PROIECT DE LICENTA

EZCab:
Aplicatie VANET pentru firmele de taxi

Conduca\tori stiintifici:
Prof.Dr.Ing. Valentin Cristea
Ing.Dr. Ciprian Dobre

Autor:
Sinziana-Petronela Mazilu

Bucuresti 2009
University Politehnica of Bucharest
Faculty of Automatic Control and Computer Science,
Computer Science Department

GRADUATION PROJECT

EZCab: A VANET application for taxi booking

Scientific Advisers:  
Prof.Dr.Ing. Valentin Cristea  
Ing.Dr. Ciprian Dobre

Author:  
Sinziana-Petronela Mazilu

Bucharest 2009
Man became truly mobile when he invented the wheel.
# Contents

1 Introduction .............................................. 1

2 Related Work ........................................ 4
   2.1 VANET ........................................ 4
   2.2 Cab Booking based on SMs .......................... 5

3 EZCab General Architecture ......................... 8
   3.1 Peer-to-peer based Architecture .................... 9
   3.2 Gateway based Architecture ........................ 10
   3.3 Emission nodes based Architecture .................. 12
      3.3.1 Fixed emission nodes .......................... 12
      3.3.2 Dynamic emission nodes ....................... 12
   3.4 EZCab Routing Protocol ............................. 15
      3.4.1 Peer-to-peer Architecture Protocol .............. 15
      3.4.2 Gateway and emission nodes based Architecture Protocol .......................... 16
      3.4.3 Entities behavior ................................ 17
      3.4.4 Short EZCab Booking Protocol ................. 19
   3.5 EZCab Communication Messages .................... 20
   3.6 EZCab Routing Algorithm ........................... 23
   3.7 EZCab Security .................................... 24

4 EZCab Implementation Details ....................... 26
   4.1 VNSim .......................................... 26
      4.1.1 Maps ...................................... 26
      4.1.2 Traffic Simulator ............................. 27
      4.1.3 Scenario Generation ......................... 28
      4.1.4 VNSim Models ................................ 29
      4.1.5 DSRC Protocol Implementation ................. 30
CONTENTS

4.2 EZCab Implementation ........................................... 31
   4.2.1 EZCab Entities and EZCab Communication Protocol .... 32
   4.2.2 EZCab Communication Messages ......................... 34

5 Simulations and Results .............................. 37
   5.1 Simulation Scenarios ........................................ 37
   5.2 Results .................................................. 41
      5.2.1 Intersection Scenario ................................. 41
      5.2.2 Politehnica University Campus Scenario ............. 46
      5.2.3 Downtown Scenario .................................. 51

6 Conclusions and Future Work ...................... 60

Bibliography ................................................. 64
3.1 Peer-to-peer EZCab Architecture . . . . . . . . . . . . . . . . . . . 10
3.2 Gateway based EZCab Architecture . . . . . . . . . . . . . . . . . . 11
3.3 Emission points based EZCab Architecture . . . . . . . . . . . . . 13
3.4 Differences in forwarding messages . . . . . . . . . . . . . . . . . . 13
3.5 Entity behavior . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18
3.6 States of clients and cabs . . . . . . . . . . . . . . . . . . . . . . . . 19
3.7 EZCab Messages . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20
4.1 VNSim Architecture . . . . . . . . . . . . . . . . . . . . . . . . . . 27
4.2 Map Module . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 29
4.3 DSRC Protocol Implementation . . . . . . . . . . . . . . . . . . . . 30
4.4 VNSim User Interface . . . . . . . . . . . . . . . . . . . . . . . . . . 31
4.5 EZCab Implementation . . . . . . . . . . . . . . . . . . . . . . . . . 32
4.6 EZCab Messages Implementation . . . . . . . . . . . . . . . . . . . 35
5.1 Intersection Scenario . . . . . . . . . . . . . . . . . . . . . . . . . . 38
5.2 Politehnica Campus Scenario . . . . . . . . . . . . . . . . . . . . . . 39
5.3 Downtown Scenario . . . . . . . . . . . . . . . . . . . . . . . . . . . 40
5.4 Booking time for clients generation probability of 10% . . . . . . . 41
5.5 Booking time for clients generation probability of 33% . . . . . . . 42
5.6 Booking time for clients generation probability of 50% . . . . . . . 42
5.7 Booking time for clients generation probability of 66% . . . . . . . 43
5.8 Differences between average time, weighted average time and maxi-
mum time values in Intersection Scenario . . . . . . . . . . . . . . . 44
5.9 Rate of request messages still routed into the network for Intersection
Scenario . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 44
5.10 Booking time for clients generation probability of 10% . . . . . . . 46
5.11 Booking time for clients generation probability of 33% . . . . . . . 47
LIST OF FIGURES

5.12 Booking time for clients generation probability of 50% ........... 47
5.13 Booking time for clients generation probability of 66% ........... 48
5.14 Differences between average time, weighted average time and maximum time values in Politehnica University Campus Scenario ..... 49
5.15 Rate of request messages still routed into the network for Intersection Scenario ........................................... 50
5.16 Booking time for clients generation probability of 10% - 200 entities/hour/lane ................................................. 52
5.17 Booking time for clients generation probability of 33% - 200 entities/hour/lane .................................................... 52
5.18 Booking time for clients generation probability of 50% - 200 entities/hour/lane ....................................................... 53
5.19 Booking time for clients generation probability of 66% - 200 entities/hour/lane ....................................................... 53
5.20 Booking time for clients generation probability of 10% - 800 entities/hour/lane ......................................................... 54
5.21 Booking time for clients generation probability of 33% - 800 entities/hour/lane ......................................................... 54
5.22 Booking time for clients generation probability of 50% - 800 entities/hour/lane ......................................................... 55
5.23 Booking time for clients generation probability of 66% - 800 entities/hour/lane ......................................................... 55
5.24 Differences of average values for booking time in Downtown Scenario ......................................................... 57
5.25 Differences of weighted average values for booking time in Downtown Scenario ................................................... 57
5.26 Differences of maximum values for booking time in Downtown Scenario ......................................................... 58
5.27 Differences in rate of request messages still routed into the network for Downtown Scenario .................................. 58
Abstract

Finding a taxi in current urban scenarios generally means calling to an operational center of a taxi company, provide ones location and letting the operator search for an available car. EZCab is an application based on VANET architecture that solves the booking cab problem without using an operational center. With EZ-Cab taxi companies provide easily to their clients the closest available taxi based on direct communication between cars while using devices running DSRC or 802.11 wireless protocols. In this way, clients are able to send their request directly to the taxi network, using a cell phone, and have a fast response, without any human intermediaries. EZCab discovers and books available taxis using only cab-to-cab wireless communication. The benefits include cost effectiveness, time saving, security and scalability, because EZCab entities are completely decentralized. In this work we present the architecture of EZCab and its subsystems. We also present a pilot implementation of the solution, together with evaluation performance results.
Chapter 1

Introduction

One of the most challenging problems for taxi companies is to find the closest cabs for their clients. Finding a taxi in current urban scenarios generally means calling to an operational center of a taxi company, provide ones location and maybe, if operator finds an available taxi, client will have a response in proper time. This method has some drawbacks: it is not sure that chosen taxi is the closest available taxi to client and time of arrival is not known for sure. Human factor is very relative and in this case is the main communication link. On the other hand, we learn that computers are safe.

In past several years appeared the vehicular mobile ad-hoc networks (VANET) concept, an appropriation of mobile ad-hoc networks (MANET) for vehicles, and have been purposed various routing algorithms (8) (14) for this type of networks. But technology was not mature enough to support applications based on VANET infrastructure in real world. Recently started deployment for products based on IEEE 802.11 standard protocols or Bluetooth (a low-cost alternative to IEEE 802.11 family of protocols).

EZCab is a solution for booking cab problem, a real-life ubiquitous computing application built over VANET, based on mobile cab-to-cab message interchanges. Humans are eliminated as part of communication: client makes a request using PDA or cell phone and receives response with proper taxi solution for time and distance. EZCab is designed to book nearby cabs in densely populated urban areas.

Idea on which EZCab is based is simple: a taxi car receives a message request from a client. If taxi is available, it sends a message to the client showing its
availability; otherwise it sends the message further to the other taxis.

Client requests are forwarded in taxi-network by cabs until some available taxis receive the request message. Client receives the taxi responses forwarded by network entities.

Although the traditional centralized solution of cab dispatching based on operational centers is guaranteed, this solution has some issues like:

- Client requests have to go through one or multiple cab dispatchers, which introduce waiting time for the clients, especially during periods of peak cab requests.

- In order to dispatch the nearest cab to the client, all cabs in the city have to be monitored to find the closest one to the client’s location.

