Athens University of Economics and Business
Department of Computer Science

Bus Positioning and Utilization System

1 The BU.P.U.S Team

Team Mentor

Georgios Papaioannou, PhD.
IEEE Member
e-mail: gepap@fhw.gr

Team ID

678NXZ

Team Members

Name	Leontiadis Nektarios[1]	Name	Kemerlis Vasileios
Email	p3010099@dias.aueb.gr	Email	b.kemerlis@ccslab.aueb.gr
Department	Computer Science	Department	Computer Science
Year of study	4th	Year of study	4th
Age	21	Age	21
Gender	Male	Gender	Male
Name	Gjoka Mina	Name	Zyba Gjergji
Email	p3000017@dias.aueb.gr	Email	p3010039@dias.aueb.gr
Department	Computer Science	Department	Computer Science
Year of study	5th	Year of study	4th
Age	22	Age	22
Gender	Male	Gender	Male

[1] Team leader

2 Abstract

Two of the most common problems regarding the use of public transport are the loss of time waiting for buses to arrive and the lack of guidance in scheduling one's route within medium or large sized cities, where complicated and interconnecting bus and tram networks are used by thousands of people every day. Judging by our own experience, living in the 5-million-citisen city of Athens, the time people lost due to those two factors is significant. Even if a person has established a regular route to his/her destination (e.g. place of work), traffic and unscheduled events (like strikes or road works) usually force overheads in public transport schedules and rerouting of vessels though alternative bus/tram stops.

Our objective is to help pedestrians minimise the time spent moving around a city and the frustration induced to them by the two factors mentioned above when using the public transportation system, thus making the later more efficient and user-friendly. The goal of our team is to develop and deploy an information system, which will provide specific information to pedestrians that use the public means of transportation, regarding an optimal way of moving through town, including bus proximity estimation and actual timings, destination and traffic-driven route scheduling and more. We named that system BUPUS (**BU**s **P**ositioning and **U**tilization **S**ystem). The user should be able to obtain such information using affordable small mobile devices with limited computational resources (e.g. PDAs, smart phones) that many people carry with them every day anyway. The supporting infrastructure that should be provided by bus agencies and transportation authorities is also based on common low-cost computing and telecommunications hardware.

The wireless information network of the BUPUS project can host other public services as well, including both general and location-sensitive information, available to the users through the active bus stops, which are inherently well distributed along the streets of a city.

3 System overview

BUPUS relies on Wi-Fi technology to provide bus schedule, timing and routing-across-city information to pedestrians anywhere in the vicinity of a BUPUS Smart Bus Stop (SBS), which acts as a hotspot and information-gathering terminal node. PDA or smart phone users can freely have access to the BUPUS service and obtain information regarding bus stops where the next buses for a given destination or route will arrive first, routing to various locations and time-of-travel estimations. Hand-over of the mobile device control is handled by the 802.11 protocol. Buses passing through an SBS produce an RF-identified time stamp, which is then propagated down the low-bandwidth, star-structured network of bus-stops and eventually causes an update to a centrally hosted statistical prediction model for traffic and routing times. This information is periodically transmitted back to the bus stops so that users can have fast responses at their queries.

3.1 System Description

BUPUS can be divided into three active subsystems and a supportive subsystem, which has the responsibility of interconnecting them. In detail, we have the following elements (Fig. 1):

Service Accessibility Subsystem
The purpose of the SAS is to provide the end user with the service that we are implementing, using the server-client architecture. The client part consists of the application that runs on the users' Wi-Fi enabled devices. The server is located at the Smart Bus Stops (SBS), and serves

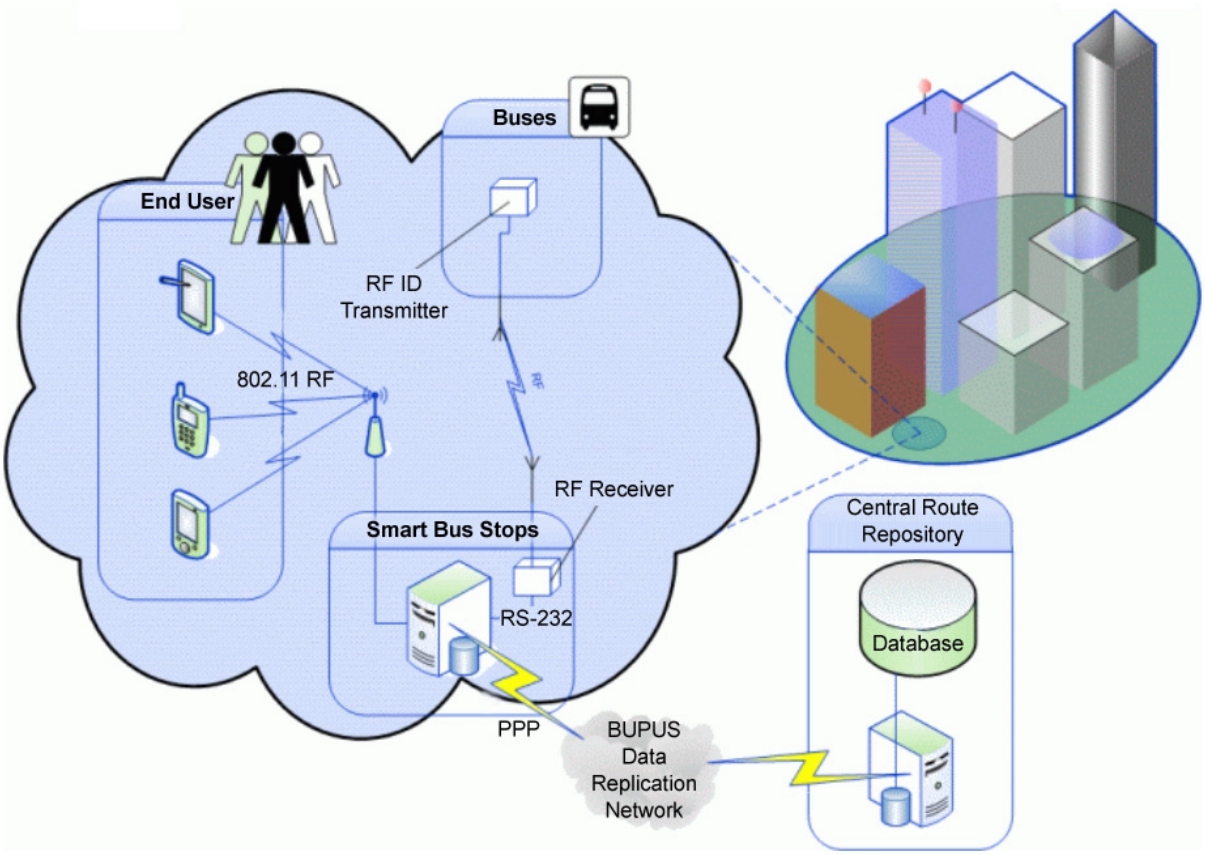


Fig. 1. System Overview

the clients by processing the queries submitted to them, using the local copy of the database (Local Route Repository - LRR).

