Fakultät für Elektrotechnik und Informationstechnik
Lehrstuhl für Kommunikationsnetze
Prof. Dr.-Ing. Wolfgang Kellerer

Final Report - Vacant Parking Space Detector

Octavio Rodríguez,
Jure Zdovc

WSN Final Report

Semester:	SoSe 2014
Group:	04
Authors:	Octavio Rodríguez
	Jure Zdovc
Advisors:	Prof. Dr.-Ing. Wolfgang Kellerer
	M.Sc. Arsany Basta
Date:	July 13, 2014

Contents

1 Introduction

An important part of city planning involves estimating the amount of traffic a certain street or city district will support. Cars going to a certain destination will ultimately have to find a parking space. This means that obtaining information on parking space usage could greatly contribute to improvement of urban spaces and regulate parking pricing.

For instance, a study conducted between 1927 and 2001 [1] in several cities around the world showed that between 8 and 75% of the traffic on the streets in the study areas were looking for parking. The average time spent cruising for a parking space in urban public parking areas varies from 3.5 to 12 minutes, with distance travelled ranging from 1 km to over 1.5 km. When multiplied by the number of cars looking for a parking space, this translates into a considerable amount traffic, wasted fuel, and unnecessary air pollution.

A parking space monitoring system could help alleviate such problems. However, current vacant parking space systems are almost exclusively used in multi-story car parks. The reason behind it is that the typical sensor-based parking space detection systems are costly and require a considerable amount of time and effort for installation. Intrusive sensors (e.g. pneumatic tubes, weight-in-motion sensors, piezoelectric cables) need to be embedded in floor and non-intrusive sensors are wired to the ceiling or walls (e.g. ultrasonic sensors, CCTV-based detection).

In this project we want to demonstrate a cost-effective alternative to such systems with a prototype using interconnected wireless sensors. Our prototype uses Crossbow MicaZ motes, but these can be of any other type. The main idea is that the sensors could be installed in parking lots without roof, or even on the streets for curb parking slots. The statistics are then fed to a web server that could be accessed, for example, by the parking lot operator or by smart devices on cars. This means that such a system could provide benefits for both parties:

Benefits for operators	Benefits for drivers
Near-real-time parking lot occupancy information.	Guided assistance in finding a free space.
More efficient and safer traffic flow.	Reduced carbon dioxide emissions.
Parking usage trend analysis.	Lower fuel consumption.
Improved space utilization.	Cruising time savings.
Valuable statistics.	Less frustration.

Table 1.1: Examples of benefits for a cost-effective Vacant Parking Space Detector system.

2 Project Description

2.1 Requirements

In order fulfil the basic needs of any parking detection system, our proposed solution will feature the following:

- Low power consumption – The system will focus on event-driven updates in order to conserve as much energy as possible and prolong time between sensor battery replacements.
- Ease of installation – With a wireless sensor network approach, this system will avoid cabling installation. Also, dynamic routing makes it possible to install sensors in a plug-and-play fashion.
- Reliability – The routing protocol focuses on reaching the control center keeping several alternate paths ready in case of connectivity loss.
- Error detection – By keeping track of each device's power level on every message, sensors running low on battery can be quickly identified and get a replacement before losing connectivity.
- Fast reaction times – The system will to notify parking status changes on time for the next car looking for a parking spot.

2.2 Topology

The two scenarios where this solution could be applied would be mainly in parking lots, and multi-story car parks, but also curb parking spaces. In order to communicate the status of the parking spaces, wireless motes are programmed with one of three roles (see figure 2.1):

1. Sensor - Mote running on batteries equipped with a light sensor and encased in a protective shell that is fixed to the floor or pavement. It is responsible for monitoring parking space occupancy and inform its immediate Gateway of a status change.

2. Gateway - Mote connected to a power source in an elevated position (e.g. a street lamp). It provides network access to the Sensors by forwarding their status messages towards the control center. It also builds a distance-based routing table that consists of neighbor Gateway addresses and their distance to the control center.