The EZCab solution comes to resolve these problems, being simple, faster and scalable, due to its completely decentralized idea of communication, based on direct communication between entities and using only short-range wireless network interfaces.

The EZCab solution is possible due to two technology trends:

- PDA and cell phones became powerful mobile computers equipped with short range wireless capabilities.

- The presence of GPS receivers, wireless network interfaces and embedded systems in taxi vehicles.

EZCab is not a new idea: an implementation based on Cooperative Computing model with Smart Messages is available (13). Cooperative Computing is a distributed computing model based on execution migration, where applications are collection of Smart Messages and each node cooperates by providing a common system support (12). Smart Messages are migratory execution units that execute on nodes of interest named by content and reached using self-routing at intermediate nodes.

Due to the time of SM computing, older prototype of EZCab computes requests in a large area of time (13). Unlike previous work, we focus on the idea of simplifying the cab-to-cab communication protocol. We change SMs with simple messages containing only useful information data, without execution units. Every entity in the ad-hoc network has a PDA with EZCab application and will be able to compute data received.
The following chapters are organized as follows:

In *Related Work* chapter, first we speak about facilities of VANETs and applications that can be deployed for such networks. Next, we present a cab booking application that uses Smart Messages. Old version of EZCab was implemented by Rutgers University and is based on idea of SMs routing into the network.

In third chapter we present the EZCab architecture, which includes three types of purposed architectures: peer-to-peer based architecture, gateway based architecture and emission node based architecture. We also speak about EZCab routing protocol in each type of architecture, about EZCab communication messages and EZCab routing algorithm. Finally, we present some ideas related to security of EZCab application.

*EZCab Implementation Details* chapter has two sections: one with VNSim traffic simulator ideas and details of current implementation, and second with details of implementation for EZCab over VNSim. EZCab is designed to be an integrated module for VNSim. We present implementations of EZCab entities, communication messages, EZCab booking protocol and how communication between clients and taxis is simulated in VNSim.

In the *Simulation and Results* chapter we present several evaluation scenarios of EZCab application: a simple one, Intersection scenario and two real life scenarios, based on a map of a Bucharest area and a map of the downtown of Manhattan. In the second part we present and analyze the results from tests based on these three scenarios.

Finally, in the last chapter, *Conclusions and Future Work* we speak about EZCab application and the importance of EZCab test results comparative with results obtained with other booking cab methods and we purpose some extensions to our EZCab implementation.
Chapter 2

Related Work

2.1 VANET

VANET (Vehicular Ad-Hoc Network) is a form of Mobile ad-hoc network, to provide communications among nearby vehicles and between vehicles and nearby fixed equipment, usually described as roadside equipment.

In VANETs vehicles tend to move in an organized fashion. The interactions with roadside equipment can likewise be characterized fairly accurately. Most vehicles are restricted in their range of motion, for example by being constrained to follow a paved highway.

A VANET tends to operate without any infrastructure or legacy client and server communication. Each vehicle equipped with VANET device will be a node in the ad-hoc network and can receive and relay others messages through the wireless network. Collision warning, road sign alarms and in-place traffic view will give the driver essential tools to decide the best path along the way. To this end a special electronic device will be placed inside each vehicle which will provide Ad-Hoc Network connectivity for the passengers.

There are also multimedia and internet connectivity facilities for passengers, all provided within the wireless coverage of each car. Automatic payment for parking lots and toll collection are other examples of possibilities inside VANET.
2.2 Cab Booking based on SMs

EZCab idea is based on the older version of EZCab, implemented by Rutgers University, New Jersey (13).

The idea of vehicle-to-vehicle communication would not be possible if mobile phones and PDAs won’t become relatively powerful data computers, having wireless capabilities.

EZCab consists of a mobile ad-hoc network of computers embedded in taxis and client handhelds, which communicate through short-range wireless networks such as IEEE 802.11 or Bluetooth. Instead of booking cabs through a centralized dispatcher, EZCab clients book available cabs by communicating directly with other EZCab nodes over the mobile ad-hoc network.

Existing cab booking systems (2) use fixed infrastructures such as Internet or cellular phone networks to relay client requests to a centralized dispatcher and track the location of cabs for optimal dispatch.

To minimize the size of routing table, EZCab use a probabilistic measure to indicate the probability of finding an available cab through each next-hop neighbors. EZCab use geographical routing algorithms (9) (10) (7), and this eliminates the idea of maintaining routes from sender to the destination, forwarding decisions being based on local knowledge. There are implemented three routing algorithm for finding available cabs: flooding, probabilistic-on-demand and probabilistic proactive. All the three algorithms rely on an underlying broadcast mechanism and assume symmetric links.

Flooding is the basic mechanism to propagate messages in many MANET. In this scheme, each EZCab client request is broadcasted to all its neighbors recursively, up to a maximum number of hops or until it arrives at an available cab node, whichever comes first. There are different mechanisms to prevent routing loops. In very dense networks, flooding algorithm does not work very well due to unavoidable wireless contention.

Probabilistic-on-demand algorithm builds routing tables on demand for nodes lying in a given number-hops range from client. The routing table of a node consists of entries with the probability of finding an available cab through the nodes neighbors. Since the route to each neighbor could be invalidated due to mobility, each routing entry records its last update time; the system uses this information to age routing entries. The Probabilistic On-Demand mechanism is similar to the
reactive routing protocols in MANET such as AODV (14). The overhead of the EZCab routing scheme is expected to be lower than that of Flooding when nodes move slowly and many requests occur within a limited region and time interval. Also, network gives scenarios when routing tables are reconstructed from scratch, a relatively expensive process.

Probabilistic-proactive is similar to the probabilistic-on-demand scheme, building routing tables with probabilities to find available cabs in each entry. The routing updates are triggered proactively by the presence of available cabs, not by on-demand requests from client stations.

EZCab protocol consists in two phases: Cab booking and Validation. Cab booking is a three-way handshake protocol in which client and driver stations collaborate with each other to forward the clients request through the network until an available cab is discovered. When the booked cab arrives at the vicinity of the client, the driver initiates the Validation phase to mutually authenticate with the client. The Validation protocol uses a challenge-response scheme based on public-key cryptography. The client and the driver stations exchange their public keys during the three-way handshake Cab Booking protocol.

EZCab implementation is based on Smart Messages, a middleware architecture based on execution migration, similar to mobile agents, based on Cooperative Computing Model. Cooperative Computing is a distributed computing model based on execution migration, where applications are collection of Smart Messages and each node cooperates by providing a common system support (12). SM defines architectures for programming distributed applications over mobile ad-hoc networks. An SM is a distributed application consisting of code, data, and a lightweight execution state. Instead of transferring data among nodes involved in computation, SMs migrate the execution to each of these nodes. An SM carries routing code and routes itself at each node in the path toward a node of interest. Each node cooperates to support the SM execution by providing a virtual machine (VM) for execution over heterogeneous platforms, a shared memory addressable by names (tag space) for inter-SM communication and synchronization, and a code cache for storing frequently executed code.

SMs name nodes by tag names and migrate to nodes of interest using a high level migration function that implement routing. The routing is executed at each node on the path toward a node of interest; hence, SMs are self-routing applications.
CHAPTER 2. RELATED WORK

The SM platform is developed by directly modifying Sun Microsystems Kilobyte Virtual Machine (KVM); KVM features a small memory footprint (160K) suitable for most embedded devices. The tag space and SM operations are available to applications as a Java API.

Older implementation of EZCab use SM platform installed on HP iPAQs running Linux. The iPAQs use Orinocos 802.11 cards for wireless communication, and each of them is connected to a Geko 201 GPS receiver.

A taxi service based on real-time GPS information collected by a centralized dispatching center over cellular network has been implemented in Singapore (11).
Chapter 3

EZCab General Architecture

EZCab application is designed and implemented for ad-hoc network computers of embedded devices in taxis and client handled devices, which communicate between them using wireless interfaces as IEEE 802.11 or Bluetooth. This ad-hoc network has two types of nodes: permanently ones, consisting of cars from more taxi companies (if we presume that taxi companies activate in one big area city network) or cars from a single taxi company (if we consider that each taxi company has a network) and temporary nodes, represented by clients (a client will be part of the network beginning with a message request injection until finding a cab and finishing the three handshake protocol or until communication is interrupted by client or caused by a network timeout).

Client will book a nearby available taxi using direct communication with taxis instead of calling a central dispatcher and wait for a response. This decentralized architecture is cheap, simple and scalable for real-life problem, but also has several issues: we don’t know if at least a nearby available cab is found and if it is, we cannot estimate the real time of arriving for a cab to his client.

Also, EZCab presents new technical and computational challenges which do not exist in traditional systems: distributed protocol for finding at least one available taxi for customer, ensuring that available taxis accept one client request at one moment of time and ensuring that both taxi and client positions are well known for network entities. Finally, we must ensure the network security, providing a mechanism for clients and taxis to authenticate each other when they change messages.

