Clubman Automated Rally System (CARS)
Submitted for review as part of
MSc in Advanced Computer Science

Matthew Jakeman BSc (hons)

September 13, 2004

Supervisor: Prof. Nigel Davies

Additional Material:
http://cars.mjakeman.co.uk

Disclaimer

I certify that the material contained in this dissertation is my own work and does not contain significant portions of unreferenced or unacknowledged material. I also warrant that the above statement applies to the implementation of the project and all associated documentation.

In the case of electronically submitted work, I also consent to this work being stored electronically and copied for assessment purposes, including the department's use of plagiarism detection systems in order to check the integrity of assessed work.

Date:

Signed:

Contents

List of Figures

List of Tables

Abstract

In the world of competitive motor sport safety of competitors and spectators alike is paramount as fatalities and injuries lead to a decrease in the popularity of the sport. The current systems have numerous failings such as the inability to communicate accidents to competitors on a stage and transmission of any injured parties details to the emergency services is unreliable and untimely. The CARS system addresses these issues by allowing marshalls to report accidents electronically, simultaneously to competitors and emergency service crews. The CARS system is implemented using the Global Positioning System for the location information and a wireless network for transmission of data. Usability tests indicated that the interface was suitable for its task, however the use of GPRS as a network medium requires further testing. Future work includes examining bandwidth saving measures and determining if dropped packets can cause a competitive advantage to competitors.

1 Introduction

Amateur rallying, like all motor sport, is a dangerous sport. Safety is a large part of organising and running any motor sport event and the systems that are in place at many events do not take advantage of the numerous technologies that are now widely available and relatively cheap. These could be used to make the sport safer for competitors and spectators alike. In this project we consider the use of wireless networks combined with the Global Positioning System[1] (GPS) to implement a system to improve the safety of rallying. This enables a system to be produced that can monitor the positions of cars during an event and display this information to people at the rally who deal with safety. Further to this if an accident occurs details of it can be sent electronically from the accident scene to paramedics and anyone else who needs to know, telling them the exact location of an accident and details about it, such as any medical conditions the people involved might have to enable paramedics to instantly be aware of this and treat the injured people accordingly.

Wireless networks are increasing in popularity in many different areas due to their ever decreasing costs and the ease with which they can be used because of the increased support for them. The data rates that they can now provide are also increasing so they can support more devices. Technologies such as GPS, which uses a network of satellites to triangulate the position of a receiver on the earths surface, are also useful as they can be used to constantly track the position of objects. In this case this would be useful in rally cars to monitor their positions on a stage. By using technologies such as these, it is possible to create a system that can monitor the position of cars and this information can then be used for other purposes such as reporting accidents to help improve the safety of the sport.

Timing at amateur rallies is also done using a fairly un-sophisticated method as explained in more detail in section 2.3.1. This is another area that could be improved upon by using automated systems to provide more accuracy and less chance of mistakes being made that could prove to be very costly to competitors in terms of final position.

Rallying is a form of motor sport in which the event is split up into several smaller parts called stages. Stages usually vary in distance from about 3 to 30 miles and can

take place over anything up to 4 days. In between stages the competitors get a chance to take their car back to a pre-designated service area at which they are allowed to perform maintenance on the car such as changing tyres, fixing engine problems etc. The time they are allowed at each service interval is pre-determined and if they cannot maintain or fix any problems with the car in that time they are not allowed to continue in the rally.

The cars used for rallying are unusual when it comes to motor sport as they all have to be road worthy. They must all have valid MOT certificates (the test a car has to take every year to prove it is road legal) and they are all cars that were originally made for the road.

Rallying differs to most forms of motor sport because there has to be two people in the car during competition, the driver and the co-driver (also referred to as a navigator). The drivers responsibility is to drive the car around the stage in the quickest time possible. The driver is aided in his task by the co-driver who has one of two types of directional notes at his disposal during the stages.

1. A map of the stage (an example of which can be seen in Figure 5) showing the direction and which route the car has to take.

2. Pace notes. These are notes written using one of several different notations that the co-driver is able to go through extremely quickly and determine what they say about the stage ahead. They are short notes that describe in great detail what terrain is ahead, such as "4 right plus" meaning a fourth gear right hand corner accelerating on the exit. This gives the driver a lot of information in the short time that they have to hear and interpret it due to the speed at which they are going through the stage. An example of the notes a co-driver uses with the pace note system can be seen in Figure 1.