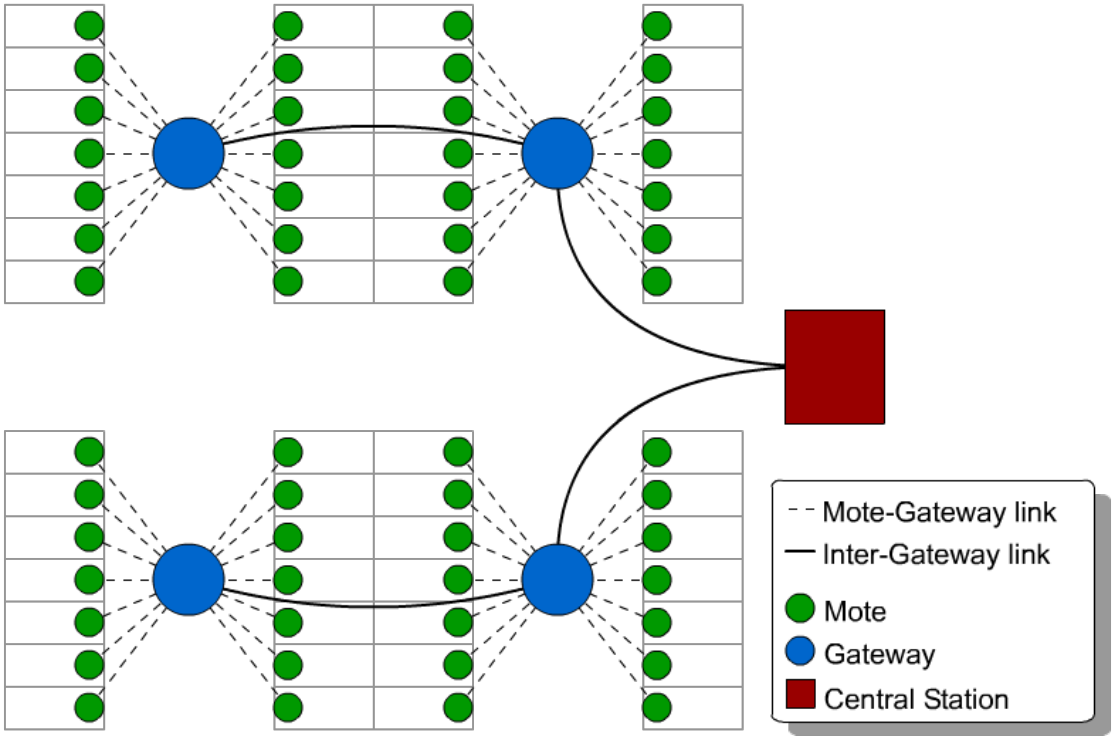

Figure 2.1: Example topology of a small parking lot

3. Central Station (CS) - Mote connected to a computer or web server. This device will receive and process all status messages from every Gateway. It also provides statistical information and GUI for the parking lot operator and information to be printed in an outdoor display for the drivers.

The complete space detection and status forwarding is illustrated in figure 2.2.

2.3 Routing

The routing protocol proposed is dynamic and meant to ease installation in a plug-and-play fashion. The objectives of our implementation is to develop a simple, yet effective and fast route selection. It should provide quick reaction times and backup path switch-over in case of Gateway saturation or connectivity loss. Lastly, it should allow new Sensors or Gateways to be added to the network with as little configuration effort as possible, if any. We decided a protocol mix of Dijkstra and rumor routing could meet all these requirements.

2.3.1 Rumor Routing

In the original concept of query-based rumour routing [2], each node has a neighbor list and an events table. The events table contains forwarding information to all the events it knows. After a