Bus Identification Subsystem
The BIS is the subsystem, which provides all the input information needed by the system. It consists of RF transmitters placed on the buses and the RF receivers placed on the SBS. When a bus is within the range of the bus stop (approximately 15 meters) the ID of the bus is transmitted to the RF receiver and propagated along with the time stamp to the BRDRS.

Bus Route Database Replication Subsystem
The BRDRS consists of the SBSs and the Central Route Repository (CRR). The SBSs, as a whole, replicate the information collected by the BIS, to the CRR. The Central Route Repository (CRR) collects data from each SBS and then performs some statistical processing to extract normalized bus timings for all combinations of bus stops. All the timing estimates are then properly efficiently encoded and transmitted back to the SBSs. See section 4 for more details.

3.2 Innovation

Work conducted so far in this research and application area is focused on trying to inform citizens about bus arrivals via kiosks or other static information media, usually located at bus stops. These systems provide limited amount of information to end-users. BUPUS on the other hand aims at a mobile, location-sensitive version of such an information system, provided users are in possession of small portable devices, which support 802.11 and java technology, such as smart phones. The mobility aspect of BUPUS significantly extends the range of on-line services that may be requested and provided by such a system. Users will be able to have a constant update of available options for circulation and, in essence, a public-transport-based navigator system with optimal routing and time-of-travel estimation.

3.3 Design Methodology

The project life cycle model used is a modified version of the well-known waterfall model (system concept, system requirements identification and analysis, architectural design, detailed design, implementation, system testing and system deployment). We chose to use waterfall with overlapping phases and subprojects model. This model was imperative due to the limited time available for the project implementation because it allows a greater degree of parallelization of tasks while maintaining a coherent structure for the project as a whole. Special care has been taken in the RFID sub-project, where alternative, fall-back solutions where concurrently investigated in case the implemented components failed to meet the task requirements.

3.4 Performance Requirements

The user should be able to obtain the information supported by BUPUS via affordable, mobile devices with limited computational resources (e.g. PDAs, smart phones). Usable data need to be distributed and as closely located to the end-user as possible, in order to ensure quick response times and encourage the users to use the BUPUS service. Moreover, the snapshot of the information contained in the LRR should represent as much as possible, the *current* traffic conditions so that time prediction can be more accurate. That means that every piece of information in the LRR should not be frequently updated, preferable once a minute. At the same time, taking into account the need for affordable links between the bus stops and the higher network nodes, inevitably traffic to and from bus stops has to be very efficiently channelled through connections, as low in bandwidth as 64Kbps.

4 Implementation and engineering considerations

4.1 Service Accessibility Subsystem

The Service Accessibility Subsystem provides the BUPUS service to the end-users. This subsystem is consisted of two parts, the client application and the service provider distributed at each Smart Bus Stop.

The client application is the application installed on the users' devices (PDAs, Smartphones). Through the application's graphical interface, the user is able to utilize the miscellaneous services provided by our system. The application collects the user's input and with this information builds a request message. Then, having a response message received, projects the useful information. The device *must* have the following characteristics:

- Capability to use a wireless network connection using the 802.11b protocol (Wi-Fi). The device will be using this network interface in order to communicate with the rest of the system. Through this communicative line, queries will be made and answers will be returned. This choice of using this specific protocol was taken given the challenge of keeping the cost down while providing this service for free to the users. Using existing wireless technologies such as cell phones with e.g. gsm (sms) or i-mode could cost a considerable amount of money. BUPUS uses license-free band frequencies for wireless communication. Of course, the design of BUPUS allows for adaptation and utilization over GSM or other protocols in order to be adopted by telephony providers as a part of their services to customers.
- Java Runtime Environment. The application is implemented using Java Mobile Edition supporting the Connected Limited Device Configuration (CLDC) version 1.0 and the Mobile Information Device Profile (MIDP) version 1.0. Target devices should have a compatible virtual machine. The choice of using Java to implement the application was made, based on the factor of portability that this language provides. This characteristic enables the application to be executable in a variety of mobile devices and operating systems (PalmOS/Linux/Pocket PC/Symbian).

The application presents the user of the following choices:

- **Route me.** This is the main procedure. Through this, the user can select a starting and a destination bus stop and the system will provide him with the best route connecting these points and the estimated time of arrival to the destination. The interface guides the user to select the desired bus stops by searching either by bus stop title prefix or by region. The results of the search are presented as a list from which the user can select the one he/she wants. The user input is very intuitive and uses local route and region naming conventions to accomplish the task quickly and easily. The system builds the route by calculating the shortest path connecting the two points. For this purpose, Dijkstra's algorithm for Shortest Path (SP) finding is used.
- **Show nearby bus stops.** This procedure provides the user with the bus stops in the vicinity of the request. For this purpose, the query result contains all the bus stops located within a radius of 100 meters from the bus stop that acts as a service hotspot (itself included), which is practically, the closest bus stop to the user. Each bus stop maintains a fixed table of its nearest peers, as well as their locations coordinates in the Global Positioning System (GPS). Therefore, this query is trivially implemented.

- **Show buses stopping at a bus stop.** The user selects the desired bus stop through the process described above and the system provides him/her with the bus lines, which pass through this specific bus stop, in the form of a list. Then, through this list, the user can select a bus line and view its route.
- **Show me the route of a bus line.** The user inputs the bus line ID and the system provides a path-sorted list of the bus stop names. The user can use this service interchangeably with the previous query to further select a bus stop from the list, and see all the bus lines that stop at this specific bus stop. The combination of the two queries provides a manual way to navigate through the bus network.
- **When this bus is going to be there.** This procedure guides the user through a process of selecting the desirable bus line and bus stop (using procedures described above) and, as a result, the system provides the estimated arrival time of the next bus on the selected route.

Fig. 2. The BUPUS client

The service provider is placed within the Smart Bus Stops and is responsible for serving the users' requests, by using the information stored in the Local Route Repository. It is implemented using Java, so that in the process of the communication between the two parts of the subsystem, the format of the information exchanged, will be universally the same, thus reducing the processing power needed.

There was a thought for encoding the messages using XML in order for the procedure to be more standardized, but it was quickly rejected because of the computational resources needed for parsing the xml document on the client side. Moreover, the use of XML would introduce overhead in the information exchanged.

Each response to a request is built and formatted so that the user's device will not be burdened with any processing intensive work. In the transaction process within the Service Accessibility Subsystem, packets are exchanged through a reliable stream. The stream is opened whenever a transaction is needed and closed when it is complete. The connection-oriented approach was exercised because of the characteristics of the environment the service is going to be used in, namely, the industrial noise in the city, causing many packets not to reach their destination and making an error recovery subsystem mandatory. The format of the messages exchanged is depicted in figures 3 and 4.

