Remote Fleet Management for Police Cruisers

Sung Yun Kim, Kaitlin Wilson-Remmer, Andrew L. Kun, Member, IEEE,
W. Thomas Miller, III, Member, IEEE

Abstract—We describe two prototype Remote Fleet Management software modules for police cruisers. One of the modules is used for automatic vehicle location (AVL) the other is used for cruiser status monitoring. The software modules leverage the existing Project54 infrastructure that integrates electronic devices in cruisers and connects cruisers to each other and to headquarters. This infrastructure allows client-server connections over 802.11 wireless networks as well as over police radio networks. The AVL module transmits GPS information to headquarters over the radio network. In order not to overwhelm the radio network, heuristic rules are used to decide when to transmit data and when to discard it. The AVL module was successfully tested in the field. The status monitoring module is designed to use the On-Board Diagnostics II (OBD II) standard to connect to the vehicle’s internal bus. The prototype was successfully tested in laboratory conditions with simulated OBD II data.

Index Terms— Automatic vehicle location, in-car device integration, on-board diagnostic, remote fleet management

I. INTRODUCTION

The Consolidated Advanced Technologies for Law Enforcement Program (CAT Program) is a collaborative effort between the University of New Hampshire and the New Hampshire Department of Safety (NHDS). The CAT program has developed an in-vehicle system, called the Project54 system that integrates general purpose computing facilities, voice and data radio communications, and special purpose devices such as radar, lights and siren, video units, and GPS units [1]. The system software provides centralized control of the local devices. Multiple modes of user interaction have been provided for all functionality including voice-command input and feedback, touch screen LCD display and traditional keyboard and mouse. The system provides access to both local and remote data in support of typical public-safety applications such as license and registration checks, computer aided dispatch, vehicle navigation, reports/forms entry and so forth.

Manuscript received December 15, 2004. This work was supported by the U.S. Department of Justice under grant 2003CKWX0151.

Sung Yun Kim is with the Electrical and Computer Engineering Department, University of New Hampshire, Durham, NH 03824 USA (phone: 603-862-1357; fax: 603-862-1832; e-mail: sykim@cisunix.unh.edu).
Kaitlin Wilson-Remmer is with the Electrical and Computer Engineering Department, University of New Hampshire, Durham, NH 03824 USA.
Andrew L. Kun is with the Electrical and Computer Engineering Department, University of New Hampshire, Durham, NH 03824 USA.
W. Thomas Miller, III is with the Electrical and Computer Engineering Department, University of New Hampshire, Durham, NH 03824 USA.

The data system architecture being deployed in New Hampshire is being optimized to provide maximal functionality considering the low-bandwidth and possibly intermittent wireless data channels typically encountered in public-safety operating environments [2]. As of December, 2004 the Project54 system has been deployed in over 240 police cruisers in the state of New Hampshire.

In this paper, we explore Remote Fleet Management (RFM) options for the Project54 system. RFM is an increasingly popular technology that monitors a group of vehicles wirelessly. RFM is used for many purposes: GPS tracking (also referred to as Automatic Vehicle Location (AVL)), scheduling maintenance, remotely diagnosing vehicle problems and reporting the condition of vehicles. We expect that this technology will provide increased officer safety, an efficient work environment, and substantial financial savings for law enforcement fleets. AVL increases the safety of police officers with the ability to determine the location of the vehicle while it is on the move. Efficiency would be improved since potential damage can be detected and fixed before an actual breakdown of a vehicle. Costs would be reduced since the labor required to collect and update vehicle status for police cruisers would be reduced.

We created two prototype RFM software modules by leveraging the existing Project54 infrastructure that integrates devices in cruisers and that connects cruisers to each other and headquarters. One monitors vehicle location using GPS data. The other monitors the status of the vehicle and maintenance needs using a diagnostic tool based on the On-Board Diagnostics II (OBD-II) standard. Each vehicle in the fleet that uses the RFM client would communicate RFM data to a centralized server via digital radio or using an 802.11 wireless connection. One has to take into account that the digital radio connection is a low-bandwidth connection (in New Hampshire the State Police use a 9600 bps connection). Therefore, only high priority information would be transmitted immediately using digital radio, whereas less urgent data could be sent at a convenient time on an 802.11 wireless network. For example, if a cruiser is pursuing another car it may be important to update the server with the cruiser’s location every few seconds. However, if the cruiser is parked in a lot while the officer is in his office, there is no need to send updates frequently.