The architecture of our EZCab implementation is based on EZCab implemen-
CHAPTER 3. EZCAB GENERAL ARCHITECTURE

... bringing new ideas like different EZCab nodes architecture, simple communication messages, simple communication protocol, EZ-Cab routing algorithm for preventing lost of request messages and to minimize the waiting time for an available taxi or solutions for ensuring the network security.

3.1 Peer-to-peer based Architecture

EZCab application is based on two types of entities: clients and taxis. Every cab includes an embedded computer device and forms with all other cabs an ad-hoc wireless network, so that every taxi can change information with all other network entities. Customer has to join this network to make a request for an available taxi. In this case, client is part of the network and must have a PDA or mobile phone to communicate with all other entities. If driver stations are homogeneous, client stations have different capabilities. Based on these types of clients and taxis, we have the peer-to-peer communication model: client communicates with cabs directly, and taxis forward the request messages received until an available taxi is discovered. Also, response message for client is forwarded in the network until the message is received by client.

We note that message can reach the destination on multiple channels. For example, the same response message is received by client from more intermediary cabs. But client will retain the first received copy of the message. All other copies received later will be dropped. Taxis will have the same behavior when receiving a message from network.

Diagram with general peer-to-peer EZCab architecture is presented in figure 3.1.

There are three types of messages in peer-to-peer communication model. First type is a request message composed by client and inserted into taxi network. Message is forwarded until is received by an available taxi. From this moment, we have a second type of message: the taxi response with a purpose for the client. Message is forwarded backward to the client. When is received, if client accepts the offer of the available taxi, compose a third type of message with acknowledge, which is sent to the available taxi.
3.2 Gateway based Architecture

Another solution is based on gateway stations. In this case, there are three types of entities in EZCab network: clients, taxis and gateways. A gateway is an intermediary node between clients and taxis. To find a cab, client needs to connect to a gateway that forwards the client request in the network. Also, gateway receives the responses from available cabs and sends the answer back to the client. Customer will send the confirmation message for the cab via gateway.

Gateway is responsible to filter answers (retain the first answer and drop all other copies of the same message received later). More, gateways can choose the optimal response from all taxi offers received (based on criteria like time of arriving or distance between client and cab) and send to the client a single solution.

This type of architecture is a better choice if we want the network to be monitored: to know how many taxis are registered in the network, how many clients have been registered so far, how many requests have received at least one answer or if client repeated the request. Also, there is an advantage in terms of security: customer real identity is assured by the telephone number sent by client in the request message or by his security certificate, and is know for sure which cab is assigned for every client.

On the other hand, if gateway is responsible to choose the optimal answer, client no longer has the freedom to choose a cab, choice based on personal criteria. Client will only receive one answer to his request, purpose that can be confirmed or not.
In figure 3.2 is presented Gateway based EZCab Architecture.

Like in previous communication model, there are the same three types of messages. But the request message is inserted into network by gateway. Also, gateway receives responses and sends confirmations. In addition, there is a fourth type of message necessary when client connects to the gateway for sending his information. Same type of message is used when gateway sends the final answer with the offer. Client can confirm or not, using the same type of message to send the confirmation to the gateway.

Gateway based architecture is a good choice if we want to have good knowledge of the ad-hoc network states: how many taxis are, how many clients are, how many requests are made, how many clients found an available taxi. In this case, gateways function like a certificate factory. To be a part of the network, every entity must first have a certificate from the gateway. Clients, when send a message request to the gateway, they automatically are included into the network and have generated a certificate.

Communication between gateway and taxis is the same like in the peer-to-peer model.
3.3 Emission nodes based Architecture

3.3.1 Fixed emission nodes

A third type of architecture is based on emission nodes. There are three types of entities: clients, taxis and emission nodes.

Emission node based Architecture is presented in figure 3.3.

Clients have the same behavior like in the first type of architecture. They will inject a request message into the network and will wait for answers. Emission nodes are special fixed entities, responsible for a single large area of the city. These emission nodes forward between them request messages from clients or response messages from available cabs. Each node will send the request message to all cabs in its area. If a taxi is available, then will send back to the emission node a response message. If not, taxi will not send anything. Emission point collects all responses and forward to the other emission points until the response reaches the destination client. In this type of architecture, cab will not forward messages. Cabs only will receive requests messages and will respond, if necessary.

On the other hand, taxis are mobile entities and pass to several areas. Each area has a responsible emission node. So, a taxi can receive a request from an emission node, pass in a new area and send the response to other emission node. That is not a problem, because emission nodes collect messages whether the cab message is a response to the node request or not. Client confirmations will be forwarded in the same way to the waiting taxi.

Architecture is a better choice because of a smaller number of messages forwarded in the network. On the other hand, it is difficult to provide an infrastructure for this type of communication.

Figure 3.4 shows the differences between number of messages forwarded in peer-to-peer based and emission point based architectures.

3.3.2 Dynamic emission nodes

We have considered that emission nodes are static entities. Another idea is that some taxis have the role of emission nodes. Emission node behavior will be the same like in peer-to-peer architecture, only that some cars will be able to forward
CHAPTER 3. EZCAB GENERAL ARCHITECTURE

Figure 3.3: Emission points based EZCab Architecture

Figure 3.4: Differences in forwarding messages
messages. All other ones will have the same role like taxis from classical emission node based architecture.

Also, emission node cars have the same behavior like a peer-to-peer architecture cab, meaning that it can make offers to clients.

The main advantage is that we don’t need an infrastructure for this type of architecture. The number of messages forwarded is smaller than the number of messages from peer-to-peer architecture.

A first problem is the possibility that more emission node cars be grouped. A good example is congested traffic in intersections. Emission nodes will generally forward same messages between them several times. Without any condition for sending and receiving messages a deadlock is caused.

A simple solution is to force emission node cars to have conditions when forward messages. A message received in more than one copy will be forwarded only once. Another solution is choosing randomly emission nodes from cabs. That means every cab is a classical peer-to-peer architecture taxi and can have emission node capabilities. A specific number of cars are chosen to be emission nodes a period of time, choose based on distance between taxis from network.

Another problem is that emission node cars have an advantage from all other cabs when receiving requests. Emission nodes have the busy state most of the time and have priority for processing requests, a not fair-play behavior for a cab in a taxi network.

Also, in this case, the offer received by client is not the best solution all time. Sometimes a cab is closer to the client than emission node, but dynamic emission nodes have priority in request processing, so emission node will send the offer to the client (assuming that both cabs are available).

Like in first communication model, there are three types of communication messages. First type is a request message composed by client and inserted into ad-hoc network. This type is forwarded by emission nodes and sent to the cabs until is received by an available taxi. Here, cab sends a response message to an emission node, which is forwarded until the destination client is reached. If client accepts the offer, a third type of message is composed, and forwarded like a request message to the waiting taxi. Once cab receives the confirmation, it will no longer accept any requests.
3.4 EZCab Routing Protocol

3.4.1 Peer-to-peer Architecture Protocol

The EZCab based on peer-to-peer communication consists in two phases:

- Client and taxis communicate and collaborate to forward messages throw the network to find an available taxi.

- Once an available taxi is discovered for a client, the two entities validate the result (cab sends a response message which shows that is available; client receives this message and decide whether to accept the taxi offer or not. In the first case, client sends the confirmation message to the waiting taxi).

EZCab needs to provide a mechanism for entities to authenticate each other. First, customer enters the ad-hoc network and is authenticated. From this moment, client is an entity of the ad-hoc network. Client sends a message with a request for an available cab into the network, with his location and time when message was sent. The message is routed in the network by the routing layer between cabs until an available taxi receives it. The taxi has a status that changes into a waiting state when accepts the request. Each taxi will know its location (longitude, latitude), so that when receives a message request, can compute the distance between him and the client and send a new type of message with response to the customer.

After a taxi is found, begins the second part of protocol: the sending of arrival time from taxi and confirmation message from client. If the available cab agrees with client requirements, the cab status is changed into waiting state and sends a message to the requester, with time of his arrival. Once the client receives a response message, has the freedom to choose to accept the taxi time offer or not. In first case, a message with a confirmation string will be sent to the taxi. Otherwise, customer does nothing.

Client, when sends a request message, has an intern time counter. If it doesn’t receive an answer in a given interval of time, a message request will be sent again to the ad-hoc network. Also, is possible that a message be processed by a taxi, after the client sent a confirmation message to another taxi, so that request normally expired. To avoid this situation, taxi will analyze only those requests that have the difference time between receiving time and request one less than a constant value,
which is the same with timeout repeating request value of the client.