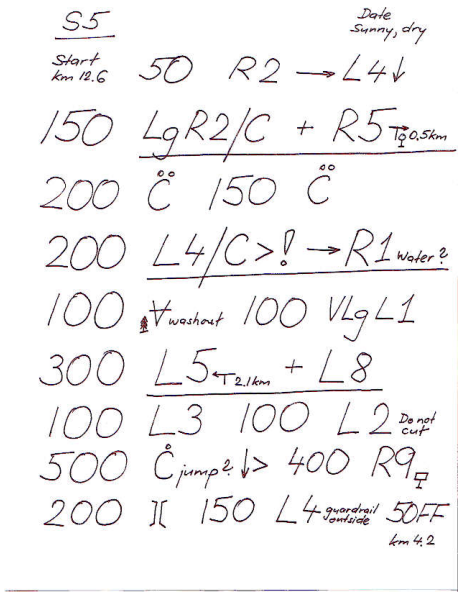

Figure 1: Example co-driver pace notes.

Using one of the above methods the co-driver relays information to the driver about what hazards are approaching on the stage such as turns, jumps, etc, so the driver is able to drive the car through the stage. This is different to circuit based motor sport as the circuit is the same every time so the driver can learn it in advance, therefore not needing a co-driver to guide them through the competition. The co-driver is also responsible for keeping track of all the information that the driver may need to be aware of such as the time that they have to be at the start of the next stage, matters like this need to be kept track of strictly because if they are late getting to a start they may be subject to time penalties.

The driver and co-driver both have microphones attached to their helmets and speakers in the ear space inside the helmet as shown in Figure 2. These are connected via an intercom unit inside the car. This allows them to communicate inside the car whilst on a stage despite lots of external noise, this is how the pace notes are relayed by the co-driver so the driver can clearly hear what is being said.

Figure 2: Rally Helmet.

The environment inside a rally car is very restricted (see Figure 3) for the driver and co-driver because of how tightly they are held in the seats by the 4 point racing seat belts that can be seen seen in Figure 4, for this reason everything they need during a stage has to be easily accessible which is why a co-driver usually keeps all the information they need fixed on to a clipboard.

In most forms of motor sport the competitors compete directly against each other on a racing circuit. They all start off at the same time in an order decided by some form of qualifying (usually a timed lap of the circuit). This is very different to rallying. The order in which competitors start in rallying is decided by the drivers performances in previous rallies, this information is looked at by whoever organises the rally and the competitor with the best previous performances start first, the competitor with the second best performances 2nd and so on through the field. Cars do not start all at the same time as in traditional circuit racing either. The cars set off at time intervals (usually 30 seconds

Figure 3: View inside of an amateur rally car.

or a minute). The winner of a rally is the competitor that completes all OF the stages in the shortest amount of time.

Figure 4: View of co-drivers seat showing seat belts.

2 Introduction To Amateur Rallying In The UK

Amateur stage rallying has several varieties. The following sections describe these in more detail and also looks at the systems currently in place at these events to handle the issues of safety (both for the competitors and the spectators) and timing. Section 2.2 analyses the different types of rallying that exist and explains which one the CARS system is aimed at and for what reasons.

2.1 Types Of Rallying

Amateur rallies in the UK are organised by local motor clubs, generally they organise a rally that the majority of the clubs members are interested in or one for which there will be the most general interest in, depending on the local championships to them so they can make the most money for the club out of the event. This section describes the various different types of amateur rallying that are generally found in the UK.

2.1.1 Multi Use, Single Venue

As the name suggests the event takes place entirely at a single venue and uses the same parts of some stages more than once. A good example of this and one that will be used heavily throughout this report is the Three Sisters karting circuit in Wigan. An example

of a stage diagram given to the co-drivers at a rally from an event at Three Sisters can be seen in Fig. 5.

Figure 5 shows a stage map for two consecutive stages that are identical. The MSA rules state that no more than two stages in a rally can be identical so in a typical rally at this event with 12 stages there are 6 different stage maps.

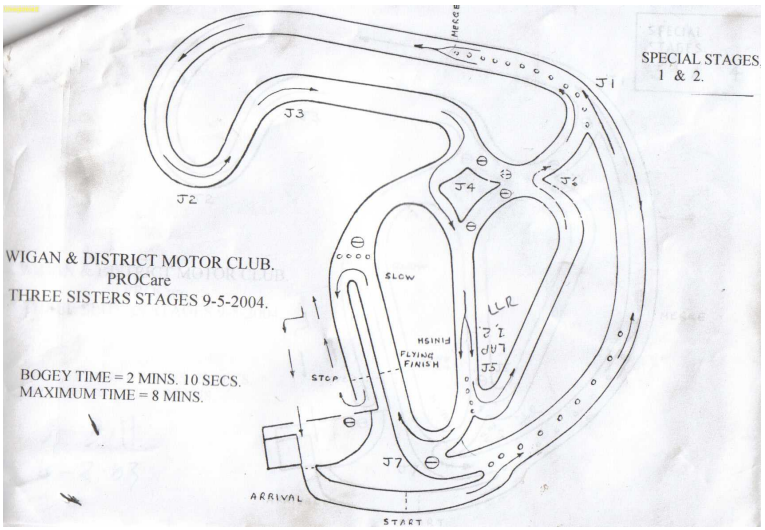

Figure 5: A stage diagram for two stages from Three Sisters in Wigan.