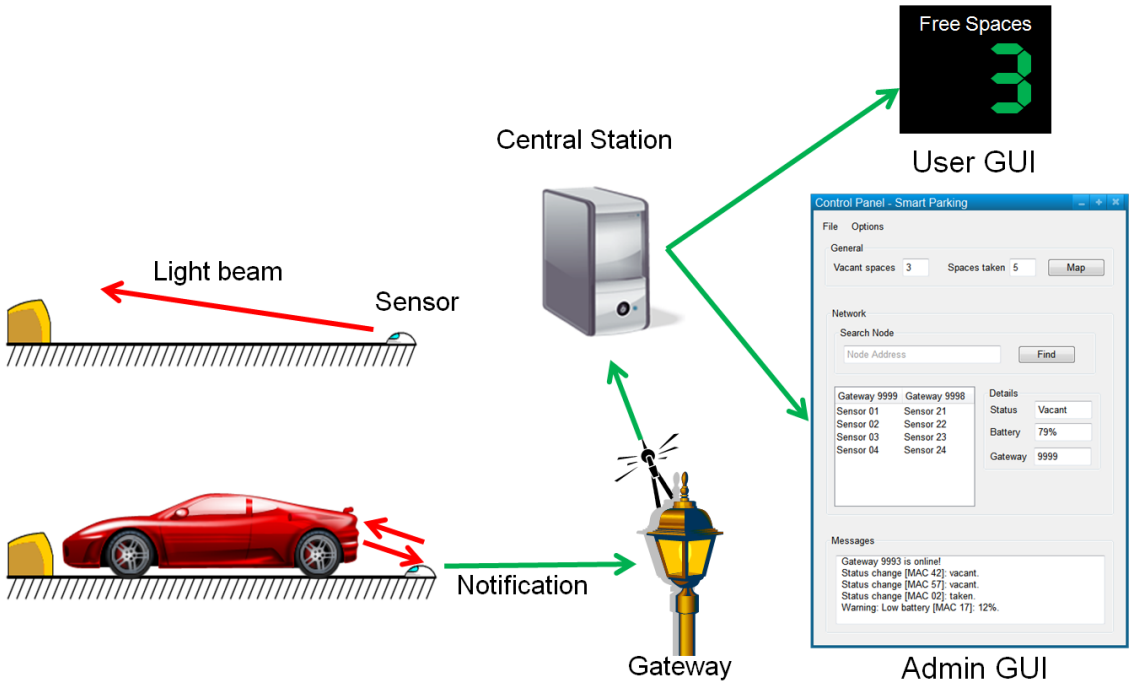

Figure 2.2: System setup: Sensor motes are fixed onto the ground and detect if a car is using the parking space. If the parking status changes, the Sensor notifies its Gateway. The Gateway injects the notification into the network where it reaches the central station. The central station publishes the information via an Admin GUI or a public display on the street.

node witnesses an event, it sends a special long-lived packet to the network known as an agent. The agent travels around the network hopping from node to node creating a path originating from the event.

Once there are events, agents and paths on the network, any node may send a query packet for a particular event into the network. If the query packet reaches a node who knows the route to the event (as told by a passing agent), it will forward the query. Otherwise, the query will be sent and propagated in a random direction until the query reaches a node with a route to the event.

Properties

- Power saving, improves network longevity.
- Robust in dealing node failure.
- Tunable, allow trade-off between setup overhead and delivery reliability.

Assumptions

- No coordinate system is available so location information is not known by nodes.
- Static topology.

2.3.2 Dijkstra's algorithm for best path selection

This algorithm seeks the best path from a source node to a destination node in a given network. It starts by assigning a tentative distance value to every node: set to zero for the initial node and to infinity for all other nodes. Marks all nodes unvisited. Sets the initial node as current. Creates a set of the unvisited nodes called the unvisited set consisting of all the nodes.

Once all these initial settings are done, an iterative process begins in which each node is evaluated for the shortest distance:

1. For the current node, all of its unvisited neighbors are evaluated and their tentative distances are calculated.
2. It then compares the newly calculated tentative distance to the current assigned value and assign the smaller one.
3. When all of neighbors of the current node has been evaluated, the current node is marked as visited and removed from the unvisited set.
4. If the destination node has been marked visited or if the smallest tentative distance among the nodes in the unvisited set is infinity, the algorithm stops.
5. The next step is to take the unvisited node with the smallest tentative distance as the new "current node" and go back to the first step.

Properties

- Clever way to use cost/distance to select a path.
- Focused in looking for one path only (i.e. the algorithm must be run on every device).
- Looking for the shortest path by flooding consumes a lot of energy.
- Flooding increases the chance of collision.

2.3.3 The rumor-Dijkstra hybrid

As we have seen, rumor routing and Dijkstra's algorithm have attractive properties that could provide advantageous to our purposes. In order to provide a practical, simple and fast solution we had to mix the most convenient features to our topology :

- After booting, a new node will broadcast a special packet called GW_REQ (meaning "Gateway request". See section 3.0.6 for more details on packet types) announcing its MAC address and its initial distance to the central station (255, the equivalent to infinity).
- Every Gateway that receives a GW_REQ and is currently available, must reply with another special packet, an UPDATE, which contains its MAC address and its distance to the CS.
- The new node will wait five seconds to receive as many UPDATE packets as possible. After that, the device reads each reply and selects the five best metrics.
- In the case of Sensors, the metric is the Received Signal Strength Indicator (RSSI; the higher, the better) and for Gateways is distance to the central station (the lower, the better).