Both the client and driver stations have a timeout mechanism which allows them to cope with highly dynamic network configurations. If the driver station *waiting state* times out before receiving the last part of the handshake protocol, the cab will be made available again. When cab sent a response message to the client and still expects the client confirmation message, in the time interval when the response is still available, cab receives other requests, but will not process them. These requests will be saved in a special queue, based on some criteria like time when message was sent or distance between cab and client. The queue will contain only request messages that aren’t expired. Also, the queue will be updated on a time interval upon these criteria. The queue messages will be ordered based on:

- time until the message expires and
- distance between cab and client.

If response message is not received before timeout, taxi will have again the *available status* set and will process first requests messages from expecting queue.

Client and taxi applications will not process more than once the same message: each entity will have a *processed messages* queue, with all messages processed in a last period of time. If received message is not in this queue, it will be processed and after, added to the queue. Otherwise, it will be dropped.

Cabs will have priorities, for example, their distance to the client, how long they have been idle or the number of clients served in last hour. The goal of such priority system is to guarantee fairness and optimality.

Also, client is allowed to make a single request in a period of time. After inserting a request message into the network, he waits a period of time for responses and after this period, if he has not received any response, he can make another request. Request can be insert again into the network automatically after a timeout or client can decide whether to make a new request or not.

### 3.4.2 Gateway and emission nodes based Architecture Protocol

For gateway based and emission nodes based architectures, EZCab protocol is the same, only that messages are different forwarded. In gateway based architecture,
client sends the request to a gateway which injects the message into the network. Also, response message is received by the gateway and forwarded to client. The protocol basically is the same, only that clients communicate only with a gateway and gateways replace customers in peer-to-peer EZCab protocol.

In the third type of architecture emission nodes act like cabs from the first type of architecture, only that they are fixed, and cars only respond to the request message, if is necessary. Emission nodes forward between them messages, send requests and confirmations to taxis on their area and collect message responses while taxis only send answers. Client communicates only with emission points, but the EZCab protocol is the same.

3.4.3 Entities behavior

In peer-to-peer architecture communication between the two types of entities (clients and taxis) is made by message transfer, using three types of messages:

- message request from client
- message response sent by an available taxi to the requester
- confirmation message from client.

Messages routed in ad-hoc network will not contain any code section; only data. Taxi station and client station have an EZCab implementation which process data received from messages. This is different by Smart Messages, which have included code and data, and used entity’s capabilities to run. So, we purpose a classical communication, where client station and cab station are static (both have implemented a program that executes when a message is received or arrival queue is not empty; otherwise have an available state), and message transfer is the dynamic part.

Taxi node behavior

Taxi can have four states in EZCab Peer-to-peer Routing Protocol:

- **Available State** - If arrival queue is empty, cab expects to receive any request message.
- **Compute State** - Once taxi has received a request (or as much as long request queue is not empty), taxi take data from every message in order from queue(described in protocol section), complying with security rules (entities in
the same network etc.), compute data received and forwards the message (busy state, waiting state) or sends an offer message to client (waiting state).

- **Waiting State** - Taxi sends a response message to the client, his state is waiting state, waiting a confirmation message but also receives other request messages, puts them on a request queue, sort by criteria like distance between taxi and client or time when message expires. Once confirmation received, taxi goes to busy state; otherwise, if time of waiting exceeds a timeout constant, taxi goes into available state.

- **Busy State** - Taxi has busy state and will no longer compute any request message. Instead, it will receive request messages and will forward them to other taxis.

**Client node behavior**

Customer can have three states in EZCab Peer-to-peer Routing Protocol:

- **Request State** - Client node enters into ad-hoc network and sends a request message.

- **Waiting State** - Client node is waiting for response message from an available taxi.

- **Confirmation State** - Client received a proposal message from a cab. If client does not find in the message a convenient proposal he will be able to make
another request\,(request state)\,or he can close the communication. Otherwise he sends a confirmation message for the available taxi. In both cases client will drop all other response messages. After the confirmation message is sent, client will no longer wait for a response message (end of communication).

In the other two types of architecture, behavior of clients and taxis are the same. Gateways and emission nodes doesn't change with anything the sequence of states for these two types of entities.

### 3.4.4 Short EZCab Booking Protocol

A second type of protocol is similar to the first one, only that the third part, the client confirmation, is removed. Client injects a request and at least one cab gives a response. But response message contains this time a phone number of the taxi instead of his id number. Client, if accepts the offer, will no longer send a confirmation message. Instead, client will take a call to the taxi driver. From here, booking cab problem is based on direct human communication. The short cab booking protocol is the first part of EZCab protocol.

This is a very good alternative for the EZCab protocol because the security and the integrity problems of the ad-hoc network are resolved. Direct communication between entities ensures that both client and taxi are real. A good example of utility of this protocol is given in security section.
CHAPTER 3. EZCAB GENERAL ARCHITECTURE

3.5 EZCab Communication Messages

The main difference between the old EZCab prototype implementation and the new one are communication messages between entities. If the old implementation is based on Smart Messages, which contains executive units, we purpose to use simple data messages, only with useful required information. Every entity in the network receives messages and process them using EZCab application installed.

Figure 3.7 shows the flow of messages between client and cab nodes.

Based on EZCab peer-to-peer protocol stages, there are three types of EZCab communication messages:

1. Request message that is introduced by client in ad-hoc network. This message will contain the following fields:
   - *Client certificate* - a client certificate signed by client to ensure the security of the network.
   - *Network identifier* - the identifier of the network in which messages are routed; an entity that receives a message from a different network will drop the message. If client does not have a preference on choosing a network this field will be set to a default value and taxis will no longer filter the message by network identifier.
• **Client location** - a data structure that provides coordinates of the client location.

• **Sending time** - value that helps to know when client request expires. An expired request is no longer processed or sent forward.

• **Number of hops** - field that shows how many hops request message can be computed by taxi entities. Field is initialized from client and decremented until value is 0. After that message will be dropped.

Each taxi station will know its location (latitude, longitude) so when receives a message request, can compute the distance between its location and client location.

(2) When an available taxi receives a request message, sends another type of message to the customer with time of his arrival. The message contain the following fields:

• **Taxi certificate** - the taxi station certificate that ensures the security of the network.

• **Client certificate** - the certificate of the destination entity.

• **Network identifier** - field that has the same utility as in the previous type of message.

• **Sending time** - value that helps to know when client request expires. A request expired is no longer processed or sent forward.

• **Number of hops** - value that shows how many hops response message can be forwarded by taxi entities. Field is initialized from taxi station with a maximum hops constant. Once arrived to a taxi station will be decremented until value is 0. After that message will be dropped.

• **Response string** - a string with auxiliar information.

When a response message is sent a time counter is set. If in this time interval [0, timeout constant] waiting taxi will not receive a response message from client, it will be again available to process other requests.

(3) Once the client receives a response message from an available taxi, has the freedom to accept the taxi offer or not. In first case a third type of message will be sent to the waiting taxi. This message will contain:
- **Client certificate** - the client certificate that ensures the security of the network.

- **Taxi certificate** - the certificate of the destination entity.

- **Network identifier** - field that has the same utility as in the previous type of message.

- **Sending time** - value that helps to know when client request expires. A request expired is no longer processed or sent forward.

- **Number of hops** - value that shows how many hops response message can be forwarded by taxi entities. Field is initialized from taxi station decremented until value is 0. After that message will be dropped.

- **Confirmation string** - auxiliar information.

In real scenarios, due to economical reasons, there is a small probability that taxi companies collaborate to find each customers. Each company is not willing to route messages for a competing company. For this reason, we can have a new field for each type of communication message, containing the identifier of the taxi company in which client makes the request. In this way, there are more subnetworks that forms the ad-hoc network. When receives a message from a different company, a cab can drop it or can send it further, ignoring the content. A client can choose the taxi company to use, simply setting a header value of his request message. His EZCab application running on PDA or cell phone will have this option.
3.6 EZCab Routing Algorithm

Because we don’t use Smart Messages like in the previous idea of implementation, messages will be routed by an algorithm based on flooding. Being designed by end-to-end data transfers, EZCab algorithm has two principal characteristics:

- limited geographical area
- multiple possible destinations.

Each client request is broadcasted to all neighbors recursively, up to a maximum number of hops or until it arrives to an available cab node. Client will consider only the first available cab response. All other responses will be dropped. The maximum number of hops is constant, which shows the visibility of a request message.

When a taxi receives a request and his state is available, it will send an offer message to client entity. Algorithm is the same, based on flooding. Only that we now know the destination which is the client node who made the request.

In the third part of the handshake, customer send an acknowledge message to the taxi who sent the response message with the proposal. Client flooding algorithm is the same as cab flooding (again the destination is known).