We will discuss the prototype RFM software tested, as well as long-term goals for the eventual RFM modules. Section 2 contains a brief background on RFM, distributed software objects and the OBD-II technology. Section 3 consists of a description of Project54 and in Section 4 we discuss
II. BACKGROUND

A. Related Research

The Intelligent Electronic Systems (IES) program in Mississippi State University designed a prototype wireless network technology to monitor real-time vehicle performance. The entire Mississippi State University bus system will adopt this technology for optimization of the fleet [3]. The PowerTools [4] software architecture was developed in conjunction with Volvo. It collects vehicle data when the vehicle is moving and allows the mechanic to become a remote worker diagnosing the vehicle via the Internet. Our proposed system is similar to these systems and builds on the Project54 infrastructure.

In the Project54 system architecture COM is used to allow individual modules to interact using a one-to-one messaging scheme [5]. However, there are many other ways this could be accomplished. In [5] Reilly et al use Jini middleware technology [7] to create the needed distributed architecture, used to remotely manage and provide application services for in-vehicle computers and handheld devices. Their target was the EmergeITS project which employs an in-vehicle telematics system for applications used in emergency services, particularly fire services. In a related project, Prasad et al developed a truck-fleet application, in which through handheld devices, each truck in the fleet has the ability to communicate to a depot or any of the trucks [8]. Simple Object Access Protocol was used so that heterogeneous devices are able to communicate. Coulson et al [9] worked on the development of a prototype architecture (OpenORB v2) in order to achieve both deployment-time configurability and run-time reconfigurability. This architecture is an enhancement of the core of Microsoft’s COM. Potter et al [10] implemented a Distributed COM-based interoperable framework for the integration of forest decision support applications.

B. OBD-II

Auto manufacturers started incorporating OBD-II devices into vehicles as early as 1994. All cars manufactured since 1996 have OBD-II diagnostic systems. There are currently three basic OBD-II protocols in use, each with minor variations in the communication pattern between the on-board diagnostic computer and the off-board scanner. Even though there are three different OBD-II electrical connection protocols, all OBD-II compliant diagnostic tools interact with the outside world using the same set of queries and responses [11].

Standard OBD-II information does not contain such important information as mileage, fuel level, and battery life. To acquire this information our RFM module would have to communicate with the vehicle’s Electronic Control Unit (ECU) [12]. There is no standard ECU programming so that each manufacturer has a different data format. Therefore, we would need to acquire this information directly from the manufacturer.

III. Project54 Overview

A. Hardware Integration Standard

At the center of the in-car Project54 system, shown in Fig. 1, is the embedded PC. The bottom part of the figure shows the devices that connect to the PC through the Intelligent Transportation Systems Data Bus (IDB): the lights and siren, the radar, the radio, the video recorder, the GPS unit, the barcode scanner, and the push-to-talk button (used to signal that speech recognition should be performed). The Common IDB Interface (CIDBI) is used to connect all of these devices to the IDB [13]. The computer has only one connection to the bus (requiring only a single serial port on the computer) and can control up to 30 devices connected to the bus throughout the vehicle. The data bus is constructed using standard CAT-5 networking cables identical to those used to create networks of computers in offices.

The top part of Fig. 1 shows devices that connect directly to the PC: the system disk, the keyboard and mouse, the microphone and speakers, the LCD touch screen, and the wireless network card. The 802.11 network card provides wireless connectivity to a local area network.

B. Software Integration Standard

Each device in the cruiser, such as the radar, has a corresponding software application used in the Project54 software system to control it. The Project54 in-cruiser software system runs on the embedded PC. It uses Microsoft Component Object Model (COM) objects, and was designed to be fully modular so that the software in each police cruiser can be easily adapted to specific sets of equipment [2]. Application modules communicate with one another through a simple and efficient messaging system. At the center of the system is the Application Manager (Fig. 2). The Application Manager implements the message coordinator object to receive and route the inter-application messages. Each application in the
Project54 software system implements a message handler object to receive messages routed by the Application Manager. This messaging system supports the sharing of information between modules to provide the advantages of an integrated system.

Fig. 2 Software architecture

**C. Project54 Messaging**

Applications share information based on a simple application-to-application messaging model of the form: `Message(source, destination, message_id, message_text)` [5]. All four elements of the message are null-terminated binary sequences. All objects in our current system exchange text messages – that is null terminated strings (we use wide-character format strings). This simplifies debugging in a system that contains a large number of objects written by many developers. However, the elements of the message can be any null-terminated binary sequence.