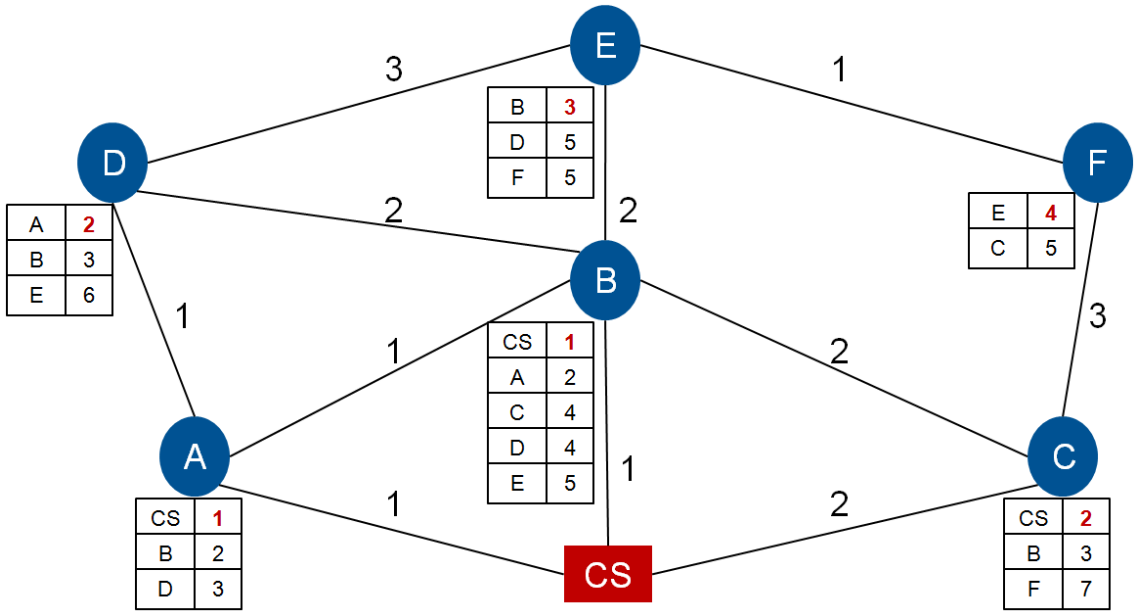

Figure 2.3: Example of best path discovery. The number in the lines connecting the nodes are the weights taken from the RSSI values of each link: 1 if $>$ 0 , 2 if $>$ -15, 3 if $\leq$ -15. The blue circles represent Gateways and the red square is the central station (CS). Each node keeps a table with five of its neighbors with the shortest distance to the CS. The red number is the distance each node reports in UPDATE packets.

- After a Gateway reads the distance of an UPDATE, it also checks the signal strength of the packet and adds a weight depending on the RSSI value:

  $$+1 \text{ if } > 0 \text{ , } +2 \text{ if } > \text{-15, } +3 \text{ if } \leq \text{-15}$$

- The replies with the five best metrics are put in the routing table. In the case of Gateways, the best distance (+ RSSI weight) in the table becomes its own distance. the device can begin with the tasks specific to its role (Gateway or Sensor).
- If there was no response at all, the device will repeat the process over and over again sending GW_REQ packets until it receives an UPDATE.

In contrast to the original rumor routing concept, the hybrid does not consider events as information source or an agent traversing the network. Instead, it makes emphasis on routing by exchanging information with neighbors only. Their only interest is about their distance to the central station and/or signal strength.

Also, instead of running a Dijkstra algorithm to check every single node for their link cost and update it after every calculation, we took the concept of cost and a step-wise evaluation of the best path to a specific node. This process happens locally on every node and this information will be exchanged with neighbors only.

UPDATE packets (equivalent to queries) are sent in three situations: during network discovery (see section 3.0.4 for more on the network discovery), if a device had to delete its routing table, or if its best route changed. This way routing changes are propagated from Gateway to Gateway.

3 Implementation Details

3.0.4 Network Discovery

The first thing a device does after booting, is to see which Gateways are in its vicinity and build a routing table out of the information exchanged with their neighbors. The process is basically the same for all three roles with a few minor differences (see figure 3.1).