Every type of message will be forwarded a given number of hops. After that, it will be dropped. Also, to prevent the forwarding of the same message many times by an entity in the network, every taxi will have a queue where to save information about the last messages forwarded. When a taxi receives a message that must be sent forward, first it will look into the last messages forwarded queue. If information about the message is already saved in queue, it will be dropped. Otherwise, taxi will save information about the message and will forward it.

If the network is very dense, flooding algorithm does not work well due to the unavoidable wireless contention. In such a situation, the third type of architecture, based on emission points is a better choice: messages are forwarded only between these emissions points, which are less than cabs, so, the number of messages forwarded in the ad-hoc network is smaller than the number of messages routed in a classical peer-to-peer architecture.
3.7 EZCab Security

Current solution for booking cabs implies that client provides their phone number and often, their real identity is required. Some EZCab architectures provide authentication mechanisms where customer identity remains anonymous, while others assure a secure communication between client and available taxi that serves customer using phone number exchange between them.

Basically, in EZCab architectures that we purposed, the integrity of communication is based on certificates. Each entity from network has certificates: cabs have by default, being all time a part of ad-hoc network and clients receive such a certificate when make a request. After a taxi is provided, client is not part of network anymore and his certificate is no longer valid. In this way client remains anonymous but in the same time the integrity of network is assured.

All type of messages forwarded into network have a field containing this certificate. Messages will be processed only if senders are recognized as part of the network. This helps that a client be served by a single cab at a moment of time.

In peer-to-peer and emission node based architectures clients and taxis will not be listed in a database. In gateway based architecture, gateways can have a database consisting of client and cab information. This information can include entity certificates and phone numbers. Information from database is confidential, but may be disclosed, if necessary.

This security solution is very useful in current urban life. For example, a client wants to send his child to school by taxi. As a parent, he will be more confident if he will have the possibility to know the identity or the phone number of the driver. The existence of database resolves this problem.

A better solution is when client and cab send to each other their phone numbers, instead of string messages for response and confirmation of response. The phone numbers will be encrypted, so only those two entities will be able to know them, decrypting the messages using keys. This solution is based on public and private keys, and requires another protocol phase, the public keys interchange.

Another idea for this solution, based on number phone interchanging, is to eliminate the confirmation phase of the EZCab booking protocol. Client will send a
request containing also his number phone and will receive a response message with
cab number phone. From this moment, communication ends. Client will call driver
to discuss the taxi arrival details.
Chapter 4

EZCab Implementation Details

4.1 VNSim

VNSim traffic simulator is a project developed by University Politehnica of Bucharest. VNSim is designed to be a realistic simulator for VANET technologies, ranging from wireless networking protocols to applications developed over VANETs. VNSim is implemented in Java and is based on the collaboration between drivers and the exchange of information between cars equipped with GPS devices or with short-range wireless communication capabilities devices. VNSim is based on discrete events. There are two main types of events implemented: send and receive. In each moment of simulation time, events are pulled from the event queue and processed by simulator. Events work with special objects, in our case, objects modeled by real car attributes.

VNSim Architecture is presented in figure 4.1.

Next, we will present some components of VNSim on which EZCab implementation is based like maps component, traffic simulator, scenario generation and DSRC protocol implementation.

4.1.1 Maps

VNSim works with road models. Each road has a category information (small street, local road, state route, interstate route and so on). Also VNSim uses TIGER files. A TIGER file specifies the entry points and the exit points for every road, along with as many intermediary points is needed. For example, a straight road does not have intermediary points, only entry and exit points, while a curved road
needs a large number of intermediary points. TIGER files are available in two formats:

- Record Type 1 (RT1), format that contains the entire road segments for map regions, with useful information like name, direction, starting points or end points.

- Record Type 2 (RT2), format that contains the intermediary points for curved roads. A TIGER file contains also general traffic-specific information like number of lanes and traffic control systems.

4.1.2 Traffic Simulator

VNSim traffic simulator is based on driver behavior model (6). The idea of this model is that driver can be in one of four states: free diving, approaching, following or braking.

*Free diving mode* means that car is not influenced by the preceding vehicles from the same road, so that driver will seek to obtain or maintain the desired speed.

*Approaching mode* means that a slower car is in front of currently car. In this case, driver must decelerate to obtain the same speed with the preceding car.
Following mode means that the preceding vehicle has the same speed with currently car. That means that driver must maintain the current speed.

The last mode, braking is similar with approaching, only that the preceding car is very close in front of the current car. In this case, driver must apply high deceleration rates.

VNSim provide also a lane changing model, for multi-lane roads, based on lane usage rules valid throughout most part of Europe, which incorporates a traffic control system.

4.1.3 Scenario Generation

Road information is stored in a special java object, called Map. To create a traffic scenario, user must load an existing Map and add to this object entry points and exit points. This information is saved in a special java object, called Scenario. Both Map and Scenario object are serialized in .smf files, for future use. After, user can load the newly created object and add traffic information on the map, specify the flows of vehicles or specify the routes to follow between entry points and exit points. This last object is serialized in a .fsc file, which contains all information necessary to load a specific scenario.

In the GUI interface, the container that manages all configuration components is Scenario Configuration. It contains a set of buttons for map control, for canceling or saving the current configuration, buttons that set the configuration mode and an Entry Exit Configuration which is responsible with displaying all configuration parameters corresponding to the currently selected entry and with displaying the map. Also, this container has a list of all existing entries. The Entry Exit Configuration is responsible with displaying the map.

A scenario is build using the following information: the name of the map used, a list of entry points, a list of exit points, a list of routes and a list of driver types. Each route is described by an entry point, an exit point and a list of segments that form the route. The designer produces as an output a Scenario object.
4.1.4 VNSim Models

VNSim traffic simulator is built by a series of modules. Information about all roads is contained by Map module (figure 4.2).

![Map Module Diagram]

The principal component of VNSim, the *moving car* part is developed in Move() function of the CarInstance class. This method computes a new position for every vehicle by simulating its motion during a small time frame (10 ms). The method is called by the simulator engine every frame, for each vehicle.

Car movement depends on traffic control system, the driver personality and positions of the other neighbor cars. A vehicle first checks the traffic control system present in an intersection (this is done when vehicle changes the road or when arrives to an intersection). Vehicle takes a decision according to the type of intersection. A second element that influences car movement is the position of adjacent vehicles. Car establishes its driving mode after all adjacent car positions are checked. The chosen driving mode of the car was discussed in a previous section. Also, car state is influenced by its driver personality. In this case The Personality class encapsulates a large number of parameters. Different personalities can be easily modeled, by
4.1.5 DSRC Protocol Implementation

EZCab implementation in VNSim traffic simulator is based on DSRC protocol. Dedicated Short-Range Communications Protocol is a multi-channel wireless protocol, still under development, that is based on the IEEE 802.11a Physical Layer and the IEEE 802.11 MAC Layer. It operates over a 75 MHz licensed spectrum in the 5.9 GHz band allocated by the FCC for the support of low latency vehicle-to-vehicle and vehicle-to-infrastructure communications.

![DSRC Protocol Implementation Diagram](image)

Figure 4.3: DSRC Protocol Implementation

DSRC protocol is implemented under VNSim using traffic simulator components (Figure 4.3). Are used the same objects for map, scenario, vehicles and events models. The DSRC OSI Stack communicates with old elements in the simulator: Physical Layer communicates with the send routine which gets the packet, creates a new Send Event and adds it to the Event Queue and with the receive routine which gets the Receive Event from Event Queue and sends the message to the Physical Layer. MAC Layer sends and receives from/to Physical Layer and Appli-
CHAPTER 4. EZCAB IMPLEMENTATION DETAILS

4.2 EZCab Implementation

EZCab is an extension of VNSim application, developed in Java programming language. Java was chosen due to its portability to existing operating systems. System architecture is based on a modular structure, this simplifying application testing and implementation. We choose to implement EZCab peer-to-peer based architecture.

The architecture of the system is presented in figure 4.5. The design is modular, multi-layered and therefore it is scalable and easily extensible.

When starting VNSim application, after we generate a map and a scenario, we can choose to run the simulator with EZCab module or not. If EZCab option is selected (figure 4.4), EZCab entities will be generated and will be applied EZCab booking protocol.

![Figure 4.4: VNSim User Interface](image-url)
4.2.1 EZCab Entities and EZCab Communication Protocol

In EZCab implementation exist only two types of entities: taxis and clients. Both cab and client have a similar implementation. EzcabCar implements taxi car behavior and EzcabClient implements customer behavior. Both objects extend CarRunningDSRC, a simulated car implementation that forwards DSRC messages into VANETs which extend CarRunningVITP. But root class for EzcabCar and EzcabClient is SimulatedCarInfo class, which is the car object from VNSim, with real car attributes.