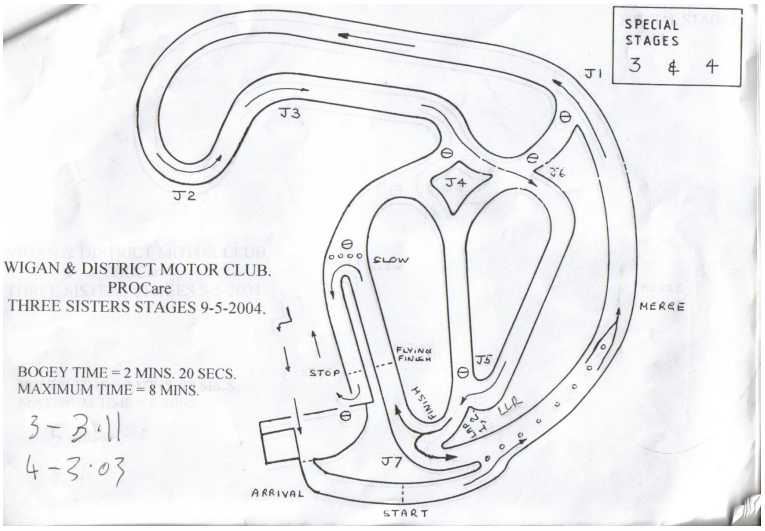

Figure 6: A stage diagram for another two stages from Three Sisters in Wigan.

In the case of Three Sisters, when two stages have been completed, the interconnecting sections of the circuit are used in different ways to make the next two stages different, also after the first 6 stages have been completed the direction is usually reversed so even if the second six stages are the same layout as the first 6, because they are running in the opposite direction they are not deemed to be the same by the regulations, this means

that relatively small venues can be used for multi use single venue events. A typical way in which the track is changed to produce different stages can be seen by comparing Fig. 5 and Fig. 6 which shows the configuration of the track for the first and second stages of a rally (Fig. 5) and the third and fourth stages of a rally (Fig. 6) held at Three Sisters. An example of a change can be clearly seen at point J4 on both stage maps. In Figure 5 as a competitor goes around for the first two stages at point J4 they have a left hairpin followed by a sharp left hand bend, when the first two stages are complete and the course is changed for the third and fourth stages point J4 is no longer the same the stage goes straight past this point.

2.1.2 Multi Use, Multi Venue

Multi use, multi venue events use more than one venue for the rallies but use each venue more than once. In a typical event of this type 3 or 4 venues would be used that would each run around 4 stages, probably using the same stage twice at each venue before altering it. When competitors have to get from one venue to the next they drive on public roads, because of this the cars that are used for this type of motor sport have to be fully MOT'd (as does any rally car), they have to have a valid road fund license and be insured for use on the road.

2.1.3 Forest Rallying

Forest rallying usually consists of stages that are run through forests on loose surfaces such as gravel, or dirt. These events are usually held over wide areas consisting of many different stages which are usually not repeated.

2.2 Analysis Of Rallying

This section analyses the three different types of rallying that were looked at in order to determine which would be most suitable for the CARS system to be based on. This depends on how easy it would be to install and use a wireless network at the events so as to provide coverage to all parties that require it.

The fact that small venues are often used for multi use single venue events as described in Section 2.1.1 makes it ideal for a system such as CARS to be implemented at. Many wireless networks are much easier to deploy, with a high level of connectivity, in small as opposed to large areas.

Multi use multi venue of rallying could be suitable for the system due to the fact that much like multi use, single venue events the places at which the stages are held are generally quite small and so network coverage would be a lot easier to provide. The disadvantage of this type of event is that it is held at more than one venue so there would have to be a network deployed at each area and these networks would probably need to communicate with each other somehow which would increase the overall complexity some what and therefore would not be as ideal as a multi use, single venue event for the CARS system.

Setting up a wireless network at a forest rally would be difficult because of the wide area that the events cover needing network coverage in order to enable all clients to be able to connect at any time, although this would be possible with some networks, it would

be unlikely that they would be able to guarantee full coverage, not only in terms of area, but also in terms of connectivity to all parties.

2.3 Current Rally Systems

The systems in place in rallying for timing and safety are all legislated by the Motor Sports Association[2] (MSA). The rules that are put forward by the MSA in the competitors yearbook[3] they produce each year have to be followed by every motor club that organises a rally and there is an MSA official at each event to make sure that the rules are followed and therefore there is minimal danger to every competitor and also to the spectators.