The procedure of requesting information could be briefly described as follows: The user's application builds a *request message* based on the user's input, opens a network connection to a server at the closest Smart Bus Stop and transmits the request. Then the server, having built

Fig. 3. Request message format

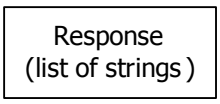


Fig. 4. Response message format

Table 1

Client request operation IDs and operand data

Operation	Parameter(s)	Description
ROUTE	[start , destination]	Calculates the route from <start> to <destination> and returns the transitions needed and the estimated arrival time
GET_BST_BY_PREFIX	[prefix]	Retrieves the titles of the bus stops that start with <prefix>
GET_BST_BY_REGION	[region]	Retrieves the titles of the bus stops located at <region>
GET_BL_BY_BST	[bst]	Retrieves the bus lines that stop at the bus stop <bst>
GET_BL_ROUTE	[bl]	Retrieves the route of the bus line with id <id>, as titles of bus stops
GET_BUS_ARRIVAL_TIME	[bst , bl]	Retrieves the estimated arrival time of the next bus of the bus line <bl> at the bus stop <bst>
GET_REGIONS		Retrieves the regions whose bus stops are BUPUS enabled

the *response message*, transmits it back to the user's device and the connection is closed. The mapping of operation IDs is presented in table 1.

4.2 Bus Route Database Replication Subsystem

4.2.1 Local Route Repository

All the LRR data are held in a relational database. Before designing the database scheme we took the following assumptions into consideration:

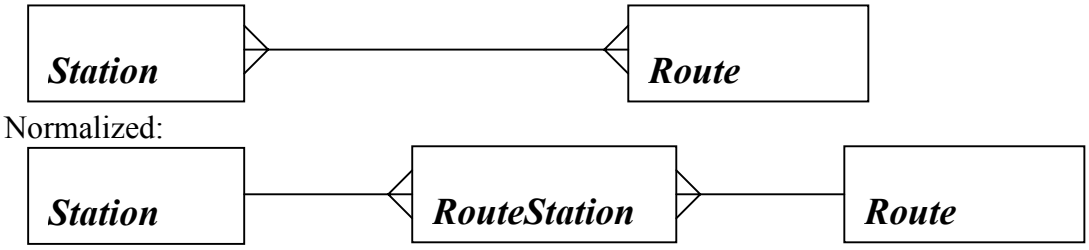
- Each bus (as a vehicle) has a unique number/ID assigned
- Each stop has a unique station ID assigned to it. Bus stops on the same location and route but opposite directions are considered to be different and are consequently assigned different IDs, even though they may share computing and networking resources in the implementation level.

- Each Route has a unique route ID assigned
- A Bus Line Number (BLN) may correspond to a circular route or a bi-directional one. In the later case, it is assigned to two different routes each one representing an opposite direction.

Data are separated into dynamic and static ones.

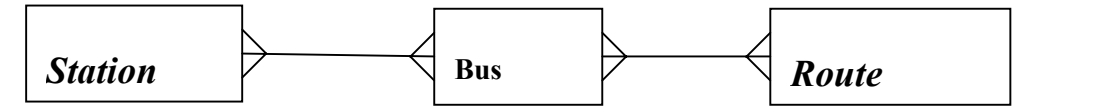
Dynamic data are generated continuously and describe the current status of the transportation system. By current status we mean the estimated arrival time of currently active buses to specific bus stops. LRR gets dynamic data from CRR on a repeated basis. Static data represent the pre-designed bus routes and stop locations and are locally stored in the LRRs.

The Entity-Relationship Scheme for the static Data is:

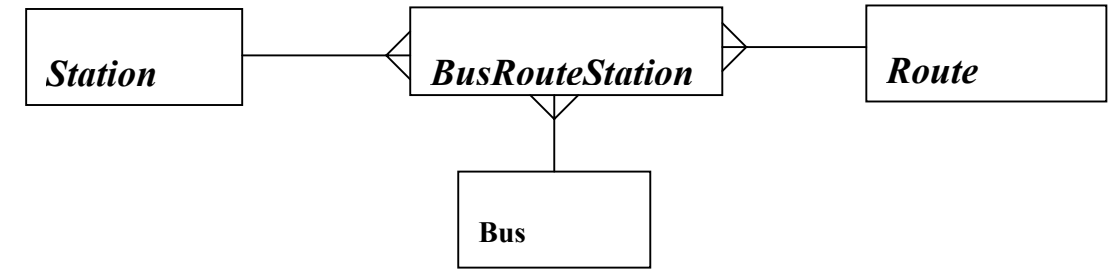


Normalized:

The Entity-Relationship Scheme for the dynamic Data is:



Normalized:



Based on the E-R, the resulting tables are presented in Fig. 3. The Station and Route tables correspond to Station and Route entities, which appear in both dynamic and static data design. The RouteStationInstance table corresponds to BusRouteStation Entity. It keeps the estimated time a bus will arrive at a specific station on the route.

Each bus stop has a specific set of (X,Y) coordinates, which determine its position and are used in calculating nearby stations at the initialization of the system, as well as for informing the clients for the exact location of a bus stop on the map.

All specified user queries are extracted with SQL queries except from the estimation of the best route given the starting end ending location (bus station). This question is more complex for a single SQL query toproduce a meaningful answer. To this end, we used a modified version of the Shortest Path Dijkstra's Algorithm.

We considered a weighted directed graph where:

- Vertices are all the stations

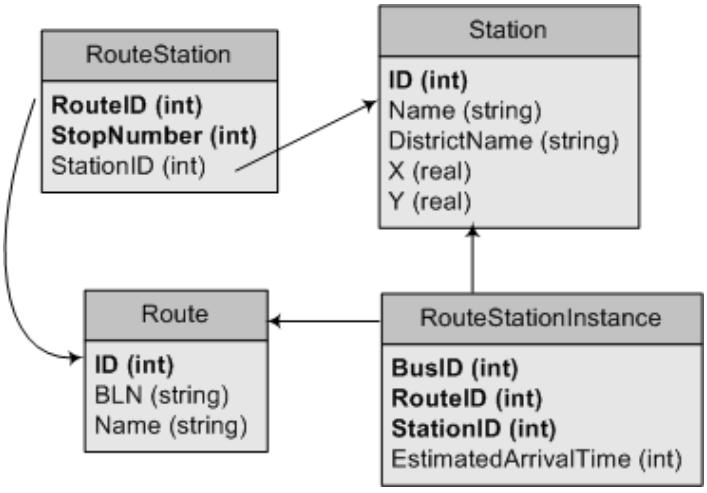


Fig. 5. The LRR database tables that describe the current state of the system.