Figure 4.5: EZCab Implementation

Cabs and clients are nodes of ad-hoc network: taxis are permanently nodes and customers are temporary ones. As we said in the previous section, the core of VNSim simulator is the Engine class. This engine keeps track of all the nodes and takes care of delivering messages among them. The simulator is a single-threaded, discrete, event-based simulator. In EZCab, nodes are modeled by EzcabCar or EzcabClient classes, both with the same root class, SimulatedCarInfo. Both classes are similar, client being implemented like a car with a very slow speed. We choose to implement client like a special car because in real world, clients inject requests and receive answers using PDAs or cell phones similar with cabs devices. Taxi and
client have the same communication capabilities.

EzcabCar and EzcabClient have two important methods: \textit{onReceive} and \textit{prepareMessage}. \textit{onReceive} method is called by the simulator engine, when the node has received a message. \textit{prepareMessage} method is called by the simulator engine, when the node intends to send a message. Messages, at this level, are seen as simple byte sequences.

Both objects, EzcabCar and EzcabClient have a field that shows their type of entity: \textit{taxi} or \textit{client} and a field showing entity state: a cab can have \textit{available}, \textit{waiting} or \textit{busy} state and client can have only waiting or busy state. Client and cab objects are generated in \textit{addCar} method from Engine class. On each discrete moment, if type of application is EZCab, then a taxi or a client is created, depending on the restrictions. Client is different by a car when is created only by speed: he will have only 1km/h. All generated cabs will initially have an available state. After receiving and forwarding request messages from clients, its status will change into waiting state and after, busy state, depending on confirmations sent by clients or on ad-hoc network state.

As we said, when a cab is created, has the \textit{available} state and waits requests from clients. Client will be created from first having the \textit{waiting} state, because on the moment of his generation, he injects into the network a request message. After he receives a response and he agrees with, he sends a confirmation message and his state changes into \textit{busy} one.

When receiving a message, depending on its header, clients and cabs will process its body. In our EZCab implementation, messages are processed only if they have the EZCab byte header. Otherwise, \textit{onReceive} function of the superclass is called. Based on current received message processing, the state of entity is updated and a new message is created. This new message is scheduled right after its creation or when VNSim engine calls the \textit{prepareMessage} function from entity class. Messages are filled in using send events. Also, messages are received from VNSim engine which schedules receive events.

Both entities are implemented according to peer-to-peer EZCab protocol phases and restrictions. Also, clients will process only EZCab messages containing re-
sponses from taxi cars. Cab object has two processing queues. One with messages processed previously (in last twenty minutes for example), used to verify no more than one message copy received. This extension is not used in our EZCab simulations. A second queue is used to keep requests when cab has the waiting state. Messages from this queue are sorted on some criteria like time until the request expires or distance between cab and client who made the request in order to choose another client to serve, if confirmation from previous client is not received in a period of time. EZCab Client object ends his activity when sends a confirmation message. Cab objects, being permanently entities, will finish the activity when at their exit from the map. From that moment, cabs wont forward messages anymore.

There are three distinct message flows: the first of them starts when a client inserts a request message into the network. The message is routed until is processed by an available cab or until expires. The second flow starts when available taxi sends an offer to the client, scheduling another send event containing a response message. The last flow consists on routing of confirming message sent by client.

Peer passes the message to the routing module. Messages sent by the peer contain application-level information, intended to be understood by the peer modules on other nodes. The routing module encapsulates the EZCab messages in routing packets and prepares them for sending. When the simulator engine calls the prepareMessage method, the corresponding sequence of bytes is returned, and the simulator engine will take care of sending it, according to the wireless transmission simulator module. A second step starts when the simulator engine delivers a message to a node. Thus, it calls the onReceive method, passing it the corresponding sequence of bytes. It is then decoded by the routing module, and the EZCab message is sent to the peer, which deals with it accordingly.

4.2.2 EZCab Communication Messages

The classes which model the different types of messages exchanged between the various components are presented in the UML class diagram in figure 4.6.

EZCab messages are created and understood by the peer modules. There is a common EZCab message, extended by principal types of EZCab messages: request, response and confirm. Every EZCab message is characterized by two fields: type
In our implementation for VNSim simulator, a request message must include the identifier of the sender, the number of hops that message passed (initialized with a maximum value permitted) and the time of sending (necessary to know when request expires). An EZCab response will have extra information, including the sender cab, the destination peer and a hops value. The last type of message includes also identifiers of client and available cab.

When a message is processed and must be forwarded, a new message is created, having the same information, only that number of hops is decremented. Information will be routed as long this number is greater than zero.

EZCab messages are encapsulated by the routing modules in routing packets, which contain additional routing information.

In our EZCab implementation, we followed that a message contains as few information as we can, without modify the architecture: messages routed in simulated EZCab application have at most 23 octets.
In simulated EZCab implementation, we assumed that each taxi company is willing to route messages for a competing company. So, EZCab messages will not have the special field for company identifier.
Chapter 5

Simulations and Results

5.1 Simulation Scenarios

All scenarios have been tested on a computer with Intel Dual Core 2GHz processor, 2 GB RAM memory, using Windows Vista and Ubuntu 9.04 Operating Systems.

For EZCab module testing, we used three test scenarios: a simple test scenario, used also in our implementation of EZCab application, a more complex one, with many intersections, each of them equipped with traffic lights, and a third test scenario that models an urban jam, with lot of streets and intersections. Last two scenarios are used for application evaluation and are real world models.

First test scenario (figure 5.1) is a simple intersection, with two streets, equipped with four traffic lights, every street having two lanes. We used this scenario not only to help in our application implementation, but also to test the time during an available car is found for a client. Cars are generated near this intersection, where is an urban jam. Basically, cars will change messages between them when they are waiting on a traffic light. In this case, we measured the success rate for requests messages, for different shares between clients and taxis.

The rate for entity generation for this scenario is 100 entities/ hour/lane. We have a group of four tests, with different probabilities for generating client entities: 10%, 33%, 50% and 66% from total entities. EZCab ad-hoc network is analyzed for 10 minutes (time of server) or 1 hour and 30 minutes (time of simulation). We used intersection scenario with adaptive traffic lights, predefined routes, and a probabilistic forward dissemination model and using DSRC communication protocol.
Simulation results contain the number of requests injected into the network, number of responses from available cabs and time during an available taxi is found for a client, for different probabilistic shares between clients and taxis from ad-hoc network.

The second scenario is a real world map: Politehnica University Campus (5.2). Each street has two lanes and on every intersection we have traffic lights. We assumed that traffic is a congested one on this map, this scenario offering more complex situations to test EZCab application behavior: how many requests are injected into network, how many clients had received an answer and time during each client find a taxi.

In this scenario, entities are generated all over the map, and taxis can change messages between them not only on intersections.

Rate for entity generation for this scenario is the same like in intersection scenario: 100 entities/hour/lane. We made a group of four tests, with same probabilities for generating clients from all entities: 10%, 33%, 50% and 66%. These different shares for client and cabs make a difference between networks with few client requests (for example, in the middle of the day when common transport cars are available) and networks where are more clients than cabs (for example, 1 a.m.
on exit from cinemas, where a lot of possible clients make requests). EZCab ad-hoc network is analyzed for 10 minutes (time of server) or 1 hour (time of simulation).

We test scenario having fixed traffic lights, predefined routes, probabilistic forward dissemination model and using DSRC communication protocol.

The third scenario is also a real life scenario: the downtown of a big city, in our case the center of Manhattan, New York (figure 5.3). We choose this model because of intense traffic, of multitude of routes, intersections and traffic lights and because of a great number of entities. On this scenario we had more complex tests, modifying not only probabilistic shares between clients and taxis, but also the number of vehicles per lane in one hour.

We have two groups of simulation tests, each one having four tests, one for each probability, like in previous scenarios. Each group has a different number of entities generated per lane in one hour: 200 and 800 entities. The number of entities generated is greater than the number from other two scenarios. Also, because of the great number of events that must be processed, we tested EZCab application for 2 hours and 30 minutes (time of server), which means 5 minutes in simulation time.
Besides the other situations tested in previously scenarios, we vary the number of cars per lane in one hour to test the EZCab behavior on an urban jam context.

Figure 5.3: Downtown Scenario

In all scenarios, we assumed that client accepts first message offer received from network, and drop all others received responses.
CHAPTER 5. SIMULATIONS AND RESULTS

5.2 Results

5.2.1 Intersection Scenario

For intersection scenario we have four tests, with different probabilities to generate clients: 10%, 33%, 50%, 66% from all entities generated, the difference number being taxi cars. We test scenario with 100 entities/hour/lane for 1 hour and 30 minutes simulation time.

In following diagrams is shown the booking time for an available taxi for different clients at different moments of simulation time (time to complete succession of all stages for booking protocol) in all four test scenarios (figures 5.4, 5.5, 5.6, 5.7).