1. Once a device has booted, it generates a MAC address, which is nothing more than the hardware's serial number.
2. A GW_REQ is broadcasted with the device's MAC address. Every Gateway or CS who receives a GW_REQ, replies with an UPDATE packet. The UPDATE contains the replying Gateway's MAC address and its distance to the CS.
3. The device will wait for five seconds to accumulate as many replies as possible.
4. If there were no replies, the process gores back to step 2. Otherwise, each response received is read and evaluated.
5. The responses with the five best metrics are put in the routing table. Gateway metric is reported distance to the CS + weighted RSSI. Sensor metric is signal strength.
6. If the device is a Sensor, it will send a parking status to the best Gateway in its table.
7. If it is a Gateway, the best metric in the routing table becomes its distance to the CS.
8. If it is a Gateway or the CS, it will broadcast an UPDATE with its distance to the CS.
9. Lastly, the device will start its normal operation according to its role.

3.0.5 Parking Status Propagation

After a Sensor has built a routing table, it maintains communications with only the Gateway with the best metric. From this point, the Sensor will spend as little battery as possible. It does this by staying putting the wireless communications module in sleep mode. The light sensor (or infra-red, or any other type that best suits the environment) will probe its surroundings periodically to detect any changes. As soon as the light sensor detects a change, it will send an interruption signal to the wireless module to wake up. Then, the process described in figure 3.2 takes place:

1. The Sensor looks in its routing table for the first element (best metric).

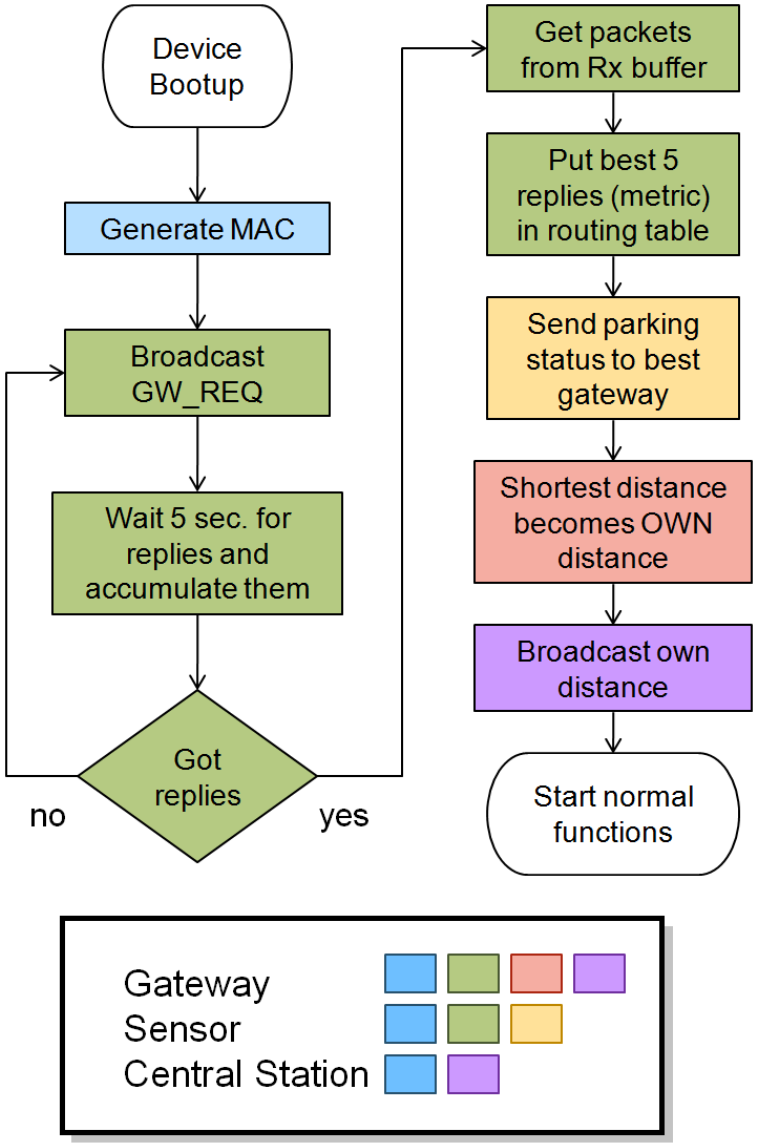

Figure 3.1: Network discovery process for the three roles.

