10	G	4	A0591B
1A	G	1	A0330A
1A	G	2	A0287A
1A	G	3	A0285B

Table 4.4 Hareket_Saatleri_Isgunu.txt

hat_kodu	servis_no	sira_no	gun_tipi	saat_gidis	saat_donus
10	381	1	I	01.01.2000 06:25	01.01.2000 08:00
10	381	2	I	01.01.2000 09:10	01.01.2000 16:25
10	381	3	I	01.01.2000 17:30	01.01.2000 18:40
10	383	1	I	01.01.2000 07:05	01.01.2000 08:40
10	383	2	I	01.01.2000 09:50	01.01.2000 17:05
10	383	3	I	01.01.2000 18:10	01.01.2000 19:15
10E	441	3	I	01.01.2000 09:00	01.01.2000 09:50
10E	441	4	I	01.01.2000 10:40	01.01.2000 11:35
10E	441	5	I	01.01.2000 12:25	
10E	442	1	I		01.01.2000 13:40

- Table 4.1 shows Hatlar.txt, **bus_route text file**.

  Selected:

  bus_id =10E

  trip_time= 90

  line_name = KADIKÖY-ESATPAŞA

- Table 4.2 shows Duraklar.txt, **bus_stops text file.**

  Selected:

  stop_id = A0001B

  stop_name= TEKEL DEPOSU

  x_coordinate=29.01630000

  y_coordinate=41.02790000

- Table 4.3 shows Hattin_Duraklari.txt, **stops_of_the_line text file.**

  Selected:

  bus_id=10E

  direction_type=D

  stop_id= A0627B

- Table 4.4 shows Hareket_Saatleri_Isgunu.txt, **arrival_departure_time text file.**

  Selected:

  bus_id=10E

  service_number=441

  day_type=I

  departure_time=01.01.2000 10:40:00

  arrival_time=01.01.2000 11:35:00

Explanation of the Selected Data

- **bus_id (**hat_kodu**) =** It is given to the whole line. Each line can have several buses operating on the line. The number of buses for each line (bus_id) approximately varies with the number of people using the line.

  Example: 10E

- **trip_time (**sefer_muddeti**)** = It is the round trip time.
- **line_name (**hat_adi**)** = It shows the source and destination neighborhood name which are departure and arrival stops' locations.

  Example: KADIKÖY-ESATPAŞA
- **stop_id (**durak_kodu**)** = It is given to bus stops. Their id must be unique and each bus stops in certain stops and they are determined by the municipality.

  Example: **A0001B** (TEKEL DEPOSU)
- **stop_name (**durak_adi**)** = Each stop has a name but names are not unique.

  Example: **TEKEL DEPOSU** (A0001B)
- **x_coordinate (**x_koordinat**)** = Stop's x coordinate (longitude) on the earth.

  Example: 29.01630000
- **y_coordinate (**x_koordinat**)** = Stop's y coordinate (latitude) on the earth.

  Example: 41.02790000
- **direction_type (**yon**)** = It is the direction of the bus's route. G symbolizes departure from the main stop and D arrival to the main stop.
- **service_number (**servis_no**)** = Each bus has a service number.

  Example: 441
- **day_type (**gun_tipi**)** = It is type of day, week days are symbolized by I and weekend days are symbolized by C and P. C for Saturday and P for Sunday.

  Example: I
- **departure_time (**saat_gidis**)** = Departure time of bus from the peron. Time is represented as hh:mm:ss.

  Example: 10:40:00
- **arrival_time (saat_donus)** = Departure time of bus from the peron. Time is represented as hh:mm:ss.

  Example: 11:35:00

When we analyzed data, we considered two main points.

1. There are two types of paths. The first one is RING, which means the bus returns to the starting stop. Bus's starting and ending stops are the same. In the second type, the bus departs from one stop but arrives at a different stop.
2. There are two ways of assigning service numbers to the bus line. One of them is the situation of having two consecutive numbers in which first of them is an

odd number. The other situation is having two numbers in which first of them is an odd number and the other number is equal to the number plus 2.

Example: 331-332 are assigned to one bus
331-333 are given to different buses

4.1.2 Data Structures

We store selected items in the structures which are strHT, strDR, strHD, strHS.

- strHT is created from Hatlar.txt and we used it to define which bus_id spends time on the line. Structure of strHT is;

```
structure strHT     {
                        string hat_kodu;
                        integer sefer_muddeti; //minute
                    }
```

- strDR is created from Duraklar.txt and we used it to store stops' information. Structure of strDR is;

```
structure strDR     {
                        string durak_kodu;
                        string durak_ad;
                        float x_koor;
                        float y_koor;
                    }
```

Stop's name and coordinates can be same but their ids must be unique. If the stop coordinates are same but ids are different then we can say that this stop is used for arrival and departure directions.

- strHD structure is created from Hattin_Duraklari.txt and it is used to store bus's line information. Structure of strHD is;

```
structure strHD     {
                        string hat_kodu;
                        string yon;
                        string durak_kodu;
                        integer sira_no;
                    }
```

This structure shows whether the bus is RING or not. If the bus has directions for return and forward that means that the bus is not RING, otherwise if the bus has one direction which is return it means that the bus is RING.

- strHS structure is created from Hareket_Saatleri_Isgunu.txt and it is used to store arrival and departure time in each service. Each route can have more than one bus.

```
structure strHS        {
                         integer servis_no;
                        integer sira_no;
                        string gun_tipi;
                       string durak_kodu;
                       integer saat_gidis; //second
                       integer saat_donus; //second
                     }
```

4.1.2.1 Distance Computation

The given coordinates depend on the longitude and latitude. For Turkey the longitude's distance is 85 km and latitude's distance is 111km. Each coordinate must be converted to the same unit of data; here 1 unit corresponds to 1 kilometer. The Euclidian formula can be used now as

$$\sqrt{((y_2 - y_1)111^2 + (x_2 - x_1)85^2)}$$

We calculated the distance between two consecutive stops by above equation. In this structure there are "n" stops and "n-1" intervals.

We added the intervals and found the distance between starting stop and ending stop of the each bus_id. Algorithm 4.1 is used to find the intervals and total distance of the each bus_id.

Algorithm 4.1 Total distance

```
total_distance=0;
total_distance_a=0;
total_distance_d=0;
for i=2:size(strHD_Durak_Don)
        if distance_a(i) = = 0 // bus has return trip to go to the main stop
                distance_a(i)= ((y(i)-y(i-1))*111^2+(x(i)-x(i-1))85^2);
```

Algorithm 4.1 (cont'd)

```
                    total_distance_a=total_distance_a + distance_a(i);
          end
end
 if (bus_id== Not RING)
          for i=2:size(strHD_Durak_Git)
               distance_d(i)= ((y(i)-y(i-1))*111^2+(x(i)-x(i-1))85^2) ;
               total_distance_d=total_distance_d + distance_d(i);
          end
end
total_distance=total_distance_a + total_distance_d;
```

4.1.2.2 Time Computation

There are two choices to calculate the round trip time. The first one is trip_time (sefer_müddeti), and the second one is subtractions of sequential trip times. If trip_time is less than the subtractions of sequential trip times, period time will be used for the round trip time. When we have a RING, there is no D column for it in struct strHD.

The first step is to check if the bus is RING or not. If the bus_id is RING just sequential departure times will be subtracted. To explain it clearly some examples are shown below.

RING ;

Table 4.5 Example of RING time schedule

	Departure	**Arrival**
i-1	time(i-1)	
i	time(i)	

If those two sequential times, shown in Table 4.5, are for the same bus_id and they are in the same service_number, the first time will be subtracted from the second one and it is compared with the period time of the bus_id which is less than the other, as shown in Algorithm 4.2, it will be used for the stop round trip time.

Example:

We assumed;

TD = Total distance between starting and ending stops (it is found in "Distance Computation")

d (i) = Distance between two stops(centimeter) (it is found in "Distance Computation")

trip_time = 30 minutes (it is given by IETT)

time (i-1) = 21840 seconds (it is given as hh:mm:ss and we transformed it to second)

time (i) = 23520 seconds(it is given as hh:mm:ss and we transformed it to second)

Algorithm 4.2 Time computation for RING

```
time=time(i)-time(i-1);(second)
time=time;(minutes, which is transformed from second)
        if(time<trip_time)
                interval_time(i)=(d(i)*time) / TD;
        else
                interval_time(i)=(d(i)*trip_time) / TD;
        end
```

We can calculate the passing time from each stop by adding time intervals to the departure time from the each stop till end of the path.

The trip time calculation depends on the chosen time, total distance and the interval distance between two sequential stops.

Not RING ;

Table 4.6 Example of Not RING time schedule

	Departure	**Arrival**
i-1	time1(i-1)	time2(i-1)
i	time(i)	0

This is for a bus_id which is not RING and the calculation will be more complex than the RING. Second column is for the departure time from the peron, and third column is for the arrival time from the last peron. Algorithm 4.3 shows the calculation of each interval time.