Applications address the text messages to other applications via the destination character string, and provide a return address in the source character string. The Application Manager uses the source and destination strings to deliver messages to the correct applications. The `message_id` string provides an application-defined message qualifier to help simplify message sorting and processing inside applications, and the `message_text` string contains the actual message. For example when the lights application needs to let another application, called `pscreen`, know that the cruiser’s strobe lights have been turned on, it would send the message: `Message(lights, pscreen, any_id, L"STATUS STROBES ON")`.

**D. Client-Server Messaging Using 802.11**

The messaging scheme employed by the Project54 software also allows for communication between distributed Project54 applications. In [5] Pelhe et al described using this method to accomplish the successful communication between a handheld device, one in-car Project54 software system communicates with one handheld. This allows for each application, both in the in-car and in the handheld systems, to have a unique application name. For the applications to communicate to each other across a network, a proxy application is employed (Fig. 2). The objective of the proxy is to render the underlying communication network invisible to components within the Project54 system. This is accomplished through aliasing, where the proxy is known to the Application Manager not only as its own name, but also as the names of remote applications.

The in-car Project54 Proxy Application supports messaging with remote applications (or remote Application Managers) via UDP/IP network packets. The UDP protocol was chosen because it does not require a permanent connection between the distributed components. This is important in our environment because we expect that the connection between the in-car embedded computer and a computer outside of the car may break up from time to time. Using the TCP protocol would require rebuilding the software structures each time a connection is lost and re-acquired.

**IV. INFORMATION MANAGEMENT IN PROJECT54**

**A. NHDS Wireless Network**

The CAT program supported the NHDS in the building of a statewide wireless data access system for troopers in the NH State Police and the NH Highway Patrol (Fig. 3) [2]. The statewide data access system employs the Project 25 digital radio standard, which handles both voice and data communications. In addition to the digital radio, a second type of wireless data communication, 802.11b, can be utilized as part of the system. Using high bandwidth wireless networking access points, such as at New Hampshire State Police troop stations and selected NH Department of Transportation fuel stations, the cruiser establishes a temporary 802.11b wireless network link to download a software upgrade or upload in-cruiser database files. By establishing two types of wireless connections, we are afforded some flexibility in how and when any stored data is transmitted from the cruiser.

At the server side, the Computer Aided Dispatch (CAD) system automates call taking and dispatching functions. The CAD system is operational state-wide, and currently handling the dispatch functions for the NH State Police, NH Highway Patrol and multiple NH local police agencies.
B. Client-Server Messaging over the Radio Network

Important to the prototype of the proposed RFM mechanism is a method of communication between the in-cruiser application and the server over a radio network. As in the case with the handheld devices, this was achieved by applying the same inter-application messaging scheme used within the cruiser to the cruiser-to-server messaging. In the case of RFM communications, the messages containing the RFM data are to be transmitted across the digital radio link. To make this as efficient as possible, we employ a proxy application with added functionality specific for the radio, as shown in Fig. 4.

![Diagram of communication paths](image)

Fig. 4 Overview of communication paths

We named the proxy application that transmits data over the digital radio P25Proxy, after the Project 25 digital radio standard. Each time the P25Proxy application receives a message destined for a server application, the P25Proxy adds the message to a queue. The main benefit of the P25Proxy is that it interprets Radio Control Protocol (RCP) feedback from the radio to determine whether the next message in the queue should be sent, discarded or held in a wait state. The P25Proxy currently supports five RCP packet types. These are: radio power has been turned on, data service is not registered, packet loss has occurred, data service is not available, and data service is available. Based on the knowledge that the P25Proxy has of the state of the radio, the P25Proxy either transmits, discards or delays the message. By taking advantage of the RCP information, the system does not waste time attempting to send messages when it is known that messages cannot currently be sent (e.g. the radio is powered off).

The P25Proxy employs UDP/IP to transmit the message to the server. The server software follows the Project54 style, with an Application Manager and separate applications for different services and also utilizes COM objects. The ServerProxy is the proxy application located at the server that receives the messages from the in-cruiser applications. Before the ServerProxy passes along the messages to the appropriate server application, it must convert the UDP/IP packets received into the correct Project54 Message format. The only difference between how this is done in the ServerProxy and a proxy in the cruiser, is in the treatment of the source field. Because multiple cars with the same application names will be communicating to the server, something extra needs to be added to the remote application name so that each car has a unique application name. The ServerProxy accomplishes this by adding on the IP address and port number from the remote application. Thus, the server applications see the source application in the following format: dclient:ip.ip.ip.ip:port, whereas the actual source name is: dclient. For example, the source name “setup” may become “setup:10.1.9.54:1234”.