2.3.1 Timing

The timing system that is currently employed at most events involves a number of people who all have synchronised watches. Cars start off on a stage at pre-determined intervals (usually 30seconds or a minute). These times have to be stuck to as this is the point at which the competitors time is taken as beginning, there are no individual timers for each car as such, just the time at which the car was supposed to start the stage and the time at which it crosses the finish line. The responsibility of checking the car starts at the correct time is down to a number of people at the start control of the stage. One person checks the time the car arrives at the first point of the arrival (basically the queue of cars waiting to get on to the stage) control and the order at which the cars arrive. Two more people check the time at which the cars enter the start control and their order (at this point the cars cannot change their order if any are late as they are committed to their start times which are recorded at this point). Another two people check the start time on the co-drivers time cards to check that the records match up, note down the actual start time of the car and note down the order in which the cars start the stage, this way, there are several sets of records that can be used to check that the cars started at the times they were supposed to. The co-driver also knows the start time so that way they can check they are starting at the correct time and if they believe there to be a mistake they can notify one of the people on the way to the start to have it checked and if there is a mistake, have it rectified.

The time at the finish is usually recorded by an electronic device that can measure the time at which a car passed the finish line accurately. This time is then noted down and the start time deducted from it to obtain the time it took for the competitor to complete the stage.

2.3.2 Safety

Safety at rallies is the top concern of the organisers. Many volunteer marshalls are present at every event and their primary task is to monitor the stages and if any accidents occur to help the competitors and try to alert other approaching cars that there is danger ahead to try and slow them down and, if necessary, to stop cars, for example when a car has had an accident on the stage and is blocking the way and cannot be immediately removed.

Safety at events is not just confined to the safety of the people within the cars. Many events attract a lot of attention from the press and from people wishing to spectate. Spectators can generally gain access to be able to watch from a vantage point extremely

close to where the cars are competing, this means that the marshalls have to keep track of where people are and try to keep them a safe distance from the stage so the risk of them getting injured if a car has an accident is minimised.

One of the big problems currently in rallying is how the accidents are reported. At all rallies there are radio cars positioned along the stages that are in direct contact with the officials at the rally headquarters. The people in the radio cars have sheets of paper listing all the cars in an event, each time a car passes them they tick its number off the list, this way if a car does not come past they can note this and radio this fact in to the headquarters, if the car doesn't turn up at their point this indicates that an accident may of occurred between them and the last radio point. The problem with this is it may just be the case that a small accident or spin occurred that lost the competitor time to another car or the car could have broken down and pulled off the stage. This is why this information cannot be used as a definitive reason to say that an accident occurred. At small events this is generally not a problem because there are many radio cars positioned around the stages and the distances between them are small. This means that either there is line of sight between them so they can see what is happening or that the time intervals are small so determining whether an accident has occurred is a lot simpler. At large events however, the distance between radio cars can be large so this task is made more difficult.

At large events when an accident occurs a marshall has to go from their marshalling post on the stage to the nearest radio car and inform the person there of the fact and whether medical assistance is needed for the competitors or any spectators. This can be very costly as it can take much time to reach the radio car and therefore wastes time that the emergency services could be using to reach the injured people.

Accidents are not the only reason that hazards can come about during a stage. An often occurrence is that a car will break down during a stage, if this happens in a position that is dangerous in any way, marshalls will again have to wave down approaching cars, keep spectators safe and also try to remove the car from the stage.

3 System Requirements

The requirements for this system were derived by contacting a number of different types of people who would potentially use the system. A highly qualified MSA steward, who is responsible for running stages at events as well as dealing with all types of accidents on stages, was contacted and helped to derive many of the requirements.

Multiple drivers and co-drivers were contacted and discussions were had via e-mail to derive some of the requirements, not just with regards to the co-drivers application, but there was also great interest from the drivers and co-drivers with regards to the emergency services application and the way with which the marshalls can report accidents and their severity. This is because the safety of the competitors is most often what is at stake when an accident occurs and the way with which these are handled has great implications for how safe the sport is to them.

Marshalls at an event are responsible for aiding in the safety aspects such as warning approaching cars on a stage if a competitor has crashed and dealing with removing a crashed car from the stage so that others can pass safely.

Marshalling does not require any special qualifications and anybody who wants to can go along to an event and sign on as a marshall. For this reason it was important to get feedback for the requirements from marshalls who have been present at many events so that the feedback was just and had reasoning behind it rather than being based on the views of people who had only got experience at a small number of events and therefore would be less likely to know the relative importance of a number of the issues. For these reasons a number of people who had marshalled at least 15 events each were contacted and liaised with to help derive some of the requirements.

3.1 Co-Driver Requirements

1. The co-drivers client shall at all times let the co-driver know whether the system is online or not using a visual signal.