![Booking time for clients generation probability of 10%](image)

Figure 5.4: Booking time for clients generation probability of 10%

Booking time is measured in milliseconds (X axis) on different values of current time simulation (series axis). In following diagrams, current time simulation is represented in logarithmic scale.

We notice that for 10% probability to generate clients from all entities, cabs are booked for clients in a very short time, few clients waiting more than 2-3 seconds (figure 5.4). Request messages inserted into network are few so a good part of clients receives responses (table 5.2).

As the number of clients increases compared with the number of taxis from the network, time for booking protocol also increases (figures 5.4, 5.5, 5.6, 5.7).
CHAPTER 5. SIMULATIONS AND RESULTS

Figure 5.5: Booking time for clients generation probability of 33%

Figure 5.6: Booking time for clients generation probability of 50%
If for 10% probability test, we can see from diagram that booking time is no more than 2-3 seconds, for 33%, 50% and 66% probability tests booking time is increasingly more than 2-3 seconds.

Also, the average booking time increases with the number of clients from the network (figure 5.8). We notice that in all scenarios, client will book for a cab no more than few tens of seconds.

All values have been measured in miliseconds.

What is interesting is that the maximum values for booking time in all four test scenarios are relatively close to each other.

Test scenario consists of a single intersection so messages are changed between entities when cabs are waiting on a traffic light. On the other hand, there is a strong possibility that cars out of the range map throw exit points, so cabs will not be a part of the ad-hoc network anymore. Cars pass in another area and distance
Figure 5.8: Differences between average time, weighted average time and maximum time values in Intersection Scenario

from all other vehicles makes impossible communication with them.

In this scenario, part of the cars will out of range map and will not communicate anymore with other nodes and a part of request messages won't be processed anymore (for example, a client receives a response message, sends a confirmation but waiting cab will not receive the client message because it got out of the range map and from this moment communication is closed). For this reason some requests are still waiting to be processed.

Figure 5.9: Rate of request messages still routed into the network for Intersection Scenario

In figure 5.9 we notice that the number of request messages that are still routed into the network increases with the number of clients (the share of customers in the network).

Results obtained are optimistic, if we compare booking time using EZCab with booking time using the classical solution (when calling to a dispatcher). Waiting time when calling a dispatcher is an average of 2-3 minutes, while time obtained
with EZCab is an average of few seconds (less than a minute).

Table 5.2: Client requests, responses and waiting requests for Intersection Scenarios

<table>
<thead>
<tr>
<th>Probability</th>
<th>Requests</th>
<th>Responses</th>
<th>Waiting requests</th>
<th>Rate of waiting requests</th>
</tr>
</thead>
<tbody>
<tr>
<td>10%</td>
<td>80</td>
<td>50</td>
<td>30</td>
<td>38%</td>
</tr>
<tr>
<td>33%</td>
<td>359</td>
<td>115</td>
<td>244</td>
<td>68%</td>
</tr>
<tr>
<td>50%</td>
<td>506</td>
<td>87</td>
<td>419</td>
<td>83%</td>
</tr>
<tr>
<td>66%</td>
<td>427</td>
<td>97</td>
<td>330</td>
<td>77%</td>
</tr>
</tbody>
</table>
CHAPTER 5. SIMULATIONS AND RESULTS

5.2.2 Politehnica University Campus Scenario

Intersection scenario is used to show how long a client waits to book an available taxi, when clients and taxis are close to each other. But in real scenarios, there are more streets, with a lot of intersections on a geographical area much larger. For this, we considered a second test scenario, a real world map improved with series of traffic lights for each intersection and considering that Politehnica University Campus Scenario area has a high traffic.

Clients are generated in all entry points of the map, not only close to intersections, like in previous scenario and request messages are forwarded in a larger area. Also, a smaller number of cars will out of range area so request messages have a greater probability to be routed on large distances, all booking protocol phases being reached.

Like in previous scenario, we considered four type of tests (10%, 33%, 50% and 66% probability that generated entities be clients) at 100 entities/hour/lane.

Booking times for different simulation time values in these four test scenarios are presented in figures 5.10, 5.11, 5.12 and 5.13.

Booking time and simulation time are expressed in milliseconds. In result charts, simulation time values are represented in logarithmic scale.

![Figure 5.10: Booking time for clients generation probability of 10%](image-url)
Figure 5.11: Booking time for clients generation probability of 33%

Figure 5.12: Booking time for clients generation probability of 50%
We notice that when number of clients is smaller comparing with number of available cabs from the ad-hoc network, clients have a fast response to their requests, difference between minimum and maximum booking time being very small (figure 5.10). Graphic is continuously; there are no booking time values that jump from a certain average.

Once with increasing of client number also increase values of booking time. Generally, the response time average is the same, only that are several situations where booking time is greater than this average, once the share of customers is larger comparing with number of available cabs.

Table 5.3: Averages and Maximum values for Politehnica University Campus Scenario results

<table>
<thead>
<tr>
<th>Probability</th>
<th>Average (ms)</th>
<th>Weighted Average (ms)</th>
<th>Maximum (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10%</td>
<td>278.33</td>
<td>275.85</td>
<td>1730</td>
</tr>
<tr>
<td>33%</td>
<td>6068.67</td>
<td>6936.11</td>
<td>93790</td>
</tr>
<tr>
<td>50%</td>
<td>3030.29</td>
<td>1641.38</td>
<td>87050</td>
</tr>
<tr>
<td>66%</td>
<td>9212.96</td>
<td>9385.85</td>
<td>72500</td>
</tr>
</tbody>
</table>

In table 5.3 all values have been measured in milliseconds.

Average booking time increases with number of request messages routed in EZ-Cab network. In this scenario, clients are generated in groups, in the same entry points.
First clients will have a fast response from network, but following customers will
expect increasingly more time to book a taxi, their requests being routed on increasingly distances in the network. We notice an increasing of booking time average (figure 5.14) and an increasing of number of waiting requests (figure 5.15) once with the increasing of clients rate. Increase rate is a linear function.

We also notice that waiting requests rate increases linearly with the increasing share of clients in ad-hoc network (figure 5.15).

In 10% test scenario maximum and average booking times are very small in comparison with maximum and average booking times from all other test scenarios. In 10% test scenario booking time values are less than one second. A good explanation for this values is that clients in this test scenario are very few comparative with available taxis. In Politehnica University Campus Scenario are more intersections and more entry points than in previous scenario, so locations where clients are generated are distributed on the map. So, requests from these few clients are processed in very little time.

Maximum and average booking times from other test scenarios have close values.

<table>
<thead>
<tr>
<th>Probability</th>
<th>Requests</th>
<th>Responses</th>
<th>Waiting requests</th>
<th>Rate of waiting requests</th>
</tr>
</thead>
<tbody>
<tr>
<td>10%</td>
<td>30</td>
<td>24</td>
<td>6</td>
<td>20%</td>
</tr>
<tr>
<td>33%</td>
<td>101</td>
<td>54</td>
<td>47</td>
<td>47%</td>
</tr>
<tr>
<td>50%</td>
<td>158</td>
<td>68</td>
<td>90</td>
<td>57%</td>
</tr>
<tr>
<td>66%</td>
<td>236</td>
<td>45</td>
<td>191</td>
<td>81%</td>
</tr>
</tbody>
</table>

Like in previous scenario, results obtained are optimistic, if we compare booking
time using EZCab with booking time using the classical solution (when calling to a dispatcher). Waiting time when calling a dispatcher is an average of 2-3 minutes, while time obtained with EZCab in this test scenario is an average of few seconds (less than a minute).

Figure 5.15: Rate of request messages still routed into the network for Intersection Scenario
5.2.3 Downtown Scenario

Downtown scenario is more complex than the second one, having more streets and more intersections. This scenario is also a real world map, representing the map of Manhattan, New-York.

If in first two scenarios, we test EZCab application having a constant number of entities generated per lane in one hour, varying the number of clients in comparison with number of taxis. In this scenario we study the behavior of EZCab application for different numbers of entities generated in one hour and different shares of clients.

For Downtown scenario, we make two groups of tests: one with 200 entities/hour/lane and the second with 800 entities/hour/lane. For each group we considered four type of tests (10%, 33%, 50% and 66% probability that generated entities be clients). We ran each test 2 hours and 30 minutes in server time, meaning 5 minutes in simulation time (in this scenario, due to the large number of entities generated per hour and map complexity, is a large amount of data that must be processed by application). Between these two groups of tests is a difference of 600 entities generated/hour/lane just to simulate traffic in two different periods of the day (for example 2 a.m. for first scenario and 6 p.m. for the second, when traffic is very congested).

Like in previous scenario, clients are generated in all entry points of the map, not only close to intersections, like in previous scenario and request messages are forwarded in a larger area. Also, a smaller number of cars will out of range area so request messages have a greater probability to be routed on large distances, all booking protocol phases being reached.