Upon receipt of an RFM report, software at the server writes the report to a text file which is named after the IP address of the cruiser. This simple naming convention allows a basic method for keeping RFM data organized by vehicle. Each day a new folder is created and named for the current date. It is in this up-to-date folder that the text files are created, with names that correspond to the cruisers’ IP addresses, so that RFM information can be stored in an organized manner. For example, inside a folder named Nov_19_04, there may be two text files, one called IP10_1_54_2.txt and the other IP10_1_54_1.txt, containing the RFM reports generated by two cars on November 19, 2004.

V. AVL Prototype Software

AVL systems have the ability to track the position of individual vehicles and to relay that data back to a remote location that can store or better utilize the information. The implementation of an AVL system within a police department, to automatically log and keep track of the location of each cruiser, is of great benefit to both the department, and the public. Not only can this provide critical information when locating an officer in danger, it can also be utilized to aid human dispatchers, or computer-aided dispatch (CAD) systems, in vehicle deployment. The AVL data can be used to estimate which cruisers are closest to an incident and the decision about dispatching a particular cruiser can be based on
this estimate.

An important area of investigation is the determination of the frequency at which the AVL data is relayed back to headquarters, since the limited bandwidth of the radio must be shared amongst the AVL and other systems in multiple cruisers. Due to limitations of the communications system used by the New Hampshire State Police, voice traffic cannot be transmitted at the same time as data traffic, and data can only be sent by one cruiser at a time. This implies that making AVL reports too frequent will overload the system. However, sparse AVL reports provide little usefulness.

The current AVL code contained in the P54 software follows a minimum time (min_time), maximum time (max_time) and maximum distance (max_distance), guideline. When the AVL code within the P54 software receives data from the GPS receiver, it determines how much time has passed since it relayed AVL data to headquarters. If this time is greater than max_time data is transmitted. If the time is less than max_time but greater than min_time the code checks the distance the cruiser has traveled to determine whether the data should be discarded or transmitted. If the arc length between the last two GPS readings is greater than max_distance the data will be transmitted, otherwise it will be discarded.

To test the AVL system, the Project54 AVL application was added to the Project54 software system on an embedded PC in an active cruiser. The AVL server code was located at the headquarters in Concord, NH. The radio in the vehicle was used to transmit the AVL data from the car to the server. Therefore, for this test, the client application included in Fig. 4, dclient, would be the AVL application. Similarly, the service provided by the server in Fig. 4, datasvr, would be the AVL service that stores the AVL information it receives from the remote clients. About every 15 seconds the AVL application receives data from the GPS unit. The min_time was set to 7 minutes, the max_time was set to 10 minutes and the max_distance was set to 4 nautical miles.

The resulting files were downloaded from the server daily. Table 1 shows about 90 minutes worth of results collected from the cruiser on the road. The GPS receiver used during this testing period output GPRMC sentences: the Recommended Minimum Specific GPS/Transit Data. This sentence type contains time, longitude, latitude, speed, date, magnetic variation and a navigation receiver warning. Due to the limited scope of this trial, only the longitude, latitude and speed were analyzed.

In Table 1, we see that the transmissions 1 through 8 happen at approximately 10 minute intervals. This is because the cruiser appears to have been stationary during the time period they were transmitted (the speed over ground was practically zero) and thus the cruiser did not travel more than 4 nautical miles (4 nautical miles = 1.15 miles). Transmissions 9 through 11 happen at approximately 7 minute intervals. We can see that the corresponding cruiser speeds were between 40 and 53 knots (a knot is a nautical mile per hour). This is fast enough for the cruiser to travel a distance that exceeds 4 nautical miles (max_distance) over 7 minutes (min_time) and the data will be transmitted.