Example:

We assumed;

TD = Total distance between starting and ending stops (it is found in "Distance Computation")

TD_a = Total distance for return trip(centimeter) (it is found in "Distance Computation")

TD_d (i) = Total distance for forward trip(centimeter) (it is found in "Distance Computation")

trip_time = for round trip time (minutes) (it is given by IETT)

time1 (i-1) = It is given as hh:mm:ss and we transformed it to second

time2 (i-1) = It is given as hh:mm:ss and we transformed it to second

time1(i) = It is given as hh:mm:ss and we transformed it to second

Algorithm 4.3 Time computation for Not RING

```
time_d=time2(i-1) – time1(i-1)
time_a=time1(i) – time2(i-1)
time=time_d +time_a;
        if(time<trip_time)
                interval_time(i)=(d(i)*time) / TD;
        else
                interval_time(i)=(d(i)*trip_time) / TD;
        end
```

In Not RING,

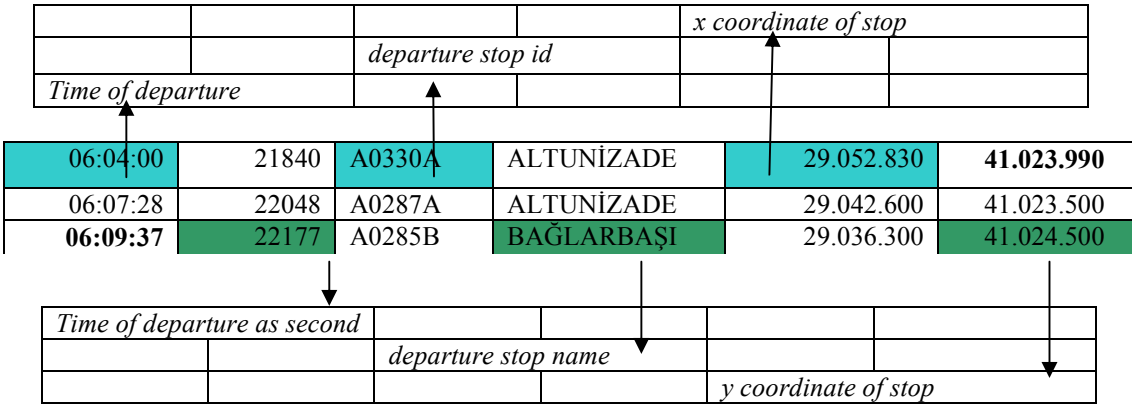
After that calculation same as RING interval time added to the starting time of the departure or arrival sequentially.

4.1.3 Bus Items

For each bus trips we created text files containing time, stop's identification and distance information. The text file includes the information of the each trip of the bus_id. Each text file name is specified for the each bus_id and they are sorted by their starting time of the trip. For example 1A_1.txt means this is the first trip of the bus whose id is 1A, it stores the forward trip information.

An example text file is shown in Table 4.7.

Table 4.7 1A_1.txt



4.2 Data Simulation

The aim of this part is to verify the data extracted in the previous part of this chapter. We used simulations for accuracy of the given data. We generated scripts in MATLAB, and used them in Google Earth and ns-2. Those tools were explained in Chapter 3.

4.2.1 Coordinate Verification

KML scripts were generated in MATLAB and they are displayed in Google Earth. We inspected the coverage area of the bus network. We wrote KML scripts to analyze them sense of proportion. The coverage concentrated mainly at center of Istanbul but does not reach to the far corners. Figure 4.1 shows all the bus stops of the IETT.

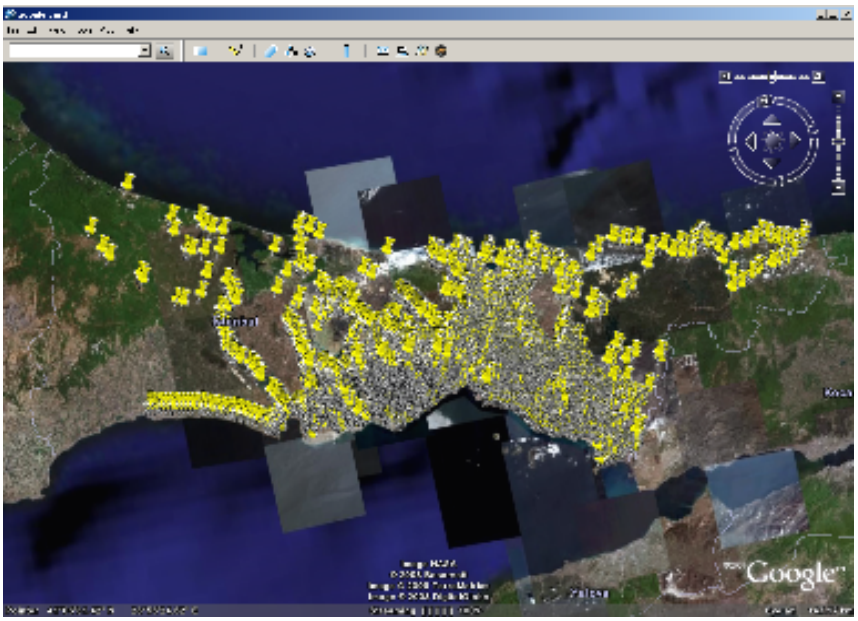


Figure 4.1 All stops are located on the Istanbul

We inspected the bus stops for many lines. We wanted to verify if actually there is a bus stop at the given locations. In some cases there are small discrepancies. For example a bus stop fall into residential areas close to the road.

We also observed intersection among lines.

Example:

10E → 10B, 1A, 14B, 522B this means 10E uses some common points with these buses. To verify, we wrote KML scripts and they are displayed in Google Earth. The observation shows that, approximately hundreds buses pass the same place and they use the same stop or not but they pass the same zone, such as KUYUBAŞI. However one point is realized that if we want to find intersections we have to check stop_ids, not zones. In Figure 4.2, is created in MATLAB and it is plotted, two lines are shown, 12H and 10E. They pass through the same zone but they do not meet.

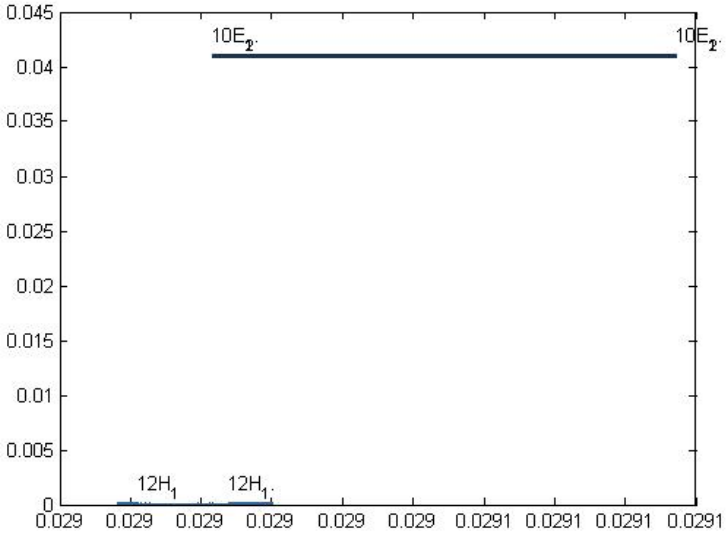


Figure 4.2 Intersection Busses

4.2.2 Measurement Verification

We generated KML scripts for bus lines in MATLAB and displayed in Google Earth. We used diameter tool of Google Earth to verify the distance between two stops.

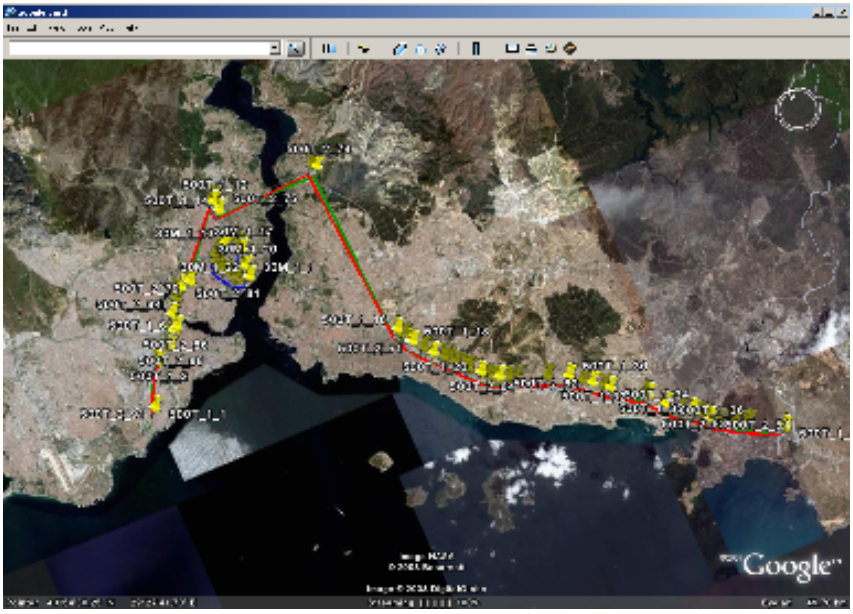


Figure 4.3 500T_1, 500T_2 and 30M_1

Algorithm 4.4 is shown in Figure 4.3. It shows 500T which is not RING and 30M which is RING and their paths are drawn on the map. Colors mean; blue is for 30M, red for forward trip and green for return trip for 500T.