- There exists an edge from node s_1 to s_2 if and only if a bus is going to be located on s_1 and after that on s_2 (physically next stop, depending on its assigned route). The weight is defined as the time the bus is estimated to travel from s_1 to s_2 (there may be many outgoing edges from s_1 to s_2 depending on the number of buses that include s_1 and s_2 in their route with $bus.\text{EAT}(s_1) < bus.\text{EAT}(s_2)$). $b.\text{EAT}(s)$ is the estimated arrival time of the bus b at station s.

Because of the extra dimension of time, the original Dijkstra's algorithm is not adequate to solve the problem. Let a vehicle arrive in station s_1 at time 10, after some steps of execution. 0 is the time (distance) of the station that the vehicle started from. Dijkstra's algorithm will take into consideration all the buses including all those that arrive before time 10 from different bus routes (all outgoing edges). This leads to a wrong route calculation. So we exclude those special occasions every time we have to move from station s1 to station s2. Furthermore we improve performance cutting all graph edges that correspond to buses of the same line that arrive after time 10. Similarly, all buses of the same line except from the first one to arrive are discarded. Apart from the weights in the form of time (distance) that are maintained, the algorithm keeps track of the route hop we made from the previous station. Each hop of the above graph corresponds to a bus line, so after execution of the best route algorithm we know the bus lines the pedestrian should take and therefore the best route is easily found.

The data stored in the database, especially the dynamic part, are unnecessarily duplicated many times. Consider the tuples of RouteStationInstance table: (busID, routeID, stationID, time). Obviously routeID is repeated as many times as many buses are in the same route and each of them has a number of bus stops to "visit". In the same manner, a busID is repeated as many times as the number of the stations it has to pass through. E.g.

[10, 11, 3, 400], [10, 11, 4, 480], […]

A more compressed format would be to hold only the weights of a graph where an edge between two *consecutive* stations exists if and only if a bus line passes (see CRR section). Of course, current bus location would have to be held in a separate table. We decided to use the first format instead of this more space-efficient one because it is optimized for fast query execution. Moreover queries are much simpler. Since we have the expected arrival time calculated for each bus, the best route algorithm is simplified too and faster to execute. The

tradeoff of course is the larger memory requirements. Since the cost of memory is very low and the speed argument is very important we made the above choice.

Another aspect that improved the speed in answering user's queries is the pre-execution of Best Route Algorithm. The above algorithm calculates the best route from a single source/station to any other. Since 90% of the queries of this type require that the best route is calculated from the current bus stop ("I am *here* and I want to go *there*"), we decided that executing the algorithm at the end of every data refresh cycle of the LRRs and storing the results in a separate cached location was the best choice.

4.2.2 Central Route Repository (CRR) and Arrival Prediction

CRR data are held in an SQL table. The table is simple and no E-R scheme was needed to design it (Fig. 6). We used SQL because of its efficiency in selection queries based on attributes, and its aggregation functions. We wanted to avoid re-inventing the wheel building an extended library that provides the same functionality. The tuples of the table contain the time a bus took to move between two consecutive stations s_1 and s_2. of that bus route. Imagine a map of vertices representing bus stops and directional edges representing the segments of a bus line. The duration time is the weight of the edge. The beginning time holds the absolute time of day the bus was located at s_1.

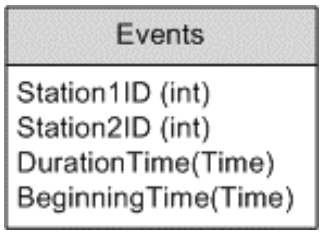


Fig. 6. The CRR Database Table

The data that are sent to an LRR are the latest bus events, plus the appropriate weight adjustments of each event-related (s_1,s_2) segment. The latest bus events are collected from all bus stops and are used to update all bus stops. The weights, i.e. the estimated time-of-travel from bus stop s_1 to bus stop $s2$, are statistically extrapolated from the readings of the actual bus arrivals within a time window.

Before deciding the statistic analysis that should be done for calculating the weights we considered these factors:

- Bus arrivals are not the same throughout the day (e.g. in the morning at 9:00 am there is more traffic than at 14:00 am).
- Bus arrivals differ from one time-of-year to another (e.g. June has much more traffic than August in Athens).

The weight calculation is done using this formula:

$$w(s_1, s_2) = w_1\overline{x_1} + w_2\overline{x_2} + w_3\overline{x_3}$$

where w_1, w_2, w_3 represent weights of importance ($w_1 + w_2 + w_3 = 1$) and

$\overline{x_1}$: average weight today (previous 2 hours)

$\overline{x_2}$: average weight in last week (in this time zone: ± 1 hour)

$\overline{x_3}$: average weight in last 2 months (in this time zone: ± 1 hour)

The statistical data are estimated based on the following algorithm:

```
While (true) do
     Listen for events from LRRs;
     If a previous event is related to the new event (same route)
```

```
          Insert (station1, station2, duration time, beginning time) into database;
          Remove previous event from pool
     End if
     Put new event into "previous events" pool
end while

Every n seconds do:
     For each available edge
          Calculate weights using w(s1,s2) = w1·x̄1 + w2·x̄2 + w3·x̄3 formula
     Send the last location and time of each bus to all stations;
     Send calculated statistic data to all stations
```

Data older than 2 months are automatically deleted. The weights w_1, w_2, w_3 are configurable and depend on how dynamic we want the calculated data to be. In general, $w_1 \geq w_2 \geq w_3$.

4.2.3 Subsystems Interconnection (LRR - CRR)

The bus stations, being possibly many hundreds in number, are connected to the backbone service network through narrowband channels in the worst case. We have assumed this limited bandwidth for the terminal interconnection nodes at least for the performance analysis. We have also assumed for the needs of this project that a dialup/ppp system is already deployed, which connects the bus stations with the previous traffic aggregation level. On the other side of the BUPUS network is the CRR connected through a leased line. The network itself is assumed to be a star topology of active network components to facilitate the grouping of bus stations per area and provide easy scalability of the system as a whole (See Future Work section for alternative solutions).

CRR → LRR

Since CRR is continuously informed of bus arrivals in all stations, it can calculate all the data that are needed to be sent back to all stations. The data consist of two parts: the first represents a snapshot of the current location of a bus, while the second consists of the calculated weights:

stationID	busID	routeID	Time
16 bits	16 bits	16 bits	16 bits

station1ID	station2ID	Time
16 bits	16 bits	16 bits

In the first tuple, time, represents (in seconds) the interval that has passed since the bus was last registered and in the second, estimated time-of-travel between the successive stops.
The amount of data (in bits) sent from CRR to LRR are easily calculated with the formula:

$$a = 64 \cdot b + 48 \cdot n(e) = 64 \cdot b + 48 \cdot s \cdot k / 2$$

Where:
b = the number of buses that are currently active
s = the number of stations (stations on the opposite side of road are considered different ones)
k = the average number of neighboring stations to one station
$n(e)$= the total number of edges in the graph

Assuming that b = 1000, s = 3000 and k = 3 (realistic numbers based on Athens). Then the total amount of data is 64*1000 + 48*3000*3/2 = 64000 + 216000 = 280kbits.