<table>
<thead>
<tr>
<th>No.</th>
<th>Time (hh:mm:ss)</th>
<th>Latitude (deg:min)</th>
<th>Longitude (deg:min)</th>
<th>Speed over ground (knots)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>07:07:32</td>
<td>43:08 N</td>
<td>70:52 W</td>
<td>0.116</td>
</tr>
<tr>
<td>2</td>
<td>07:17:47</td>
<td>43:08 N</td>
<td>70:52 W</td>
<td>0.135</td>
</tr>
<tr>
<td>3</td>
<td>07:28:02</td>
<td>43:08 N</td>
<td>70:52 W</td>
<td>0.110</td>
</tr>
<tr>
<td>4</td>
<td>07:38:07</td>
<td>43:08 N</td>
<td>70:52 W</td>
<td>0.127</td>
</tr>
<tr>
<td>5</td>
<td>07:48:19</td>
<td>43:08 N</td>
<td>70:52 W</td>
<td>0.223</td>
</tr>
<tr>
<td>6</td>
<td>07:58:35</td>
<td>43:08 N</td>
<td>70:52 W</td>
<td>0.111</td>
</tr>
<tr>
<td>7</td>
<td>08:08:55</td>
<td>43:08 N</td>
<td>70:52 W</td>
<td>0.107</td>
</tr>
<tr>
<td>8</td>
<td>08:18:04</td>
<td>43:08 N</td>
<td>70:57 W</td>
<td>52.109</td>
</tr>
<tr>
<td>9</td>
<td>08:25:32</td>
<td>43:09 N</td>
<td>71:04 W</td>
<td>53.331</td>
</tr>
<tr>
<td>10</td>
<td>08:32:38</td>
<td>43:12 N</td>
<td>71:10 W</td>
<td>40.708</td>
</tr>
</tbody>
</table>

VI. OBD-II PROTOTYPE SOFTWARE

In order to expand our RFM capabilities we are in the process of creating client-side and server-side software for diagnostic purposes. In this section we describe the prototype diagnostic software that we tested in laboratory conditions.

Our client-side diagnostic module is called the OBD-II Scan Application. Once deployed in a cruiser, this application will use an OBD-II scan tool to access the vehicles internal bus and acquire information about vehicle speed, vehicle throttle position, engine intake air temperature, engine RPM, engine temperature and engine load value. For test purposes this application currently creates a random sequence of numbers between 80 and 117, representing the vehicle speed. A new number is generated approximately every second. Fig. 5 shows a sequence of numbers generated over a period of about 30 seconds.

In creating the prototype application we envisioned the following scenario. If a cruiser exceeds the velocity of 100 miles/hour for more than five seconds this means that the cruiser is pursuing a vehicle. If this is the case, headquarters should be notified. Therefore, the OBD-II Scan Application keeps track of the speeds it obtained (generated randomly in our test case) and looks for periods of five seconds or more during which the speed was consistently over 100 miles/hour. When a sequence like this is encountered the OBD-II Scan application sends a message to the server-side application. This is shown in Fig. 5 using the “speed alert” point in the graph.

In order to test the OBD-II Scan Application in the field (using the police radio) we will have to decide when to transmit data and when to discard it, based on heuristic rules similar to those being tested in the AVL prototype software. The “five-second rule” described in the previous paragraph is our first attempt at formulating a reasonable rule for when to transmit.
We implemented the OBD-II Scan Application and its server side counterpart, the OBD-II Headquarters Application and tested them in the setup shown in Fig. 4. Therefore, for this test, the client application included in Fig. 4, delient, would be the OBD-II Scan Application. Similarly, the service provided by the server in Fig. 4, datasrvr, would be the OBD-II Headquarters Application. The two applications ran on two PC computers that were connected with an Ethernet link. This made the test simpler than in the AVL case, since we did not need to use the radio link.

We are in the process of modifying the test version of the OBD-II Scan Application in order to query the data bus of a Chevrolet Tahoe SUV. We are using the Advanced Vehicle Technologies 718 (AVT-718) OBD-II scan tool for this purpose. The AVT-718 supports communication with the host computer using an RS-232 serial communications link. This makes it simple to connect it to a CIDBI which provides an Ethernet link. Consequently, the integration of the AVT-718 unit into the Project54 communication bus only requires matching the baud rate of the CIDBI and the AVT-718.

VII. CONCLUSION

We have created prototype software for the Project54 system that will allow us to deploy modules for two types of remote fleet management (RFM): automatic vehicle location (AVL) and monitoring of the status of the vehicles in the field. The AVL prototype has been tested in one cruiser in the field while the status monitoring software has been tested in the laboratory. We are currently preparing to test the AVL prototype on six cruisers in the field over the period of two to three months and to test the status monitoring software in the field.

The current versions of the server-side software for both prototype applications only record the data they are sent by the clients. However, to make this data truly useful, it will have to be sent to other software modules for processing. For this reason we plan to tie the AVL software to the commercial CAD software in use by the NHDS. We are also working with other commercial CAD software packages, in use by local departments in the state of New Hampshire and expect to adapt both the client side and server side AVL software for use with these packages. For the status monitoring software we plan to create a simple GUI-based application that will display the data collected by the software.

REFERENCES