Algorithm 4.4 Create kml file for two buses

```
bus=bus1, bus2;
read bus1_1, bus1_2 and bus2_1, bus2_2 from "duraklar_dosyalar"
for i=1: size(bus)
        if bus(1)_1 first stop == bus(1)_2 first stop
        bus (i)=ring
        else
        bus(i)=not;
        end
end
 if ( bus(1) ==RING )
        Read x and y coordinates bus1_1.txt from duraklar_dosyalar
        write kml file
else
        Read x and y coordinates bus_1_1, and read x and y coordinates bus_1_2
        write  kml file
end
```

Figure 4.4 shows the distance between two stops. In this figure the distance is 627,47 meters. It is not the exact distance because this distance is measured by hand.

Figure 4.4 Distances between 500T_1_31 and 500T_1_32

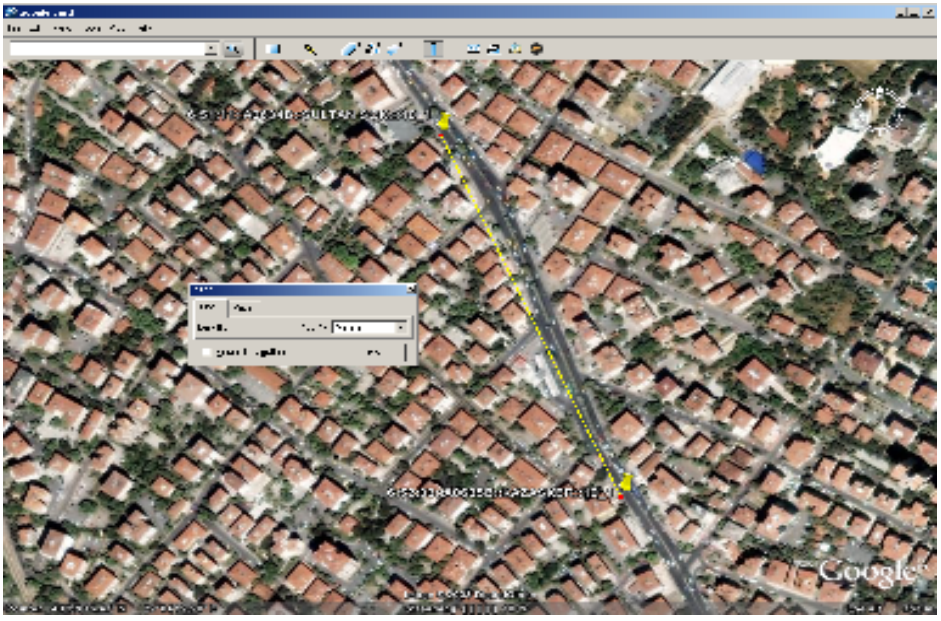


Figure 4.5 Time and distance of bus_id 10

Figure 4.5 shows the bus_id 10 in its first trip. It passes through the stops Kazasker and Sultan Sok. Their stop ids are A0635B and A0634B. The stop times are 06:51:11 and 06:52:33.

The distance between two stops is 426.47 meters and time difference is 1 minutes 22 seconds. This means that bus passes between two stops at 18,5 km/hour.

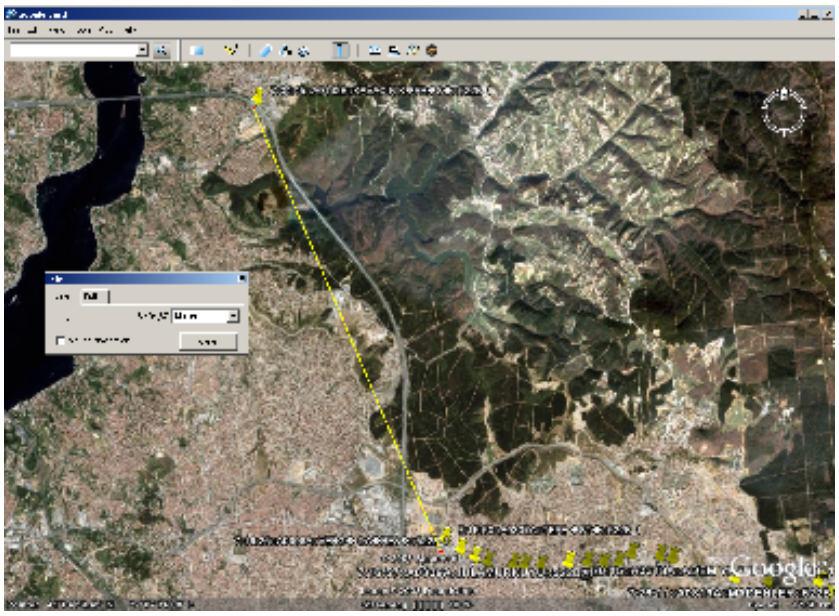


Figure 4.6 Time and distance of bus_id 522B

Figure 4.6 shows the bus with bus_id 522B. The distance between its two stops is 8404,87 meters and the travel time is 11 minutes 5 seconds. It means that bus travels through the stop with approximately 45,5 km/hour velocity.

These two velocity difference depends on the curvature of the road, waiting time on the stop and the traffic density. We ignored road curvatures and assume the paths are piece-wise linear.

4.2.3 Movement – Time Verification

We verified coordinates at the previous subsection. We generated tcl scripts in MATLAB and used ns-2 simulation tool to verify trips of busses from starting stop to the ending stop. In TCL examples, node is defined as bus. We enumerated the buses their start time.

Algorithm 4.5 shows that generation of the tcl file for given bus_ids, we wrote it to simulate node movements.

tcl(‘bus_id1’,’bus_id2’,’bus_id3’,....,’bus_idn’)

Algorithm 4.5 Generate TCL file

Determine the number of buses each bus_id
create .tcl file
read bus_id's from duraklar_dosyalar
find the maximum time
find the minimum time
find the minimum x_coordinate
find the maximum x_coordinate
find the minimum y_coordinate
find the maximum y_coordinate
find the starting and ending stops
eliminate the stops which are in the same zone radius of 50 m
each buses first time stored
time sorted
each sorted indices given to the nodes
the ordered numbers given to the nodes
set nodes in the tcl file
save maximum and minimum x and y coordinates

In Algorithm 4.5, the aim of the finding maximum and minimum values is to reduce the simulation time and simulation area. We eliminated stops (broadcasting range is available in 250 meters) which are maximum closer than 50 meters of the peron.

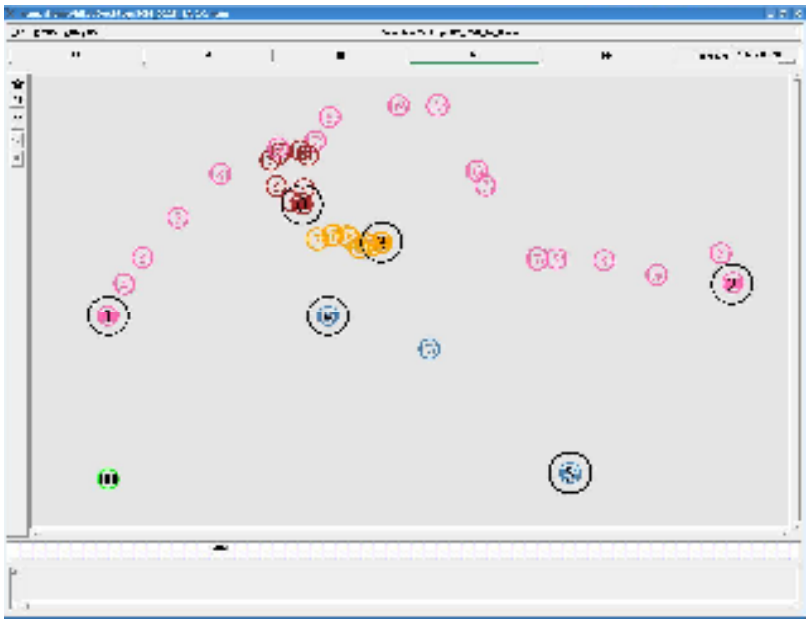


Figure 4.7 Node movement (for 522B, 10, 30M, 1A)

In Figure 4.7 shows the busses and perons. The black nodes are perons and they are stationary. The orange nodes are buses for the line "1A". The brown nodes are for line "30M". They are RING. The pink nodes are buses for the line "522B" and the blue nodes are buses for the line "10". For RING, a bus for line "1A" departs from the peron 3 and when it turns to the peron 3 it will come to rest. A new node departs from the peron 3. A bus of line "522B" departs from the peron 1 and arrives to the

peron 2 and it comes to rest at peron 2. A new node departs from peron 2, arrives to the peron 1. In Figure 17 shows the coordinates and times.

4.2.4 Broadcasting Verification

The final step of the simulation is verifying the broadcasting between nodes and data exchanging. In Chapter 3, we explained protocol creation and modification in ns-2. In this part of the Chapter 4 shows what kind of modifications and creation will be done.

1. IETT protocol is added to ns-2
2. IETT protocol is developed
3. TCL script will be written to testing the protocol.
4. Trace files format are modified to reduce size of the trace file.