Using 56 kbps narrowband connections from the last interconnection layer to the smart bus stops, with 60% effective bandwidth the time it takes to inform all stations in parallel is 280/(64*0.8) = 5.46 seconds, which is more than enough. Typical time between intermediate refreshes is 20-40 seconds. So, supposing we want data to be transmitted within 20 seconds then 280/20 = 14kbps effective bandwidth is enough.

On the other end of the interconnection network, CRR must send this information to all stations. CRR keeps always open a TCP/IP connection with any station, and it should send calculated data to any of them. The best solution would be to introduce some kind of multicasting so as data are sent to all stations only once from the CRR's side. If no multicasting is implemented, the amount of data entering the interconnection network would be the size of 280000*3000 = 840 Mbits, a number which is not very practical and would mean in most cases a long update delay.

The multicasting solution of course is UDP-based and depends on the availability of proper hardware from the ISP side. We modified the initial, Round Robin approach for the bus station updates by clustering stations into geographic groups, each one controlled by a modem server. Each member of a group is fed with new data at the same time and the Round Robin approach iterates among groups. This hierarchy can be repeated, extended or combined with a multicast approach. In any way, there are many ways to implement the interconnection network, by combining various transmission technologies, many of which have been already tested or applied to similar problems (such as infokiosks). The deployment of such a network or the reuse of a previously installed one is beyond the scope of this project.

LRR → CRR

The records that are sent from LRR to CRR are of the following format:

BusID	StationID	Arrival Time (Unix timestamp)
16 bits	16 bits	64 bits

The transmission of the data from LRR to CRR is made using the same TCP/IP connection that is created for the CRR->LRR communication. The difference is that the upload of data to CRR is much simpler and is made asynchronously. When a bus arrival is registered in the LRR, the corresponding event record is immediately sent to CRR. Assuming the extreme situation where all N stations send an arrival event to CRR at the same time, then the amount of data that is sent is $96N$ bits. For $N = 3000$, we have 288 Kbits. Therefore, the required upload bandwidth for LRR and the respective download bandwidth for the CRR are minimal.

All modules in the LRR-CRR communication were programmed with C++ (GNU compiler), except the routing algorithm, which was implemmented in Java.

4.3 Bus Identification Subsystem

This subsystem is responsible for supplying BUPUS with the current position of each bus. Two independent hardware devices are enough for this purpose. The first device, an RF transmitter, is placed inside each of bus. The second device, an RF receiver, is housed within each bus stop. Each time a bus passes by a bus stop, the unique ID of the bus is recorded along with a timestamp indicating the moment this happened. This process involves the wireless transmission of each bus's identification number. For a medium-to-large bus network, 12

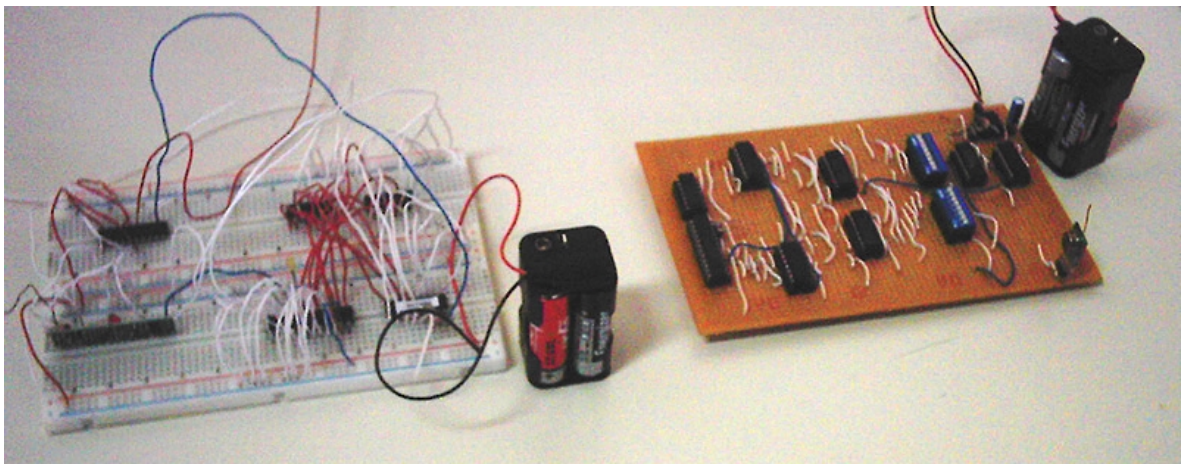


Fig. 7. The prototype transmitter stripboard (right) and receiver bredboard (left)

bits would be enough to identify each individual bus (4096 combinations), assuming a single frequency communication channel. Taking into account that:

- The frequency these transmissions occur is small (the frequency width which buses arrive at bus stops)
- the necessary range so that id recording can take place is at most 15 metres
- We need cost-friendly, robust devices with reconfigurable Ids the transmission end
- Commercial RFID circuits are sold in good prices in very large quantities and cannot be reconfigured or deployed at an electro-magnetically noisy environment

we decided to proceed into building the necessary circuits which meet the above requirements on our own. It should be noted that commercial, general purpose devices (non-ID specific) surpass the total cost by at least 1500% while offering at the same time more features, nonetheless useless for our strictly defined system.

After conducting a survey on cost-friendly wireless transceivers we concluded in two solutions:

- TWS-434 / RWS-434 by Reynolds Electronics
- TX-99 / RE-99 by Ming

The specifications of both solutions were not very different but the reviews across the Internet convinced us to use the Reynolds' pair of transceivers. The image in Fig. 7 shows the implemented prototype circuits that were used in the tests.

4.3.1 RFID Transmitter

The TWS434-based transmitter, has the advantage of using an ISM-band frequency. The device, which has to be incorporated into every network bus, should allow someone to easily form the ID, which pertains to a specific bus. In the prototype, this feature is implemented using on-board dip-switches.

The design of the transmitter had to take into account the following requirements and issues:

- The 12 identification bits needed to be converted from a parallel to a serial form in order to be transmitted. Commercial parallel-to-serial encoder ICs only have an 8bit input.

- Each burst of transmitted ID bits should be followed by a large gap of silence in order to allow other buses in the vicinity to register to the Smart Bus Stop.
- A mechanism for ensuring the start/end-of burst transmission was required, as well as for intercepting inevitable collisions.