3.2 Marshall Requirements

1. The position monitoring shall allow marshalls to view the positions of all cars on a stage in real time.
2. Marshalls shall be able to report accidents to the system.
3. Marshalls shall be able to request emergency service assistance.
4. Marshalls shall be able to recommend that a stage should be cancelled.
5. Marshalls shall be able to select the severity of an accident when reporting it.

3.3 System Administrator Requirements

1. The system administrator shall be able to monitor the positions of all cars on a stage in real time.

2. The system administrator shall be able to send a message to all other users that a stage has been stopped or cancelled.

3. The system administrator shall be responsible for sending stage routes and stage updates to all other users.

4. The system administrator shall be able to verify results at the end of a stage and send them to all users.

5. The system administrator shall be able to input stage maps to the system.

6. The system administrator shall have the ability to alter a stage map after it has been entered into the system.

7. The system administrator shall be able to inform all users that a stage has been cancelled.

3.4 Emergency Services Requirements

1. The emergency services shall be able to monitor the positions of all cars on a stage in real time.

2. The emergency services client shall alert personnel using both audible and visual alerts.

3. The emergency services shall be able to view medical details about the driver and co-driver of a vehicle that has been in an accident.

4. The emergency services shall be able to view hazardous details (ie anything about the vehicle that could be hazardous to them whilst rescuing people such as fuel type etc) about a vehicle that has been in an accident.

3.5 Overall System Requirements

1. A competitor shall not gain any competitive advantage through using the system.

4 System Design

The communications framework of the CARS system is naturally suited to using a client-server communications model. The main reason for this is that the platforms the clients run on are mobile devices that will likely have limited processing and storage resources. Because of this, a client-server model allows the bulk of the information processing to take place on the server whereas looking at other models, like a peer-to-peer architecture, requires a more general split of the total processing between all the clients on the system with no central server, this would put a severe strain on a device with low resources.

The design of the 3 clients and the server are discussed in the sections below.

4.1 Server

Network connectivity for the server can be accomplished in one of two ways:

1. Direct internet connection via a wired in connection to the internet
2. Direct internet connection via a connection to the wireless network being used by the clients

The largest amount of network traffic will be generated by the server as it has to receive updates from all the competitors, process this and send it back out again to interested parties. This means that the server will become a bottleneck to the system as it has the combined network traffic of everyone connected to deal with. If a wired in internet connection (such as a broadband connection) is available, this bottleneck is reduced as it will have much more bandwidth available to it, also because wired connections are more reliable, the problem of packet loss due to the inherent nature of wireless networks (interference etc) is reduced so more packets will get through to a client if sent over this connection. This would mean that the system will be able to support more competitors because the server can receive and send more network traffic, reducing the bottleneck. Availability of a wired connection however, would not be available at some of the events because of the remote locations, so the only possible connection would be via a wireless link, the server must be implemented so it handles both of these possibilities.

The server needs to be able to store information regarding the two main areas with which the most data is concerned. These are :

1. The competitors in the rally
2. The stages in a rally.

This information needs to be stored in databases so it can be easily and rapidly accessed at any time to keep the server running as efficiently as possible when it has to process information of either of these types and send it out to the clients. The types of information that these databases need to hold are discussed further in Sections 4.3.1 and 4.3.2.

Constant monitoring is needed of how many users are connected and what type of users these are (marshalls, co-drivers or emergency service teams) along with how to contact each of these users (ie by an IP address if using an IP based system). Using a database here will make it fast to check who is online at any time. This needs to be done

so that when messages need to be sent out to certain users it is easy to process who needs a message and where to send this message. A system of distinguishing the types of users connected from one another needs to be in place because not all the three types of clients need every message, ie medical details about people involved in an accident should not be sent out to anyone other than the emergency service crews for two reasons. The types of information that the online user database needs to hold is discussed further in Section 4.3.3.

1. No-one other than the emergency service crews should be able to get this information on their application because people do not want their medical details shown to people other than those involved in treating them as this is an invasion of their privacy, and also the system would have to conform to the Human Rights and Data Protection legislation.

2. To reduce the amount of bandwidth the system uses as this is very critical on a system using a low bandwidth network such as GPRS.

The main job of the server is to monitor a stage when it is in progress, process information sent to it by the clients and distribute the results from this to anyone it decides needs it. When a packet of data is received it needs to be able to distinguish who sent it and for what reason so a suitable protocol must be implemented on top of the networking protocol being used so it is able to perform these operations.

4.2 Clients

This section described the design of the three clients needed on the CARS system. All of the clients need to be able to connect to the GPRS network to send and receive packets, the specifics of each client is discissed in the following sections.

4.2.1 Co-Drivers Client

The co-drivers client had to be designed to run on a portable and durable device for use inside a rally car. There was not a large amount of user interaction involved between the co-driver and the client but the client had to be able to constantly report the position of the car to the server using a connected GPS device so the server was able to plot the positions of all the cars, to facilitate sending information to the server it must be able to connect to the wireless network in use and create a connection over this network to the server, this is true for all 3 clients that are needed.