4.2.5 IETT Protocol

In Chapter 3 includes how to add new protocol in ns-2.

We generated and introduced new protocol IETT to ns-2 by modifying existing ping protocol files “ping.cc” and “ping.h”

1. Defined iett packet header in “packet.h”
2. Defined new packet type in “ns-packet.tcl”
3. Defined default values in “ns_default.tcl”
4. Wrote a simple Tcl code to verify the new agent.
5. Rerun the ns-2 in order to allow changes take affect.
6. Run the Tcl code
7. Checked the nam file to verify the iett packet type.

4.2.6 Definition of the Materials

1. Copied iett.cc and iett.h as ping.cc and ping.h and we pasted them under ns-allin-one/ns-2/apps.
2. Defined iett packet type in packet.h

- enum packet_t
  ....
  PT_IETT,
  // insert new packet types here
  PT_NTYPE // This MUST be the LAST one
  };
- class p_info {
  public:
  p_info() {
  ....
  name_[PT_IETT]= "iett";
  name_[PT_NTYPE]= "undefined";
  }
- static bool data_packet(packet_t type) {
  return (
  ....
  (type) == PT_HDLC || \
  (type) == PT_IETT \
  );
- #define DATA_PACKET(type) (
  ....
  (type) == PT_SCTP_APP1 || \
  (type) == PT_IETT \
)

3. Defined default value of IETT agent in ns-default.tcl
   - Agent/IETT set packet_size 51
4. Defined new packet type in ns-packet.tcl.
   - foreach prot {

   ....
   # Other:
   ....
   IETT
   }
5. Wrote “iett.o” in Makefile
   - apps/ping.o apps/iett.o
6. Went to the directory where is the ns2, run “make” command to configure the ns-2
7. Wrote a simple Tcl code to verify the IETT agent and went to that directory where is the Tcl code was saved. Algorithm 4.6 shows simple Tcl code algorithm.

Algorithm 4.6 Simple Tcl file

Define options
Main Program,
Initialize global variables
Create God
Create the specified number of mobilenodes
"attach" nodes to the channel.
configure node
Provide initial (X,Y, for now Z=0) co-ordinates for mobilenodes
"attach" agents to the nodes
Schedule event
Tell nodes when the simulation ends

8. Wrote "ns iett.tcl" and run the ns command on the console
9. "iett.nam" and "iett.tr" will be created at the same directory as iett.tcl
10. Opened iett.nam and verified the packet type as iett, unmodified new trace file format is below

iett.tr

r –t –Hs –Hd –Ni –Nx –Ny –Nz –Ne –Nl RTR –Nw - -Ma –Md –Ms –Mt –Is –Id -It iett iIl –If –Ii -Iv

iett.nam

r –t –s –d –p iett –e –c-a –I –k RTR

4.2.7 IETT Data Exchange and Collection Protocol

In IETT agent, we used different node types and packet types.

There are two types of node as listed below

- bus
- peron

There are five types of packets as listed below

- beacon
- ack_bus (which is acknowledgement sent by a bus to the beacon of another bus)
- ack_peron (which is acknowledgement sent by a peron to the beacon of bus)
- data_diff (which is a data fragment sent to another bus)
- data_full (which is send full data to peron)

All nodes were created at the beginning of the simulation. A stop is awake during all the stages of the simulation after it is created. A bus is awake during a trip; it wakes up at the departure and sleeps with arrival at the last stop belonging to that trip. In the rest of the time the busses are asleep. Being awake means that a peron or a bus is listening to the channel. Being asleep means bus stops to listening to the channel.

By giving an energy level, which is set by the 'awake' command to the node, a node can begin listening to the channel. There are three commands, make_peron, sleep_bus, and awake_bus. "make_peron" gives the full energy to the node, thus the node behaves like a peron. "awake_bus" gives full energy to the node and "sleep_bus" gives zero energy level to the node, thus node sleeps.

When a bus departs from a peron it becomes awake and gets its measurement in every "measurementInterval" seconds.

For example;
"measurementInterval" is set to 30 seconds. We assumed that the bus departs from a peron at time zero. The bus starts to get its coordinates, on the simulation area shown by x and y, in every 30 seconds till gets 'sleep_bus' command. In trace file "60 22.4 41.2" means that the bus is on the location x=22.4 and y=41.2, at 60th second.

Bus has an array to store its measurements.
A bus broadcasts a "BEACON" in every "beaconInterval" seconds when it is awake. If any awake bus or any peron is in the range of the sender bus, they will receive that beacon.

For example;
"beaconInterval" is set to 40 seconds. We assumed that the bus departs from a stop at time zero. The bus starts to broadcast "beacon" in every 40 seconds. In trace file "40 1 2 BEACON" means node_1 broadcasted a beacon, and node_2 received that beacon at 40th second.

If a bus receives a “beacon”, it replies with “beacon_ack_bus” to the sender bus, to say that it received the “beacon” and to inform that it is a bus not a peron.
If a peron receives “beacon”, it replies a “beacon_ack_peron” to the sender bus, to say that it received the “beacon” and to inform that it is a peron not a bus.

For example;

“43 2 1 ACK_BUS” means that node_2 sends “beacon_ack_bus” to the node_1 at 43rd second.
“43 8 1 ACK_PERON” means node_8 sends “beacon_ack_peron” to the node_1 at 43rd second.
After the sender node receives “beacon_ack_bus”, the sender node chooses a random data from its array and sends it to the bus with a label named “data_diff”. When the bus receives the packet containing data_diff and some other data, it stores the data portion in its array.

For example;

```
52.0007 1 0 BEACON
52.0025 0 1 ACK_BUS
52.0067 1 0 DATA_DIFF
35.00000 3.0 8.00
```

After the sender node receives “beacon_ack_peron”, it adds all its data, stored in its array, to the packet and sends it to the peron with a label named “data_full”. When the peron receives the packet containing data_full and some other data, it writes the whole data to the trace file.

For example;

```
52.0007 4 12 BEACON
52.0025 12 4 ACK_PERON
52.0067 4 12 DATA_FULL
35.00000 3.0 8.00
65.00000 12.0 3.00
95.00000 32.0 78.00
30.00000 13.0 5.00
```

Algorithm 4.7 shows IETT packet declaration and agent tasks

Algorithm 4.7 IETT packet and agent

```
Define packet types
beacon
ack_bus
ack_peron
data_diff
data_full

Define packet header
        iett_pt
        beacon_seq_no
        beacon_ack_seq_no
        beacon_ack_dest_ip
        sensor_data data_diff
        sensor_data* data_full
        transmission time
        size_data_full;

Define events
        recv
        send_beacon
        send_beacon_ack
        send_data_diff
        send_data_full
        measure_data
        get_diff_data
        save_data

Define timers
        beacon_interval;
        measure_interval;
        data_delay


Define default values for
        beacon_interval
        measure_interval
        data_delay                              in ns_default.tcl


 command
        beacon_seq_no = 0;
        coll_data_size = 0;
        initially sleep bus and peron
        wake_bus            // The bus starts sending beacons on wake
       make_peron          //For now all the perons are listening all the time
        sleep_bus                //energy set to 0


if(awake bus or peron)

        access the iett header for the received packet
        access the ip header

        if (iett packet = beacon)
                Send ACK with the received SEQ NO
                and the beacon origin IP
        end

        if(iett packet = beacon_ack_bus)
```

Algorithm 4.7 (continuous)

```
                reply to beacon_ack
                if there is no previous reply
                and the ack is for the outstanding beacon (against late ack)
                and this node is the originator of the beacon inducing received ack

        end
        if(iett packet = beacon_ack_peron)
                reply to beacon_ack_peron
                by dumping full data
                if buffer has data empty the buffer after dump

        end
        if(iett packet = data_diff)
                Save the received data on the agent
        end

        if(iett packet = data_full)
                Save the received full data from a bus on the trace file
                if peron, ignore if bus
        end
end

send_beacon
                set beacon to iett packet header
                beacon sequence number ++
                data set to 0                   //no data send for beacon
                beacon sequence number set on the beacon packet
send _beacon_ack
                if(peron)
                        set beacon_ack_peron on the header iett packet
                else
                        set beacon_ack_bus on the header iett packet
                end

                        copy ack_num on the beacon reply
                        whose beacon it's a reply to //who takes the beacon
send_data_diff
                set data_diff on the header iett packet
                choose the data to be sent and put it in the header
                for now only one sensor data is chosen
                Set the payload size to correct the transmission time
                delay data in case multiple
send_data_full
                set data_full on the header iett packet
                assign the address only
                assign the packet size
                the number of measurements on the payload
                Set the payload size to correct the transmission time
measure_data
        get location
        add this measurement to the data collection on the node
get_diff_data
                choose random data and send

save_data
        print x_coordinate, y_coordinate, and time to the trace fil
```

4.2.8 Testing

In testing, we created new trace files in different formats for IETT packets. Trace file stores the event data as explained in Chapter 3.

First of all, two of the files under the ns-allin-one/ns2/trace directory would be modified for IETT trace file format. We modified trace file format due to the fact that it has lots of data inside of it, and gets larger while simulating it for hundreds of nodes. To protect the file against getting larger, we write the file in a new format and minimized it. We wrote only the necessary data such as time, packet type and coordinates.