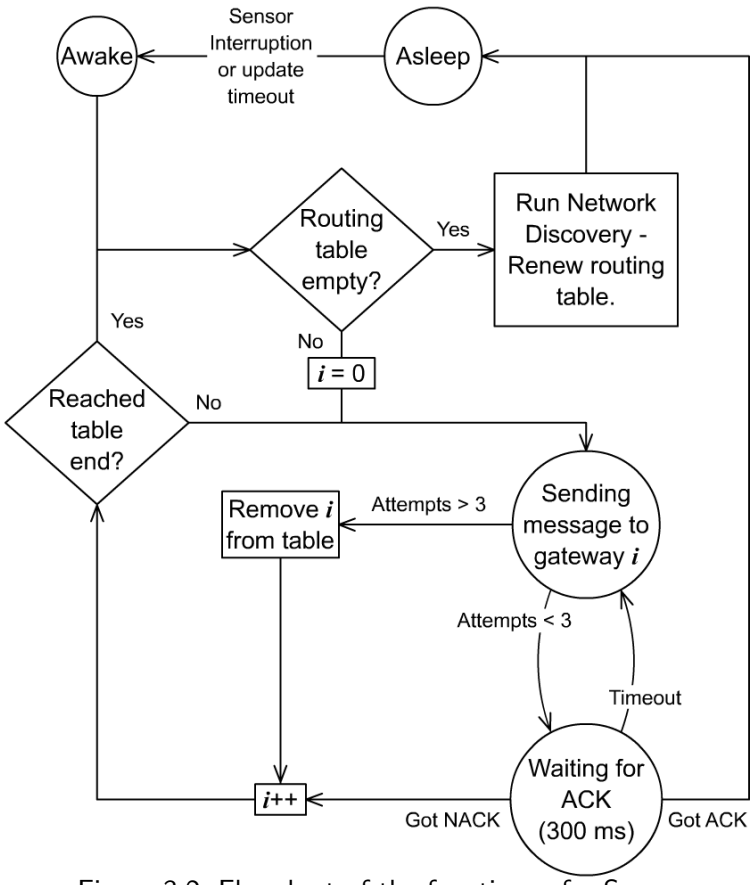

Figure 3.2: Flowchart of the functions of a Sensor.

2. If the table is empty, it runs the network discovery process. Otherwise, it extracts the MAC address of the first gateway in the table.
3. The Sensor builds a special packet depending on the current parking status: either OCCUPIED or EMPTY. This packet contains the Sensor's MAC address and is destined to address extracted from the table.
4. The message is sent to its destination and the Sensor waits 300 ms for an acknowledge packet (ACK).
5. If an ACK is received, the Sensor goes back to sleep mode.
6. if a NACK is received, the Sensor goes back to step 1, but uses the next gateway in the list on step 2.
7. If the ACK timed out, it tries two more times with the same address.
8. If there was no response, the address to which it tried is removed from the routing table and goes back to step 1.

In the case of a Gateway, the process is very similar. The main difference is that a Gateway waits for messages, for its task is to forward them. It does not go into sleep mode, because it is connected to a light post and has an "unlimited" source of electricity. But also, a Gateway sends ACKs if it receives a status message and is currently not busy. Otherwise, it will send a NACK.