Booking times for different simulation time values in Downtown scenario with 200 entities generated/hour/lane are presented in figures 5.16, 5.17, 5.18, 5.19 and 1.13 and for 800 entities generated/hour/lane are presented in figures 5.20, 5.21, 5.22, 5.23.

Booking time and simulation time are expressed in milliseconds. In result charts, simulation time values are represented in logarithmic scale.

We notice that in first group of test scenarios, booking time functions tends to be linearized, time values being close, due to the number of entities generated per
Figure 5.16: Booking time for clients generation probability of 10% - 200 entities/hour/lane

Figure 5.17: Booking time for clients generation probability of 33% - 200 entities/hour/lane
Figure 5.18: Booking time for clients generation probability of 50% - 200 entities/hour/lane

Figure 5.19: Booking time for clients generation probability of 66% - 200 entities/hour/lane
Figure 5.20: Booking time for clients generation probability of 10% - 800 entities/hour/lane

Figure 5.21: Booking time for clients generation probability of 33% - 800 entities/hour/lane
CHAPTER 5. SIMULATIONS AND RESULTS

Figure 5.22: Booking time for clients generation probability of 50% - 800 entities/hour/lane

Figure 5.23: Booking time for clients generation probability of 66% - 800 entities/hour/lane
In the second group of tests, as in previous test scenarios, booking time functions tend to have a graphic similar with *saw teeth*. As number of requests increases, functions are more nonlinear.

In both groups of test scenarios, booking time have close minimum values and close maximum values.

In first group of tests results, time booking averages and maximum values for each scenario have increasing values as number of clients generated increases; except values from first scenario (*10% scenario*), where average values are high compared with those obtained in other test scenarios (figures 5.24, 5.25, 5.26).

<table>
<thead>
<tr>
<th>Probability</th>
<th>Average (ms)</th>
<th>Weighted Average (ms)</th>
<th>Maximum (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10%</td>
<td>5188.15</td>
<td>60092.95</td>
<td>91960</td>
</tr>
<tr>
<td>33%</td>
<td>4612.5</td>
<td>6336.53</td>
<td>54090</td>
</tr>
<tr>
<td>50%</td>
<td>16816.43</td>
<td>26023.73</td>
<td>90080</td>
</tr>
<tr>
<td>66%</td>
<td>27800.77</td>
<td>30708.32</td>
<td>144100</td>
</tr>
<tr>
<td>10%</td>
<td>7817.22</td>
<td>8972.12</td>
<td>72190</td>
</tr>
<tr>
<td>33%</td>
<td>17798</td>
<td>27285.67</td>
<td>162950</td>
</tr>
<tr>
<td>50%</td>
<td>19762.93</td>
<td>29709.3</td>
<td>148610</td>
</tr>
<tr>
<td>66%</td>
<td>10322.71</td>
<td>14330.1</td>
<td>67590</td>
</tr>
</tbody>
</table>

In second group of tests, with 800 entities generated per lane in one hour, time booking averages and maximum values have not an increasing rate once with increasing of clients. Average and maximum times tend to have similar values in each test scenario. That means that in very congested traffic situations, like in our group of tests, increasing of number of requests does not matter so much, response time having similar values (table 5.5).

Like in previous test results, we notice that the number of requests routed into the network increases with the number of clients increasing. The number of waiting requests from the network when are generated 200 entities per lane in one hour is less than the number of waiting requests for 800 entities generated per lane in one hour, for each probability to generate clients.

Also, we notice that in first group of test results, the number of clients that have booked a taxi is almost the same (an average of 13 customers) for different
Figure 5.24: Differences of average values for booking time in Downtown Scenario

Figure 5.25: Differences of weighted average values for booking time in Downtown Scenario
Figure 5.26: Differences of maximum values for booking time in Downtown Scenario

Figure 5.27: Differences in rate of request messages still routed into the network for Downtown Scenario
Table 5.6: Differences for client requests, responses and waiting requests rate in Downtown Scenario

<table>
<thead>
<tr>
<th>Probability</th>
<th>Requests</th>
<th>Responses</th>
<th>Waiting requests</th>
<th>Rate of waiting requests</th>
</tr>
</thead>
<tbody>
<tr>
<td>10%</td>
<td>14</td>
<td>13</td>
<td>1</td>
<td>7.14%</td>
</tr>
<tr>
<td>33%</td>
<td>28</td>
<td>12</td>
<td>16</td>
<td>57.14%</td>
</tr>
<tr>
<td>50%</td>
<td>30</td>
<td>14</td>
<td>16</td>
<td>53.33%</td>
</tr>
<tr>
<td>66%</td>
<td>78</td>
<td>13</td>
<td>65</td>
<td>83.33%</td>
</tr>
<tr>
<td>10%</td>
<td>23</td>
<td>18</td>
<td>5</td>
<td>21.74%</td>
</tr>
<tr>
<td>33%</td>
<td>34</td>
<td>20</td>
<td>14</td>
<td>41.18%</td>
</tr>
<tr>
<td>50%</td>
<td>123</td>
<td>28</td>
<td>95</td>
<td>77.24%</td>
</tr>
<tr>
<td>66%</td>
<td>152</td>
<td>55</td>
<td>97</td>
<td>63.82%</td>
</tr>
</tbody>
</table>

probabilities to generate requests.

For these three scenarios (Intersection Scenario, Politehnica University Campus Scenario and Downtown Scenario) we considered that a request message is inserted only once into the network. If request expires, client will no longer make another one.
Chapter 6

Conclusions and Future Work

In this work, we presented EZCab, a VANET computing application for booking cabs in urban congested areas.

EZCab discovers and books available taxis using only vehicle-to-vehicle wireless communication. Unlike traditional approaches based on phone-calls with remote dispatchers, in our approach the client sends directly the request message into ad-hoc network and receives a response. The EZCab purpose is to find and book the closest cab to the requester (the client). By closest we refer to either proximity to the client or we try to optimize the arrival time, for example by choosing the fastest available cab.

EZCab is based on a previous approach, an implementation based on Cooperative Computing model with Smart Messages is available. Older prototype of EZCab computes request in a large area of time. The newly proposed EZCab prototype focuses on the idea of simplifying the cab-to-cab communication protocol. For this we changed SM with simple messages containing only useful information data, without execution units. Every entity in the cab network has a PDA with EZCab application and will be able to compute data received.

The EZCab protocol is simplified, using only three types of messages: request messages, response messages and confirmation messages, last two being having a similar structure and could being minimized to a single type. Messages are simple, short, containing minimum information.

We proposed three different types of architectures: peer-to-peer based architecture, gateway based architecture, which introduces gateway as middle communication entity between clients and cabs from the EZCab network, and emission point
based architecture, based on emission points that forwards messages between them. Each emission point is responsible for a fixed area of city and transmits request and confirmation messages to cabs. Also, taxis send response messages to clients via emission point.

Unlike previous implementations, our prototype uses only entity algorithms based on flooding combined with some optimization criteria. The EZCab architecture is decentralized, cheap and simple, but also has several issues, one of them being that we cannot estimate time of finding a free cab.

EZCab presents new technical and computational challenges which do not exist in traditional systems: distributed protocol for finding at most one free taxi for customer, ensuring that free taxis accept one client request at one moment of time, ensuring that both taxis and client positions are well known for network entities and providing a mechanism for the client and driver to authenticate each other when they change messages.

We implemented and evaluated EZCab peer-to-peer architecture in VNSim VANET simulator as an extension to the DSRC protocol simulation. EZCab implementation comes with new ideas for VNSim: existence of more than the car entity in traffic simulation (in our case, the client object, who is similar to the CarInstance object, but having a different behavior).

We tested the simulated EZCab application using three test scenarios: an intersection map (used to test the changes of messages between cabs and clients) and two real world scenario (for testing the time performances of the EZCab application). We simulated real cases like: a lot of free taxis and few clients, a lot of clients and few taxis, proportional number of clients and taxis, different distances between taxis etc.

Based on the obtained results we concluded that the solution performs well, the time needed to find an available cab being smaller the one encountered most of the time in real-world situations.

In this work, we presented and implemented a solution where cabs are cooperative and willing to propagate data between each other. But in EZCab architecture chapter we also presented a solution where cabs receive and send messages to other cabs from the same company. We plan to investigate the performance of EZCab
when it uses different classes of cabs in which communication happens only between cabs belonging to the same class, classes meaning in real world different companies.

Another extension will allow occupied cabs to act as free candidate cabs based on their scheduled drop-off location and time. For example, is more useful for a client to consider a cab that is scheduled to drop off a client that is very closer to the new client within a minute, which is far better than booking a 10 minutes away cab.
Bibliography