This client needs to be able to get the location of the car that it is fixed in at any time. For this it is necessary to connect a GPS unit that can be connected to the mobile device. This must be capable of being queried to return the current position.

The most important component of the co-drivers client is the ability to report the location that the GPS unit is sending to it back to the server for processing. This has to be done at regular intervals, a two second interval is ideal for this as it conserves the bandwidth of the system when many cars are on stage yet also provides a reasonable number of updates for plotting the positions of all the cars so they can be easily recognised by anyone observing a stage map.

There should be a timer available to the user so they can time themselves over a stage, this removes the need to have a separate device in the car to deal with timing as is currently the case. This does not need to be connected to any other component in any way and is purely there for reference by a co-driver.

The co-drivers requirements in Section 3.1 state that "The co-drivers client shall at all times let the co-driver know whether the system is online or not using a visual signal." so a way of giving visual feedback to the co-driver must be built in to the client.

4.2.2 Marshalls Client

The marshalls client has to be written for a device that is small enough for a marshall to carry on them at all times during an event. It has to be capable of reporting a car that has had an accident to the server along with the severity of the accident and also has to be able to receive accident reports that have been logged by other clients. These were the primary goals for the marshalls client although it also had to meet the requirements for the marshalls client set out in Section 3.2.

This client has to connect to the server when it is initially launched to inform it that it is online, it also has to inform the server when it goes offline so the server can maintain a list of the number of marshalls (and which ones) are connected at any given time.

The main functionality of this client is receiving location updates from the server and displaying these on a map of the stage so the marshalls can view the positions of the cars on a stage and use this information if an accident occurs to report it back to the server. Whatever is used to display the location of the cars to a user must facilitate being selected so the user can select a car and then report an accident and the information that is sent to the server can then be derived from the area on the screen the user clicked.

When reporting an accident from the client, the severity of the accident needs to be selected from one of four accident types.

1. *Accident cleared* - When an accident was reported but the car involved has managed to recover and is not hazardous any longer, ie a spin.

2. *Non-Severe* - When an accident has occurred but no assistance is required to deal with it.

3. *Severe requiring medical assistance* - When the paramedics need to be notified.

4. *Severe requiring fire assistance* - When there is a fire hazard at the scene of the accident and help is needed to extinguish it.

In certain circumstances an accident may have occurred that is extremely serious and presents extreme risks to other competitors and/or spectators. In this case after the accident has been reported from the client to the server it is necessary for the marshall to recommend that the stage is aborted so no more accidents occur as a result of the initial one. In this case it must be possible to inform the server that this is the case.

4.2.3 Emergency Services Client

The emergency services client also has to be designed on a relatively small and portable device although it doe not have to be as small as the marshalls client device as most of

the time it would be stored in a vehicle such as an ambulance. The main goals for this client are that it has to be able to receive accident reports along with details about the car, the drivers medical details and the co-drivers medical details so that the emergency service crew could respond quickly, it also had to meet the emergency services client requirements set out in Section 3.4.

Like the marshalls client this also has to be able to notify the server when it is on and offline so it is able to receive location updates and accident information. When it is online it needs to receive location updates of all the cars and most importantly, when an accident has been reported by a marshall and has been processed by the server and sent out to all available emergency service teams, it must present the location of the car that needs assistance so it can be attended to as soon as possible. When an accident report has been received and processed, any specific medical details concerning the driver and co-driver must be able to be displayed by the user along with any extra hazard details about the car so that when they reach the scene of the accident they are already in possession of any knowledge needed to help anyone that has been injured.

Once an accident has been dealt with and any injured parties are in a safe condition it has to be possible for them to report that they have dealt with the accident and are free to deal with any others that may arise.

4.3 Databases

To make the searching and usage of information such as the competitors in a rally fast a database is needed to store three of the main types of data required often by the system. The three data types are described in the sections below.

4.3.1 Competitors

The competitors in a rally need to have certain information about them stored within the system for reference and for safety and medical reasons, there may be a large number of competitors in a rally so a database was the obvious choice to hold the data, a list of the different types of information that have to be stored about a user is shown below.

- Driver Name
- Co-Driver Name
- Driver Medical Information
- Co-Driver Medical Information
- Car Number In Event
- Car Registration
- Car Make And Model
- Engine Size
- Car Information (eg anything especially hazardous to the emergency services concerned with the car such as special fuels etc)

4.3.2 Stages

For the system to function properly, certain information about each stage has to be stored in the system so the position of cars can be plotted effectively, again, a database was a good choice to use for this as it means stages can easily be stored and selected on the server. A list of information that needs to be kept about each stage is shown below.