Our RFID transmitter, whose diagram is presented in Fig. 8, addresses all the above issues and meets the link range requirement, while maintaining a very low cost for this type of application. The 16bit ID is split in two bytes and sequentially passed to the Holtek HT640 serial encoder. The timing/multiplexing circuit also triggers the encoder to send bursts of the two bytes and disables transmission (zero output) for the rest of the ID transmission cycle. This ensures that in each cycle, the 16bit code is followed by a properly large gap (multiple of the transmit-enable phase), which minimizes the probability of a collision happening.

The Holtek encoder is capable of encoding 8 data bits and 10 address bits. In our implementation the address bits are always set to zero but can be alternatively set to a non-trivial pattern for extra collision avoidance. In a case of aliasing of the transmitted ID burst, the decoder would not detect a correct address in the majority of the cases and, as a result, would not produce a new output byte, handling the collision simply and effectively. The use of the matched encoder/decoder pair tackles the collision interception and the parallel-to-serial-to-parallel conversion effectively and transparently. The "transmission enable" (TE) input of the encoder guarantees that no unnecessary transmissions take place.

Due to the random activation of each RF Transmitter board, the timing deviations due to analog components and the probability two buses to arrive at a given bus stop simultaneously, the chances of a bus passing by a bus stop while the system stays uninformed, at that moment or later on, are negligible. Even in this case, the bus will be eventually registered with the next stop in line and its status updated.

4.3.2 RFID Receiver / Time Stamp Generation

The RF Receiver constitutes the second part of the BIS Subsystem and is placed (housed) in each SBS. Its aim is the recognition of each bus via the bus ID that any RF Transmitter emits continuously. The SBS reads the output of the receiver through the RS-232 port and registers the bus encounter as the time the first encounter of a valid two-byte ID is encountered.

For the reception of data from the bus, each station is provided with the RWS-434 radio

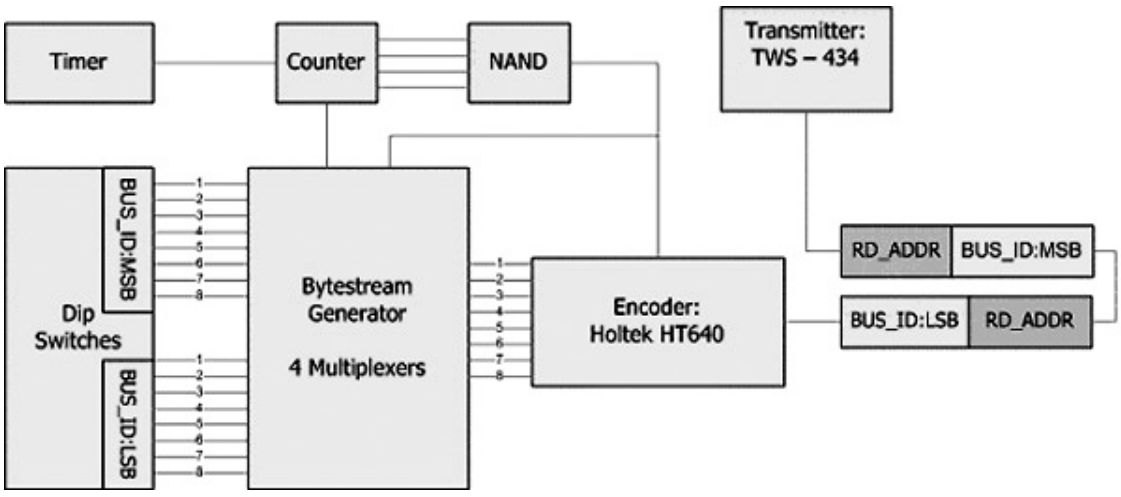


Fig. 8. The transmitter block diagram.

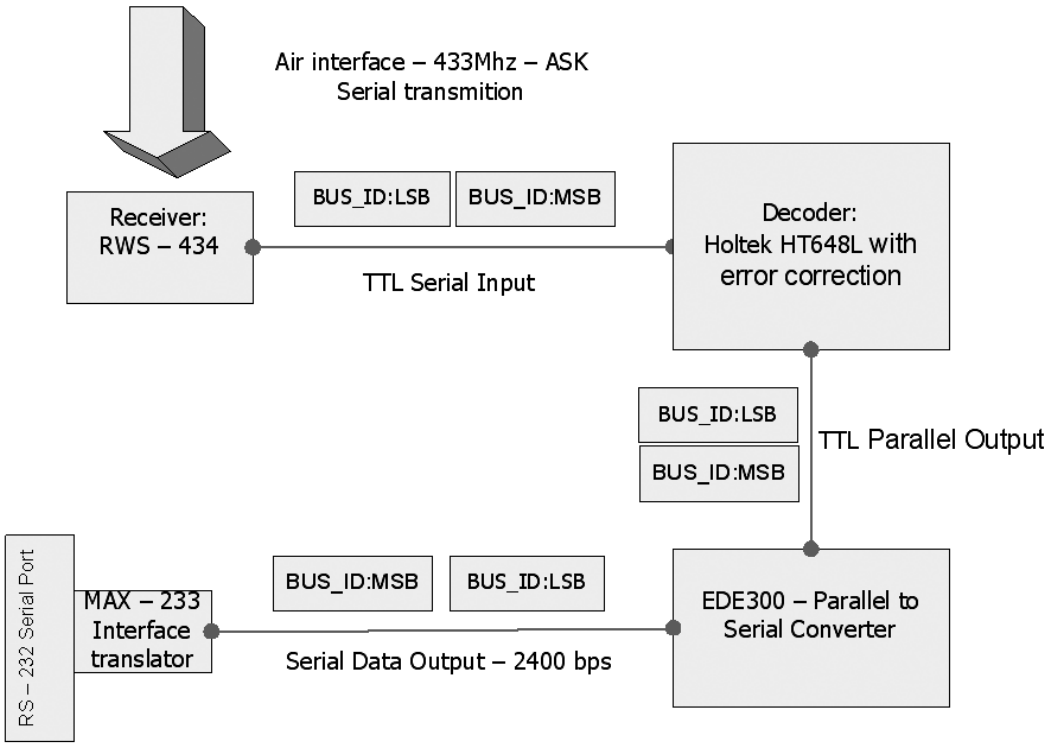


Fig. 9. The block diagram of the RF Receiver.

module from the Reynolds Electronics, which includes an adaptive circuit in order to implement the aerial data transfer. This radio module uses amplitude modulation (ASK) in band 433.92MHz and has a sensitivity of -106dbm, consequently fulfilling our needs in terms of signal quality and coverage. In order for the reception of each byte to be guaranteed, the decoder HT648L is also used with the radio receiver. This IC is capable of asynchronously decoding information that consists of 10 address bits and 8 data bits and validating the correct reception of a byte. If no error or unmatched codes are found during the reception of a byte (redundancy error detection), this byte is transferred to the output pins. Figure 9 shows the block diagram of the receiver.