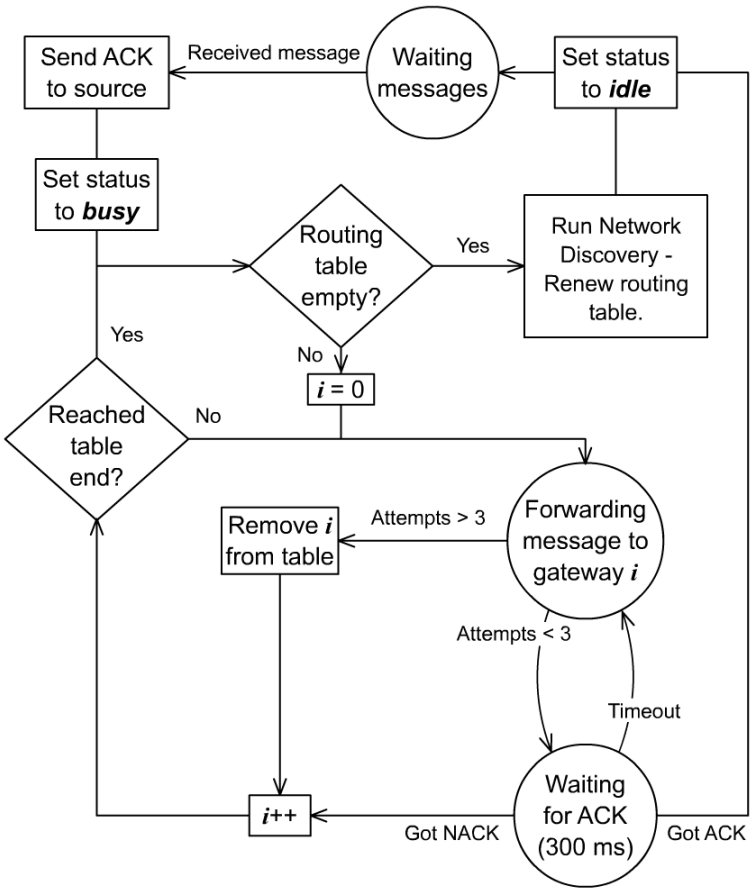

Figure 3.3: Flowchart of the functions of a Gateway.

The complete process is described in figure 3.3.

3.0.6 Packet Format

In order to provide an agile network convergence, a packet format was devised, which was small, but had the minimum information necessary to meet the system requirements. Each packet contains four fields (none larger than two bytes): source address, packet ID, command and value. In the previous sections all five packet types were mentioned in context, but a summary of them can be found in figure 3.4:

- ACK: Used by Gateways to respond to parking status notifications.
- UPDATE: Used by Gateways and the CS to notify neighboring devices of its address and distance to CS.
- OCCUPIED: Sent by Sensors to notify a parking space is now occupied.
- EMPTY: Sent by Sensors to notify a parking space is now free.
- GW_REQ: Sent by Sensors and Gateways in the network discovery phase to establish communications with neighboring gateways and create a routing table.

	SRC	ID	CMD	Value
ACK	Gateway Address*	Packet ID Number	ACK	Busy status
Update			UPDATE	Distance
Occupied			OCCUP	Source sensor addr.
Empty			EMPTY	Source sensor addr.
GW Request			GW_REQ	**unused

Figure 3.4: Packet Format. $*$ In the case of occupied and empty, the SRC field originally contains the Sensor's address. $**$ The Packet is fixed length, so the unused value is arbitrarily assigned.

3.0.7 Graphical User Interface

In order to make things lighter for the parking lot operator, a Graphical User Interface (GUI) should provide all the relevant information about the parking and device status as well as some statistics. For this prototype, we only included the critical information, which is:

- The number of vacant and occupied parking spaces
- A list of Sensors that have connectivity to the CS.
- Each Sensor's MAC address.
- Number of packets sent by each Sensor
- The reported parking status of each Sensor.
- The time of the last update of each packet.

Additionally a debug output window is included, which shows all messages sent and received by the CS to and from its neighbors (see figure 3.5).