Algorithm 4.8 shows the creation of the IETT trace file and Algorithm 4.9 is the prototype of IETT trace file format.

Algorithm 4.8 cmu_trace.cc

```
include iett.h
void CMUTrace::format
        if( packet_type=IETT)
        call format_iett
        else
        call other formats
        end
end
void CMUTrace::format_iett // at the end of the file
        if( operation= receive & sender_ip != receiver_ip)
                print packet reception time, packet src, packet receiver, packet type
        if( iett_packet=data_diff)
                        display measurement time, x_coordinate, y_coordinate
                end
                if(iett_packet = data_full )
                  print packet reception time, packet src, packet receiver, packet type
                                for i=0 ; i< size_data_full; i++
                                        display measuerement time to "myfile.tr",
                                        x_coordinate,y_coordinate
                                 end
                end
        end
end
```

Algorithm 4.9 cmu_trace.h

```
        void   format_iett(Packet *p, int offset) // at the end of the file
```

Chapter 5
Experimental Results

The demanded observation was to see which coordinates, the peron has knowledge about, in Istanbul, and at any time we want. To achieve this, we analyzed the trace file and plotted the result on the map of Istanbul, by using the necessary data. We implemented this, using two functions; read_trace and istanbul_map, in MATLAB.

We all need is in "my file.tr", we generated different file to write the data which is dumped to the Peron, and analyzed it easily, and we controlled it because the file include the information of peron and the data which are dumped at when to where.

There are two cases for function read_trace. In the first case, there are two parameters, a peron number and a time. The task, in this case, is to collect the data of the trace file of the given peron, from the beginning of the trace file to the given time. The second function parameters are peron name, time1 and time2. The task, in this case, is to collect the data of the given peron in the time interval, between time1 and time2.

Algorithm 5.1 Read trace file

```
read_trace(parameters)
min_time=load from the tcl creation function.
min_x_coordinate= load from the tcl creation function.
min_y_coordinate= load from the tcl creation function.
        if (size(parameter)=2)
                read the trace file
                find data_full which is dumped to the given peron
                check the its time which is smaller and equal the given time
                save the x and y coordinates in A
                break;
        end
        if (size(parameter)=3)
                read the trace file
                find data_full which is dumped to the given peron
```

Algorithm 5.1 (cont'd)

```
                check packet sending time which are between time2 and time1
                save the x and y coordinates in A
        end
end
```

To reduce the simulation time, we found the smallest time, min_time, while generating the tcl file, and subtracted this from all the times. In addition to this, to reduce the simulation area, we found the smallest coordinates, min_x and min_y, and subtracted them from all the coordinates in tcl file.

To work with correct information, we consider the min_time while observing the trace file by subtracting it from the given times, and select the proper lines. While analyzing the coordinates from trace file, to ignore redundancy, we only add min_x and min_y to the selected lines of the trace file, and plotted the coordinates to the appropriate place on the map of Istanbul.

Algorithm 5.2 Draw Istanbul map

```
Read file which contains border coordinates of Istanbul
Draw patch of Istanbul
Read A
Plot A on the Istanbul map
```

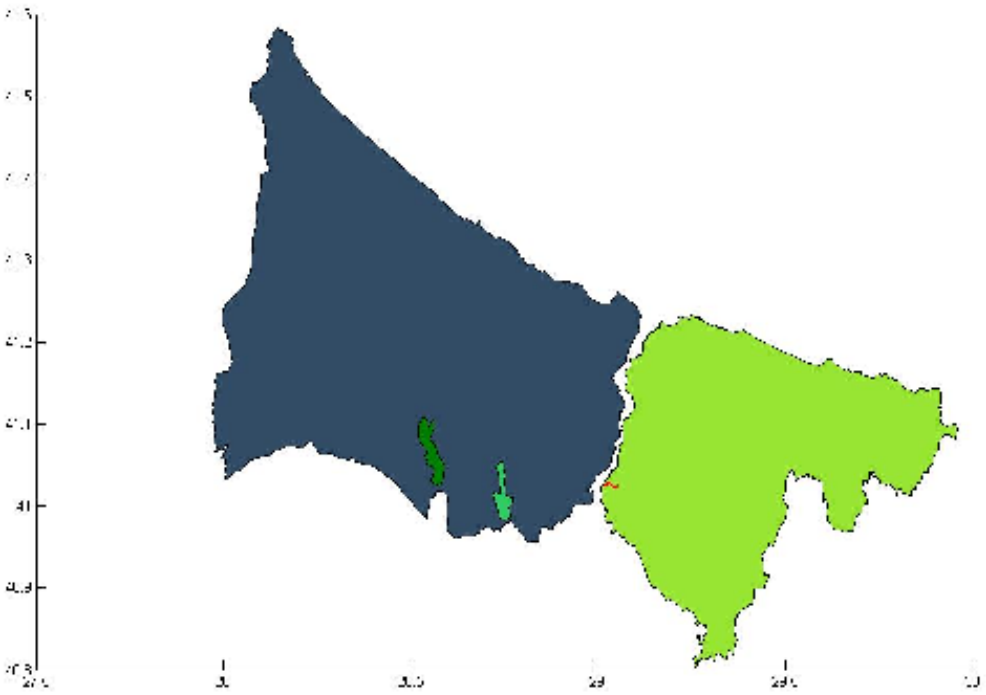


Figure 5.1 Istanbul map

Istanbul earth coordinates are considered on Figure 5.1 which is drawn by using patch function of Matlab. On the Figure 5.1, the horizontal line values are longitude and the vertical line values are latitudes of earth. Coordinate computation of Istanbul is done by considering of explanation of Istanbul latitude and longitude distances which is explained in Chapter2.

Blue part of the Figure 5.1 symbolizes the European side of Istanbul and green part for Asian side. Dark green and light green symbolizes Büyükçekmekce Lake and Küçükçekmece Lake. The lakes are drawn to show the accuracy of the plotting of the perons and path of the bus-lines, it proves that the stations and paths are not putting on the wrong places.

In this project, there are 643 bus lines and 48164 movements on Istanbul. *Movement* means each arrival to station and departure from station. For example, if bus arrival to station it means this is one movement and depart from station this is the second movement of the bus.

We used 48 bus lines in our simulation and made sure that the bus lines are from different sides. We chose bus lines whose path is inside Asian side, European side or across two sides. The total number of movement for chosen bus lines is 4465 and the total number of the peron is 34. Additionally, we chose bus lines whose trip type is RING. We used the Algorithm 4.5 when we decided to define perons and this algorithm prevent the redundant dumped data.

A video is generated by using Matlab tool because when we watch the video we can observe the changes the density of data which is dumped to the selected peron. The video generation is explained in Algorithm 5.3.

Algorithm 5.3 Create .avi file for selected peron

```
min_time=load from the tcl creation function.
max_time=load from the tcl creation function.
F = from min_t to max_t by increasing 3600 sec
Read myfile.tr to file
for all lines in file
  marked the selected peron
  marked time of dumped data
 insert each time in  F
```

Algorithm 5.4 (continuous)

```
end
for all index in F
plot each data which is in the index of the F
75 frames for each figure
end
```