The exit of the decoder passes through a multiplexer which zeros the output when the valid transmission (VT) signal of the decoder is off. This ensures that no garbage is sent to the PC and provides a separation mark to ease the ID recognition. Eventually, the 8 bits (each byte of the ID) are propagated to the EDE300 parallel-to-serial converter and the MAX233 TTL-to-RS232 level converter.

The ID recognition and time stamp marking is performed inside the SBS Support machine, which the receiver is connected with. We have developed a small program, which reads and processes the incoming bytes from the serial port. The asynchronous sampling of the EDE300 is fast enough (when compared with the pulse of the RF transmitter) so as to interpret each of the 2 ID bytes multiple times, something which must also be dealt with when receiving the RS-232 data. Because of this, the two bytes that constitute a bus ID must be different which results in reducing the effective usable bits to 15. The general idea of the algorithm sums up to the following:

The program is blocked waiting for the first byte. When that is received, an alarm signal is set with its value a little larger than the RF transmitter pulse. Our tested value that complies with our design is 1 sec. During this time a second byte different from the first byte is awaited. If

the time passes without this happening the reception is considered invalid and execution is redirected to the beginning. If a successful transmission is signaled the complete ID and its timestamp are forwarded to a queue which serves the purpose of filtering such duplicate messages which are possible when traffic is encountered in a bus stop. This may happed if for various reasons (such as traffic) the bus is forced to remain in the proximity of the smart bus stop for a while.

The construction of both the RF transmitter and receiver was first made on a breadboard for experimentation purposes. After the confirmation of a working status we proceeded into building both devices in a veroboard (stripboard), which can be more safely carried for outdoor experiments. An effort will be made for final prototype to implement the circuits on printed boards, which are lighter and more compact. All code for the communication with the receiver and the formation of the events was implemented in C/C++ (GNU compiler).

4.3.3 RFID Transaction Layers

Figure 10 summarizes the overall layered architecture of the RF identification transaction, based on the detailed analysis presented in sections 4.3.1 and 4.3.2.

4.3.4 Housing and Cost

Both the RF transmitter and the RF receiver were powered by 4-pack AA size batteries during the tests. But eventually, a non-interruptible power source could and should be used for both the transmitter (from the bus) and the receiver (from the serial cable of the SBS support machine).

The external antennae of the RF modules do not need be very long, as the copper of the

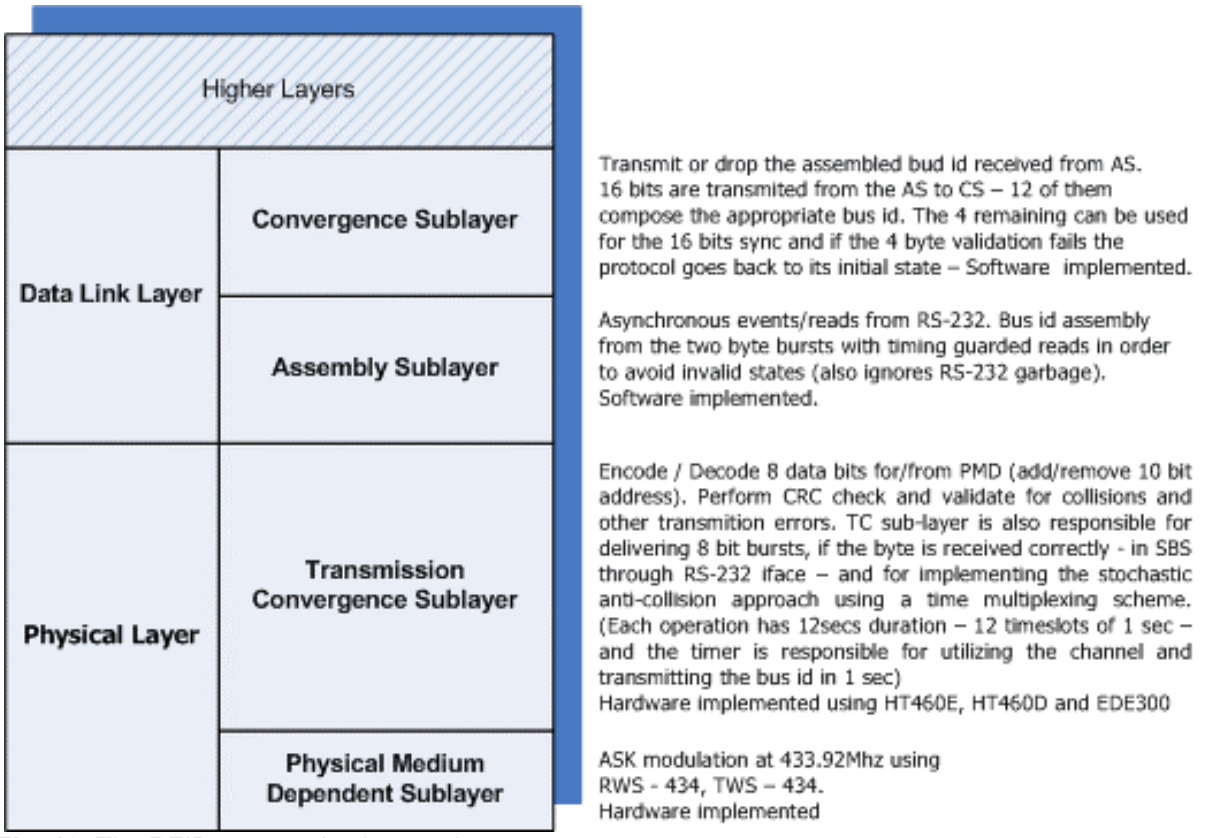


Fig. 10. The RFID communication stack

board is part of the waveguide.

Even with part pricing for small quantity orders, like ours, the cost of the RF system is very small: $18.65 for the RF transmitter and $36 for the receiver/serial communication module.

4.4 BUPUS Service Provider and Information Support Subsystem Machine

This machine will be unification point of the other three subsystems (Service Accessibility Subsystem, Bus Identification Subsystem, Bus Route Database Replication Subsystem). Each bus stop supported by the system must have this machine installed. The inputs/outputs required inside are the following:

- 56k Modem for PPP connection with CRR
- Serial port (RS-232) where RF receiver is attached.
- 802.11b Access Point running DHCP, DNS services.

Debian GNU/Linux has been chosen. DHCP v2.0 daemon is used to automatically assign IP addresses to the users of BUPUS. 802.11b wireless clients containing Prism v2/2.5/3 chips have been used due to their price and ability to be converted to software access points with the hostap linux driver.