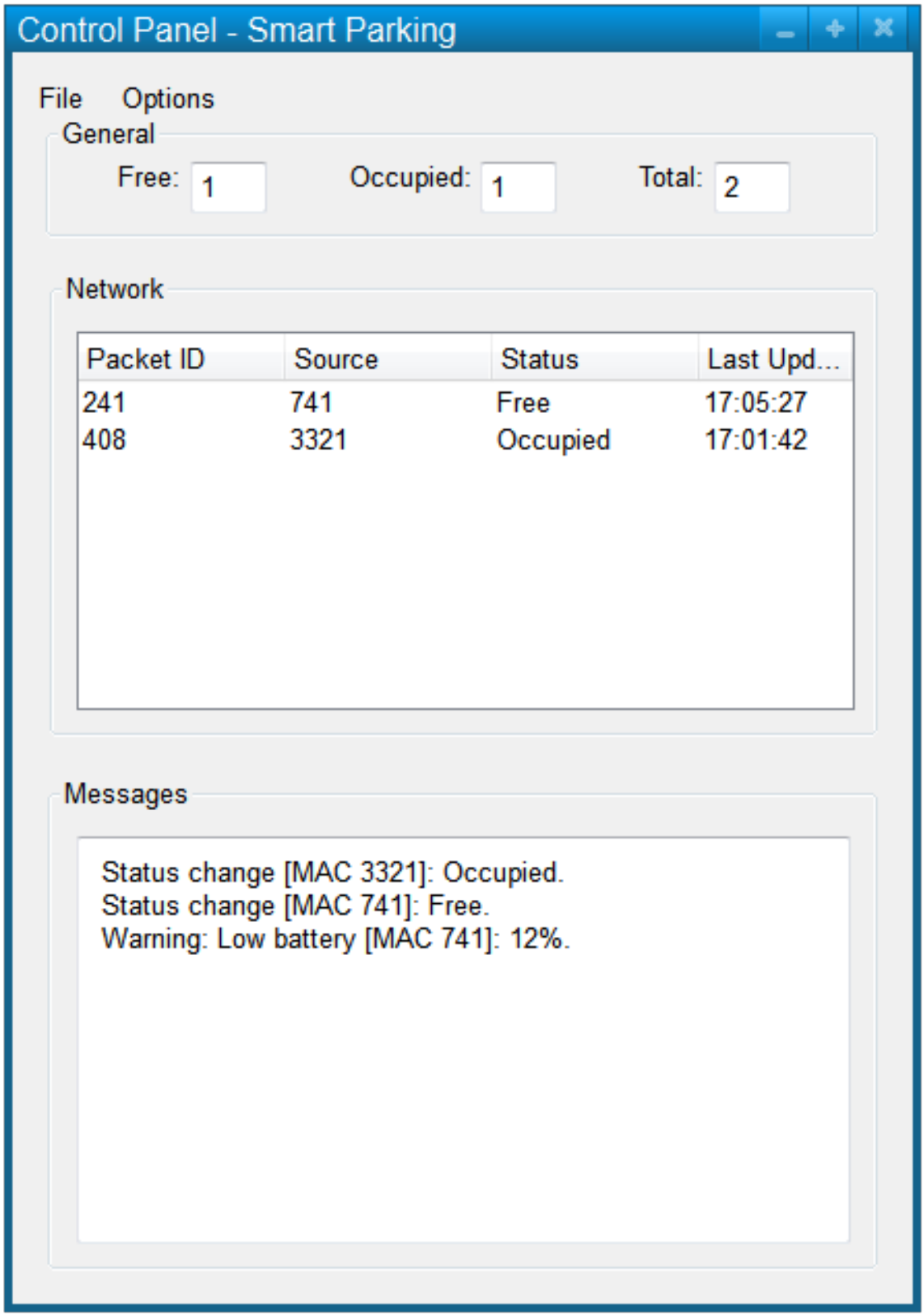

Figure 3.5: Administrator's GUI

4 Summary

Parking detection systems are not the rare thing to see. However, it is the cost and installation that makes them attractive only on big park houses. Our project confirmed that a more cost-effective alternative is possible using wireless sensor networks. Implementing them on every parking space can only bring benefits to both drivers and parking operators, and potentially to a city administration as well.

In our implementation we focused on low power consumption, ease of installation, reliability, error detection and fast reaction times. In the demonstration on the lab, it was seen that the routing protocol hybrid meets these requirements. Additionally, the solution we devised is simple, effective and uses very little resources (event-driven updates, small packet sizes, low network flooding). Also, the modularity of the software allowed small adjustments to provide all the required functionalities for each of the three roles. Lastly, the GUI adds a user-friendly way to present the parking space information and gain a broad overview of the general status.

A List of Figures

B Bibliography

[1] Donald Shoup. Cruising for parking. *Access Magazine*, 2007.

[2] David Braginsky and Deborah Estrin. Rumor routing algorithm for sensor networks. In *First Workshop on Sensor Networks and Applications (WSNA)*, pages 1–12, 2002.

[3] D.B.L. Bong, K.C. Ting, and K.C. Lai. Integrated approach in the design of car park occupancy information system (COINS). *IAENG International Journal of Computer Science*, 2008.

[4] Amin Kianpisheh, Norlia Mustaffa, Pakapan Limtrairut, and Pantea Keikhosrokiani. Smart parking system (SPS) architecture using ultrasonic detector. *International Journal of Software Engineering and Its Applications*, 2012.