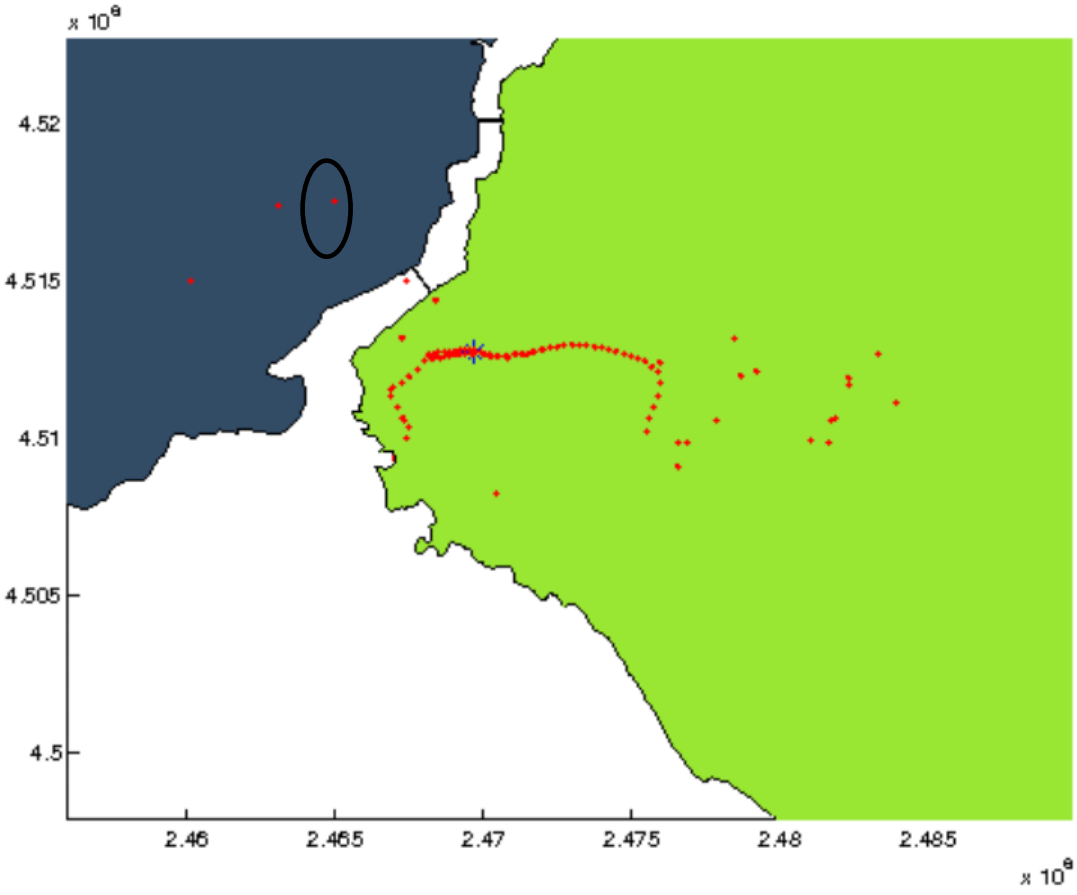


Figure 5.2 Dumped Data on selected peron Asian side

In Figure 5.2 shows that the dumped data to the selected peron which is in the Asian side at Üsküdar. The video is zoomed by the max coordinate value of the dumped data without showing all Istanbul by adding new algorithm in Algorithm 5.3. In Figure 5.2, blue star represent the selected peron, red dots are dumped data to the selected peron. The consecutively dots represent the path of the bus line which arrive to the selected peron. Otherwise the other dotted, one of them is in the black circle, represent the data which are exchanged among the buses that cross each other. For example, a bus passes from the dot, in black circle, doesn't pass from the selected peron in anyway but it exchanges the data when see the other bus which dumped its data to the selected peron. Also other scattered dots represent the exchanged data.

Two simulations were implemented for this project. Those are accomplished by varying of the parameters which are transmission range, beacon interval time and measurement interval time. At the below, there are two figures and in those figures

selected perons and the time of the day are the same, reason is accuracy of observation.

The first simulation we used the 550 meters for transmission range, 350 seconds for the beacon interval time which is time period of sending beacon and 60 seconds for the measurement interval time that time period of the location information of the bus, node. In the second simulation, the parameters are 78 meters for transmission range, 121 seconds for beacon interval time, and 30 seconds for measurement interval time.

We selected the bus line whose trip type is RING and its path is on European side and it doesn't pass any place which is Asian side or south part of the European side. Selected figure is from the video and its time is in the afternoon.

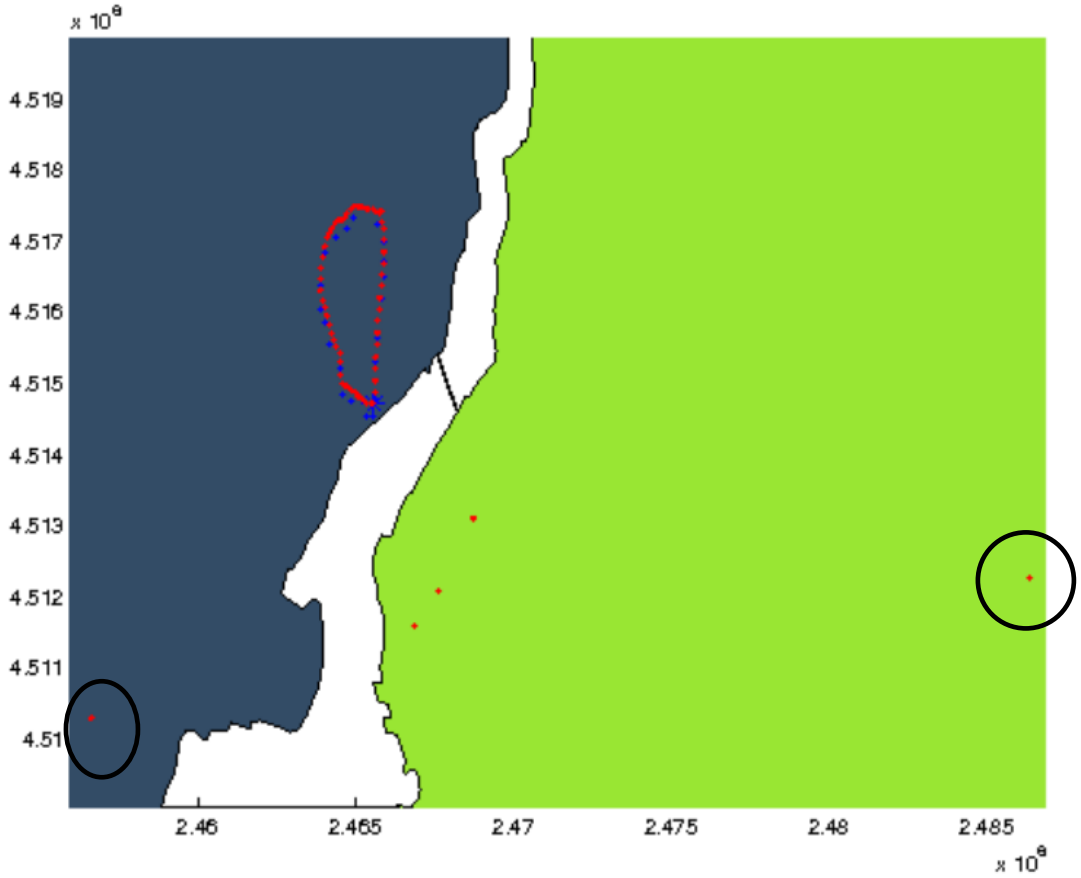


Figure 5.3 Dumped Data on selected peron European side-wide transmission range

In Figure 5.3, blue dots denotes the route of the bus line which arrives to the chosen peron, blue star denotes the selected peron, red dots outside the blue ring denotes the

exchanged data. It shows that the bus receives the data four times at the selected time.

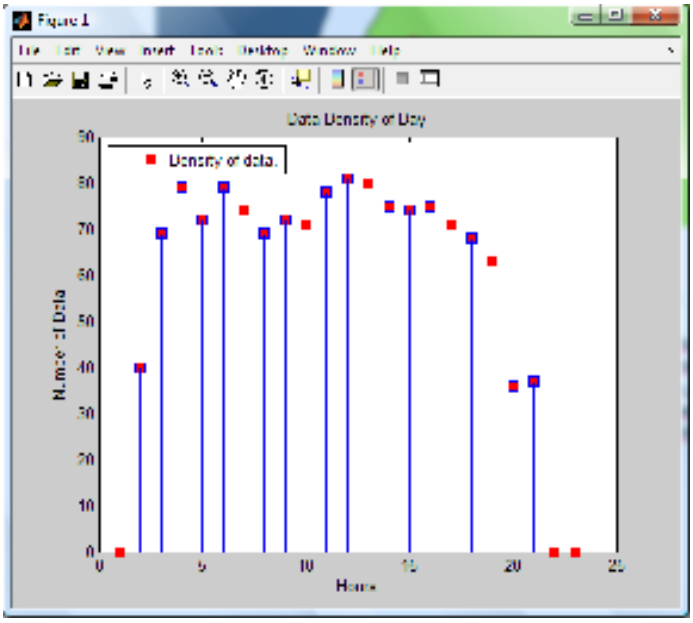


Figure 5.4 Data Density of the day of 30M

Figure 5.4 illustrates density of the dumped data, in a day. Collected data is dumped to the chosen peron in Figure 5.3. Horizontal line shown in figure denotes the hours. Hours are defined by the earliest departure and latest arrival time of movement. Each segment corresponds to one hour except the last segment. Vertical line shown in figure denotes the number of data that is dumped to the chosen peron. Density of the dumped data increases peak hours of traffic and at earliest and the latest time of the day the dumped data getting zero, it means at that times no movement for buses which depart to the chosen peron. By the way, vertical line includes the value 40, it means minimum value of the dumped data is 40 and it shows that the bus/buses dumped their data which are at least their path coordinates.