4.5 Verification and Testing

During the various stages of individual component design and incremental integration, tests were performed to ensure that the prototype and the algorithms meet the requirements.

4.5.1 Service Accessibility Subsystem

The client side of the subsystem (the application) was tested using the J2ME Wireless Toolkit 2.2. When the development of the subsystem's software was complete, a real PDA was used in order to check the subsystem's usability in the real world. For this purpose we used a Compaq® iPAQ 3900 series, with Microsoft® Pocket PC™ and with the wireless card of AmbiCom®, model WL1100-PC/PCI. As mentioned above, we stuck up on the lack of a Java Mobile Edition compatible virtual machine. There was only a J2SE compatible VM (JeodeRuntime), which could not be used as-it-is to run our application. The problem was solved using a GPL licensed Java package, the me4se, which translates calls for ME to calls for SE. Until a solution was found, we thought of rebuilding the application with another language (e.g. embedded Visual C++). Eventually, we turned down these options, because of the portability that this language provides, as mentioned before (Linux,palmOS and symbian support j2me applications), even with the utilization of additional packages.

As for the tests, to summarize, the information provided by the server side of the SAS and the projection of this information through the application to the user, behaved as planned. The drawback of using Java for the application with the need of the extra package is the delay introduced during the initialization of the application. When the initialization phase is completed, there is no other, visible to the user, delay.

Fig. 11. Snapshot of the RFID outdoor experiment

4.5.2 RF Modules

One of the major concerns for the transmitter and receiver was whether the RF pair could perform well at a large distance. In order to verify the manufacture's specifications, we felt it was necessary to conduct our own tests of the range of these products. The results for these tests are explained below.

We tested the receiver and transmitter in three distinct settings, one in the lab, one in an electro-magnetically noisy environment (computer lab with full of operating PCs and people using mobile phones) and one outdoors (urban environment, rush hour – Fig.11). Neither the transmitter nor the receiver had any antenna attached, except from the 12cm board copper strip the antenna pins were soldered to. The average measured distance of successful ID transmission has been **15.4m** with a deviation of 2.4m.

4.5.3 Queries Testing with Mock-up Scenario

In order to be able to perform queries for testing, data from 371 actual bus stops taken from the local public transport authority and belonging to 12 routes (both directions) in the region of Attiki (Athens urban area and surroundings) were inserted in the databases of the LRR and CRR. From these data, 381 weights were calculated, given an initial distribution of bus arrival times (simulated).

- For 52 active buses (avg 5 buses per route) 433 records were generated using the edge-method and 1371 using the bus-station pair method (see section 4.2).
- for 62 active buses 443 records were generated using the edge method and 1782 using the bus-station pair method.

Routing queries performed on these data yielded results that were correct, at least as far as human can verify from repeating the similar procedure by hand.

As far as the query response latency is concerned, running various queries on the LAN-interconnected CRR and LRR systems that we used for testing purposes, resulted in insignificant delays, as the system responded immediately.

5 Summary

The BUPUS project proposed and implemented a methodology and related systems for the modification of bus stops to active wireless networking points for the provision of general and location-sensitive traffic information to pedestrians, moving in the streets of a city. The project involved the design and integration of many interconnecting modules, from the central route repository system to 802.11 and RF terminal devices for the end-user and bus location identification respectively. In this process, different transmission protocols and techniques were used and many algorithms applied and extended (more than 4000 lines of code written so far). As is explained in section 5.2, various alternative design and platform integrations considerations can be made, based on the same concept.

5.1 Current Project Status

In the short time the BUPUS team had for the execution of the heavy scheduled tasks described in the interim report, the system was brought to a fully operational state, with all modules implemented. We were able to individually test the modules, the procedures involved in the intercommunication of the various parts and the system as a functional entity.

Optimizations on code fragments and algorithms are constantly made as the team seeks to minimize traffic and processing burden, without of course altering the architecture of the BUPUS system. Small details, like a scrolling graphical map with color-coded lines interface for the PDA and printed circuits for the RF modules are under way.

A full-scale demonstration, involving multiple moving vehicles has not been performed yet but this is planned to take place during the summer for our University community, regardless of the opportunity for a final presentation for the CSIDC 2005 competition. Unfortunately, a real deployment test of the prototype, involving buses and bus stop installations, is out of the financial and time limits of the project.

5.2 Future Work

The concept behind the BUPUS system allows for many alternative design considerations. An important one is the integration of the client-side services into mobile telephony services. This of course requires the collaboration of the University with one of the cellular telephony providers, a prospect we are looking forward.

We have considered several enhancements, which could further improve the effectiveness and feasibility of our system. One of them is the adaptation of a distributed replication system, which renders the central database (CRR) unnecessary. Since the bus stops in a large city are near each other (<200 m) in general, their wireless communication through ISM band is possible, eliminating the costly, wired networking. Some cities also have an aerial or telephony-based (ISDN-mostly) system for interconnecting fixed information access points in general. A reasonable follow-up study could focus on the utilization of existing infrastructure to host the SBS-to-CRR interconnection network and the unification of the services provided.

From a different perspective, The BUPUS network demonstrates the use of bus stops as 802.11 access points. Considering the distribution of bus stops on the main streets of a city, this opens up a new opportunity to provide many public information services through an access-free network. Alternatively, a service subscription system could be incorporated, which would require the payment of a small amount of money such as 4 Euro per year. The profit made could be used for the general maintenance of the system as well as for covering the expenses of the initial system deployment. The small cost per person would not deter the people from using it.

6 References

- ME4SE - http://kobjects.sourceforge.net/me4se/
- Java 2 Platform, Micro Edition (J2ME) - http://java.sun.com/j2me/
- J2ME Wireless Toolkit - http://java.sun.com/products/j2mewtoolkit/index.html
- W. Richard Stevens, **Advanced Programming in the Unix Environment**. Addison Wesley, 1993.
- I. Sommerville, **Software Engineering**, 6th Edition. Addison Wesley, 2001.
- S. Donaldson, Stanley Siegel, **Successful Software Development**, 2nd Edition. Prentice Hall, 2001.
- M. Mano, **Digital Design**, 3rd Edition. Prentice Hall, 2002.
- T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, **Introduction to Algorithms,** 2nd Edition. MIT Press, 2001.
- MySQL – http://www.mysql.com
- A. S. Tanenbaum, **Computer Networks**, 4th edition, Prentice Hall, 2003
- B. Stroustrup, **The C++ Programming Language**, Addison-Wesley Professional, 1997
- K. Finkenzeller, RFID Handbook, 2nd edition, Wiley, 2003

Appendix – Abbreviations

Abbreviation	Meaning
LRR	Local Route Repository
CRR	Central Route Repository
SBS	Smart Bus Stop
SAS	Service Accessibility System
BIS	Bus Identification System
BLN	Bus Line Number