- Stage Number
- Path to a bitmap image of the stage
- The GPS co-ordinates of the top left hand corner of the stage map
- The GPS co-ordinates of the bottom right hand corner of the stage map.

4.3.3 Connected Users

The system needs to keep a list of users (clients) that are connected to the system so that it knows who to send messages to about different events. The list of users connected at any time is stored in a database so entries can easily and quickly be added and deleted at any time. The fields in the connceted users database are listed below.

- The ID of the user connected
- The IP address of the connected user
- If the user is a co-drivers client, the car number of the competitor

By storing this information it makes it easy to search the database for specific ID's and send the relevant information directly using the clients IP address.

5 Technology Overview

This section describes the technologies that were looked at for use in the CARS system and analysed to derive which of these systems would be best suited.

5.1 Hardware

This section discusses what hardware was chosen for the various parts of the system (server and clients) and for what reasons.

The server was implemented on a laptop PC because this would be the type of machine that would generally be used at a rally due to the fact that they are generally held in remote locations so a desktop PC would not be set up and a laptop is much easier to carry around and set up at an event.

The marshalls client had to be implemented on a small portable device so it could easily be carried around by a marshall at an event. The most obvious choice for this was a PDA as it can easily fit into a pocket and also has high battery life. A Compaq iPaq was chosen because it has a battery life of around 10 hours which can be increased further with an expansion jacket which has a separate battery in it.

Implementing the emergency services client could be done on a larger device than for the marshalls, such as a tablet PC to allow more screen space for the medical details for ease of reading, however for the purposes of this project a Compaq iPaq was again used for this purpose to allow the clients to be developed side by side and because of the good battery life that would aid in testing the system, the same is true for the co-drivers client.

5.2 Platforms

This section discusses the platforms used for the project, both in terms of the Operating Systems (OS's) used on the various devices and in terms of the development platforms (languages, compilers etc).

5.2.1 Operating Systems

The OS used to develop the project on was largely decided on the basis of what came with pre-installed on the Compaq iPaq's used for the clients. These came with Pocket PC 2002, a derivation of Microsoft Windows CE, although it is possible to install other OS's on these devices, such as a version of embedded Linux it was deemed that because of the lack of support for these compared to the large amount of support available for Windows CE this would not be a suitable option so Windows CE was used. Using Windows CE on the mobile devices drove the decision for the server machine OS, to aid with compliance between the platforms the server was also based on a Microsoft Windows OS, Windows XP.

5.2.2 Development

The development platforms used were dependant upon the OS's chosen, because the mobile devices were running a derivation of Microsoft Windows CE, the ideal platform to develop for them was the embedded visual toolkit[4] provided by Microsoft, this provides

facilities for writing the code on a machine running Microsoft's Windows XP OS, the code can then be tested in one of two ways:

1. *Emulation* - The embedded visual toolkit comes with a number of Software Development Kits (SDK's) that can be installed with it for the various platforms it supports (palm-size CE, tablet size CE etc). Each of the SDK's comes packaged with an emulator for that platform so code can be built and set to run on this emulator.

2. *On device* - Code can be built and transferred to a device connected via Microsoft's ActiveSync software[5] as a native executable into the devices start menu (by default), this can then be executed by either disconnecting the device and running it manually or by using a function in the embedded visual toolkit to run it from the desktop machine and interact with it on the mobile device whilst still being connected, by using this method the program can also be remotely debugged on the desktop.

This made it a very attractive development platform and was used for the development of the three clients.

The embedded visual toolkit allows a developer to utilise one of two programming languages for creating applications for use on the Windows CE architecture.

1. C/C++

2. Microsoft Visual Basic

(NB)Microsoft C# can also be used for developing on Windows CE devices by using the Microsoft Visual Studio .NET framework and Microsofts Compact Framework (CF)[6].

For the purpose of this project the decision was taken to use C/C++ because of the authors prior familiarity with the languages and their flexibility. Using these languages presents another choice as there are two methods of programming graphical applications for Windows CE using the embedded toolkit.

1. *Win32 Application Programmers Interface (API)* - This is the standard API that has been used to program windows applications since Windows 95. It is a collection of functions all described in the Microsoft Developer Network (MSDN) documentation that can be used to manipulate all aspects of a Graphical User Interface (GUI), it can be used with both the C and C++ languages as well as many others.

2. *Microsoft Foundation Classes (MFC)* - This is a set of classes built on top of the Win32 API by Microsoft for use with C++ that make it easier for the developer to create and use GUI's, it lacks some of the flexibility of Win32 but generally decreases the development time of an application.

Despite the fact that development time using Win32 is generally a lot higher than using the MFC, the decision was taken to use Win32 because of its flexibility.