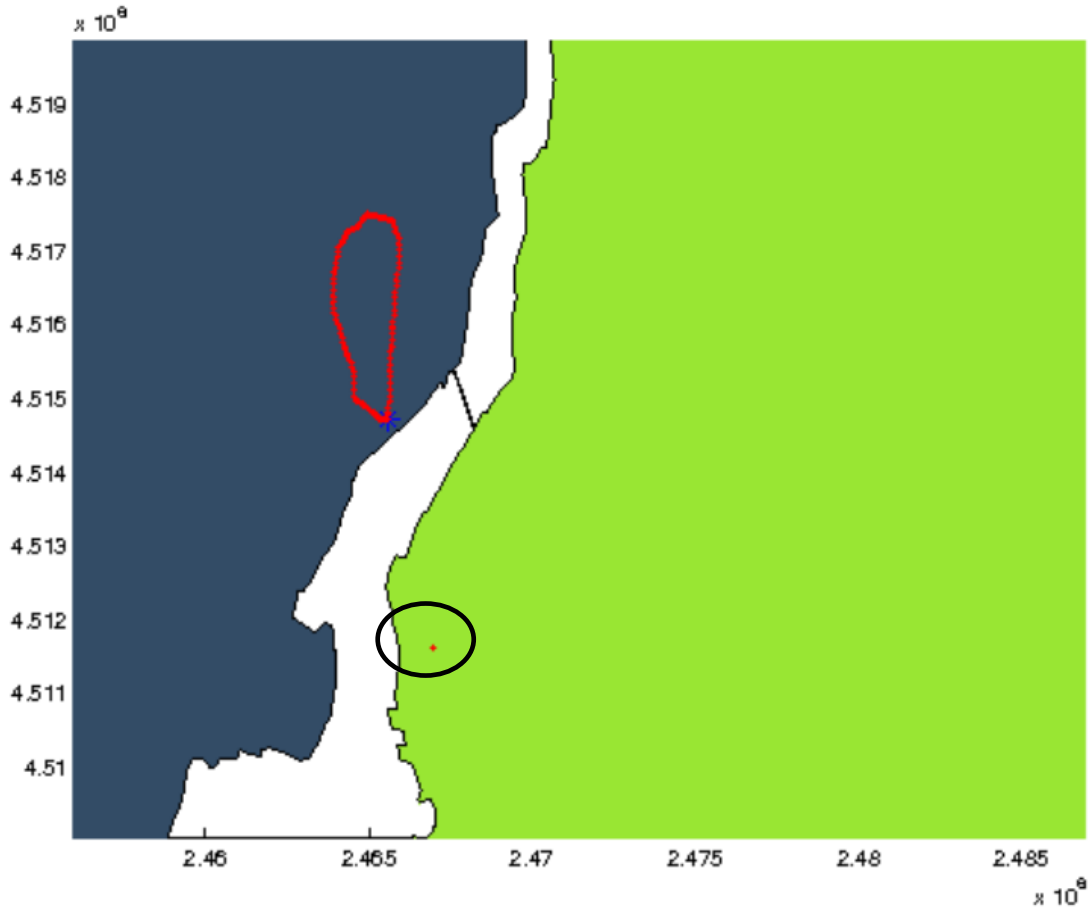


Figure 5.5 Dumped Data on selected peron Europe side-small transmission range

Figure 5.5 is generated from the second simulation and second parameters. It shows that the bus receives the data one time at the selected time.

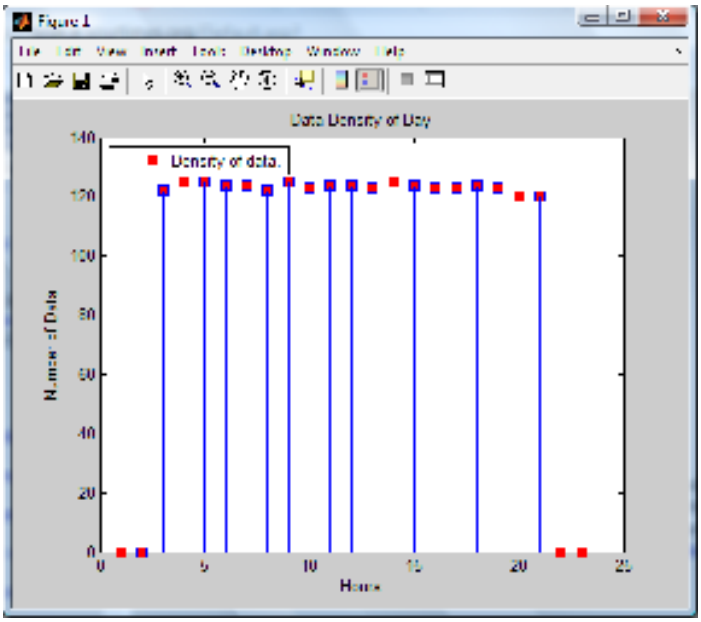


Figure 5.6 Data Density of the day of 30M

Figure 5.6 illustrates density of the dumped data, in a day. Collected data is dumped to the chosen peron in Figure 5.5.

Figures 5.6 and 5.4 illustrate the dumped data and these two graph is generated by using different parameters and those are explained at the above. These two graphs show if reduce the transmission range, number of collected data will reduce even though reduce the sending beacon time. If reduce the measurement time you can get the data, which is not the transmitted data, is buses' own routing data. Because the bus is communicating with fewer bus reason of transmission range.

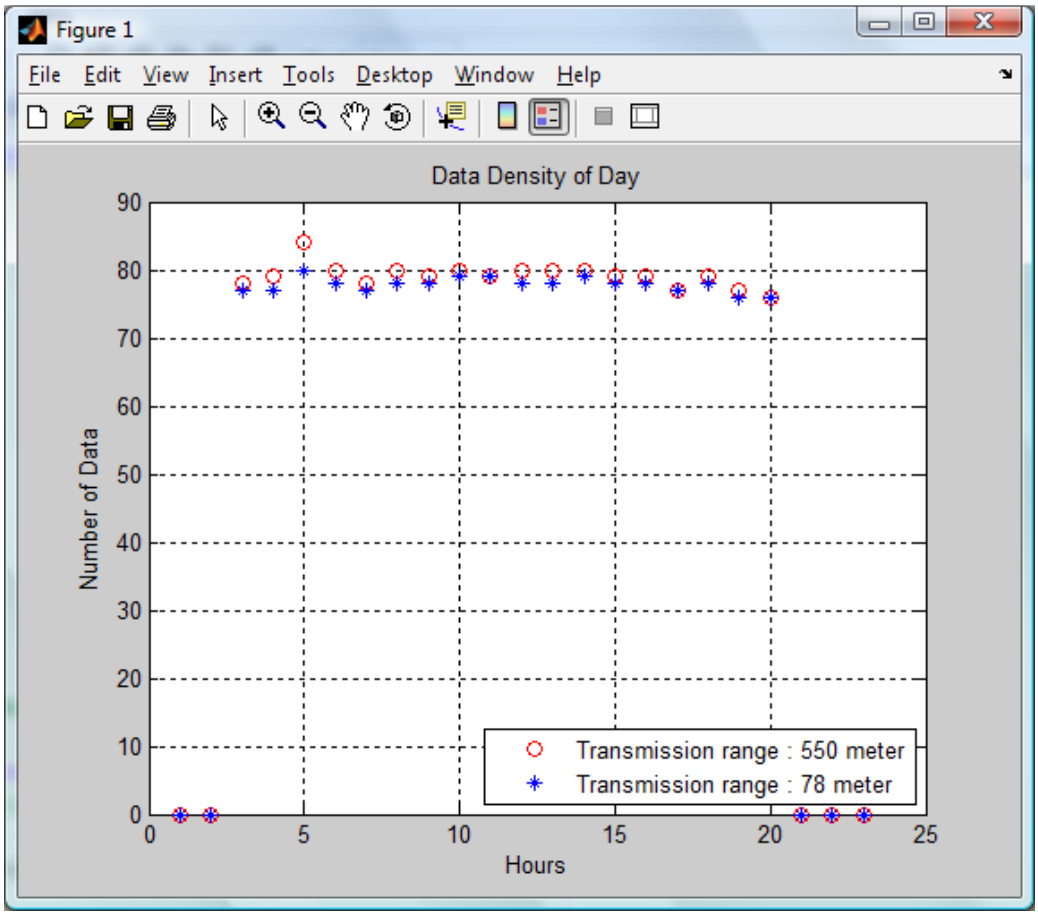


Figure 5.7 Dumped data with long time scale

Figure 5.7 denotes that the beacon interval time and measurement interval time values are same but the transmission ranges are different. Hence number of collected data depends on the transmission range of the broadcasting. For this figure sending beacon interval time is 350 seconds and measurement interval is 60 seconds. When transmission range is getting large number of the collected data is increasing.

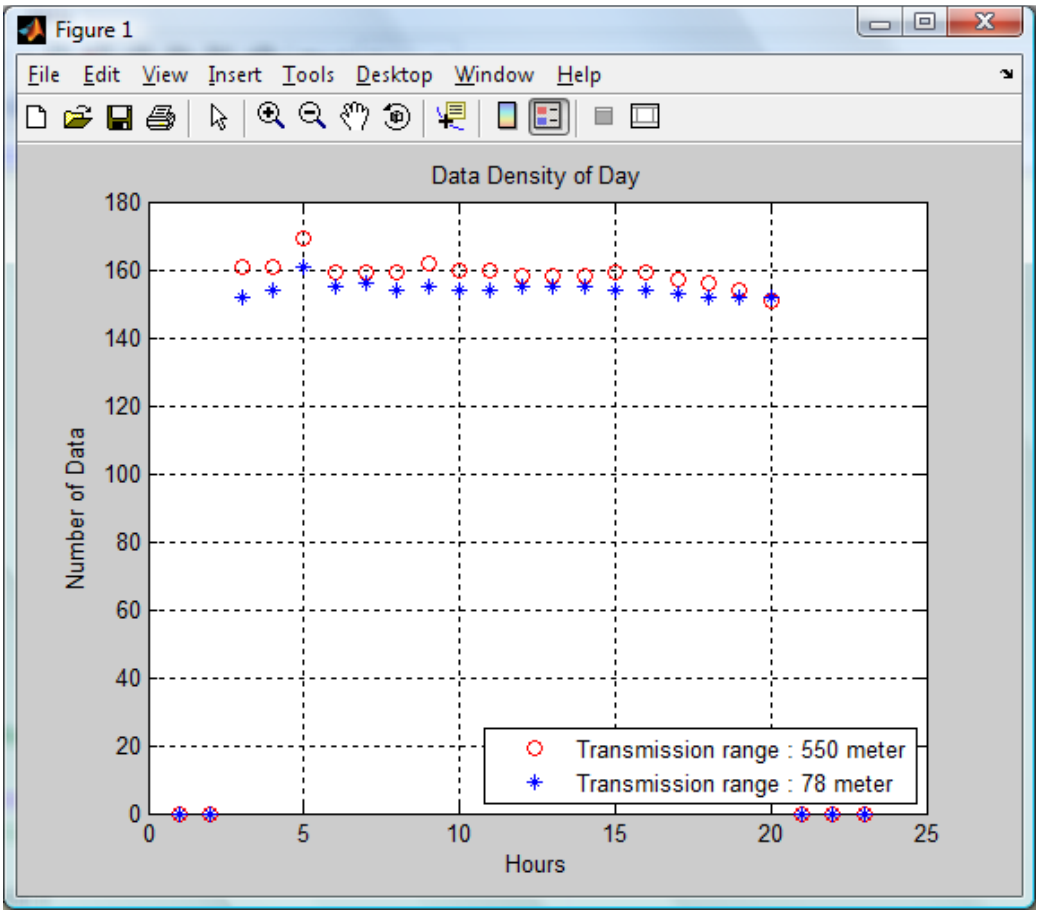


Figure 5.8 Dumped data with short time scale

Figure 5.8 denotes that the beacon interval time and measurement interval time values are same but the transmission ranges are different. Hence number of collected data depends on the transmission range of the broadcasting. For this figure sending beacon interval time is 121 seconds and measurement interval is 30 seconds. When transmission range is getting large number of the collected data is increasing.

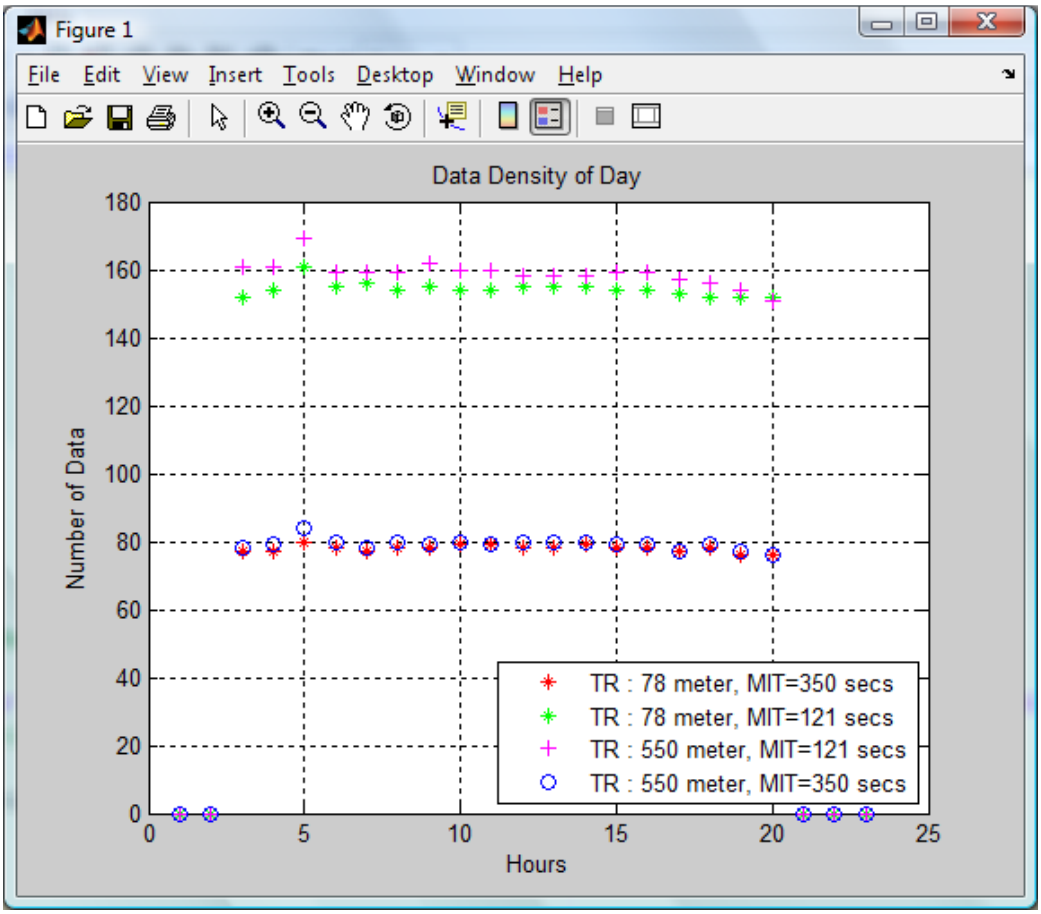


Figure 5.9 Dumped data with different values

TR: Transmission Range

MIT: Measurement Interval Time

Figure 5.9 illustrates that the number of collected data depends on transmission range and the time of the sending beacon. Number of exchanged data depends on the transmission range of broadcasting and beacon interval time, on the other hand collected data depends on the measurement interval time.

Chapter 6
Conclusion

In this thesis, we proposed a new protocol for gathering data about bus traffic of Istanbul on VANET. In this section we summarize the techniques and some suggestions for the future work.

We provided the concrete statistical data that have been saved after communication between vehicles, depending on time schedule of them. We used tools, MATLAB, Google Earth, and ns-2, to gather useful data and we analyzed them. Vehicular ad hoc network structure was used to create network among vehicles. Bus data simply consists of position and time. Each bus broadcasts periodically to all neighbors. In order to achieve this, we wrote a new protocol and modified it according to our requirements. After gathering the data, we tested the new protocol, among nodes, and implementation of it in simulation program (ns-2). Finally, the results showed that an arbitrary stop could have data about any location, not only the locations near that stop, at a specified time. We produced plots showing the quality of collected data over time.

References

[1] Tüfekçioğlu, F., *Mobility Aware, Reliable Ad Hoc Routing Protocols*, M.S. Thesis, Istanbul Technical University, 2005.

[2] Bernsen, J., Manivannan , D., *Unicast routing protocols for vehicular ad hoc networks: A critical comparison and classification,* Pervasive and Mobile Computing 5 (2009).

[3] Lochert, C., Hartenstein, H., Tian, J., Hermann, D., Wauve, M., A "Routing Strategy for Vehicular Ad Hoc Networks in City Environments", *IEEE Intelligent Vehicles Symposium,* 156-161 (2003).

[4] Bai, R., and Sighal, M., "DOA: DSR over AODV Routing for Mobile Ad Hoc Networks", *Mobile Computing, IEEE Transactions on***,** 1403-1416 (2006).

[5] Sun, M., Feng, W., Lai, T., Yamada, K., and Okada, H., "GPS-Based Message Broadcasting for Inter-Vehicle Communication, Parallel Processing", *2000 Proceedings. 2000 International Conference on*, 279 - 286 (2000).

[6] Bayılmış, C., Ertürk, İ., Çeken, C., Bandırmalı, N.*, "*DSR ve AODV MANET Yönlendirme Protokollerinin Başarım Değerlendirmesi", *Elektrik, Elektronik, Bilgisayar Mühendisliği 11. Ulusal Kongresi, EMO*, 280-283, İstanbul, Türkiye, 2005.

[7] MapXL Inc., *Maps of World,*
http://www.mapsofworld.com/lat_long/turkey-lat-long.html, 2008.

[8] The MathWorks, Inc., *MATLAB - The Language of Technical Computing*
http://www.mathworks.com/products/matlab, 2008.

[9] Google, *Google Earth*,
http://earth.google.com/intl/en_uk/, 2008.

[10] Google, *KML Tutorial*,
http://code.google.com/apis/kml/documentation/kml_tut.html,2008.

[11] Google, *Google Earth User Guide*
http://earth.google.com/intl/en/userguide/v4/, 2008.

[12] A Collaboration between Researchers at UC Berkeley, LBL, USC/ISI, and Xerox PARC, *The ns Manual*, March 22, 2008.

[13] Tcl Developer Xchange, *Learn More About Tcl/Tk,*
http://www.tcl.tk/, 2008.

[14] Downward, Ian T., *How to Simulate Sensor Networks in ns-2*, Naval Research Laboratory Code 5523 4555 Overlook Ave Washington DC, 20375-5337 (2003).

[15] Chung, J., Claypool, M., *NS by Example*, WPI Computer Science, http://nile.wpi.edu/NS/ [08/01/2002 9.09.25], 2008.

[16] Chung, J., Claypool, M., *Add New Application and Agent*, WPI Computer Science, http://nile.wpi.edu/NS/new_app_agent.html, 2008.

[17] Choi, N., *Introduction to ns-2,* Oct. 11, 2007
www.utdallas.edu/~venky/acn/ns2.ppt,2008

Appendix A Appendices

A1. CD-ROM includes the source code, simulation video, soft copy of the thesis in .doc and .pdf types and presentation.

Curriculum Vitae