Developing for the server application required a platform that could be run on Windows XP and preferably one that was as similar to the platforms used for development

on the Windows CE devices as possible. Microsoft's Win32 API can be used for development on Windows XP, therefore it was decided that this would be the ideal development platform on Windows XP. C/C++ is also supported by the Win32 API and as this is used for the mobile devices it was also used for developing the server, the only difference is that the server code is compiled using a C compiler but this is not supported on Windows CE so the C++ compiler was used although no advantage was taken by using the Object Orientation (OO) that this offered.

5.3 Databases

The database used for this system had to be fast, free, and easy to integrate into a C application. Several databases were looked at initially.

1. *Microsoft Access* - As the code was being written using the Win32 API, an obvious database to consider was Microsoft Access as it is the standard database used in a lot of Win32 applications.

2. *SQL* -

3. *Berkeley* - Berkeley db[7] is a lightweight database system that cuts down on overhead by not having a querying abstraction layer such as SQL. The data is stored in a database according to a key and retrieved in the same way. This means that if you want to store more than one item of data in a database entry the data has to be marshalled in some way, how this is accomplished is described in Section 6.2.1. Berkeley db is designed to provide fast access for storage and retrieval which means that it can process a large number of database actions over short time periods. The Berkeley db system also has a C API so it can be easily used in any C program.

Due to Berkeley DB's C API and the fact that it is extremely fast it was decided that this database would be used for the underlying database for the stages, competitor and online user databases.

5.4 Mobile Networks

The mobile network used to implement the communications part of this system is a vital component and much research was done looking at different available systems to determine which would be best suited in this type of environment, due to factors such as the handoff (the time taken to switch between different base stations) times between different base stations (when this time is large it means that at high speeds the handoff would probably not occur quickly enough to keep a continual network service running and would create a very unreliable network), but would still offer a reasonable amount of available bandwidth. The networks that were considered for this project are discussed in the sections below.

5.4.1 IEEE 802.11

Wireless networks for computer use are becoming more widely available by the day, the most common of these in the public domain being IEEE 802.11 [8]. This is now

	D-Link 520	Spectrum24	ZoomAir	Orinoco
Detection	1630 ms	1292 ms	902 ms	1016 ms
Search	288 ms	98 ms	263 ms	87 ms
Execution	2 ms	3 ms	2 ms	1 ms
Total	1920 ms	1393 ms	1147 ms	1104 ms

Table 1: Link-layer handoff time for different IEEE 802.11b cards.

widely used by companies for their networks as they are cheaper to install than wired in networks and provide good coverage at bandwidths of at least 11Mb/s with most commercial implementations. IEEE 802.11b is also being installed in many public places such as cafe's, petrol stations and MacDonald's, meaning that in a large number of places people can now use mobile computers such as PDS's and laptops to connect to the internet wirelessly.

IEEE 802.11 has a very high bandwidth specification compared to many other wireless networks, the standard 802.11b data rate is 11Mb/s and the standard 802.11g data rate is 54Mb/s. This means that for a system such as CARS the bandwidth that it provides should be easily adequate for such a project.

Handoff was another key area that was looked at whilst deciding which network to use. With cars likely to be traveling at high speeds the handoff time is essential and needs to be as low as possible to obtain the best possible performance from the network and keep clients connected for the maximum amount of time with minimal packet loss. The table shown in Table 1 compares 4 large manufacturers 80211 chipsets and the handoff times for each of their products. The information in Table 1 was taken from the paper "Techniques to reduce the IEEE 802.11b handoff time" by Héctor Velayos and Gunnar Karlsson[9].

There are three stages depicted in the table as described below

1. *Detection* - This is the card discovering that it needs to perform a handoff.

2. *Search* - This is the act of acquiring the necessary handoff information.

3. *Execution* - This is the stage at which the handoff is actually performed.

The most important figure shown is the total time taken by each of the devices to perform the handoff from start to finish. The lowest of these times is 1167 ms.

5.4.2 GPRS

GPRS[10] is the data transfer network that operates over the GSM[11] digital mobile phone network. The GPRS protocol works on top of the GSM system so the issue of availability is the same as with GSM, if a mobile phone (on the GSM network) can be used at a certain position it is possible to use the GPRS network for data transfer.

GPRS is an attractive wireless network because if the area in which connectivity is required is already covered by a GSM network then no setup is required of access points to the network and if it is not covered one mobile mast (access point) can be set up to cover a small area such as that used at a multi use single venue rally.

As GPRS operates over the GSM network it has the same constraints as it. The GSM network was specifically designed to be able to support high speed handoffs up to 250km/h[12] so that it would still operate correctly when a phone is in use in an environment such as on a train or in a car. For this reason it would be able to handle

5.4.3 Network Analysis

The lowest time in which the handoff was shown to be performed in Table 1 was 1167ms. This time is not a problem if the device using the connection is moving slowly, as is often the case with devices connected to an 802.11b network. In the case of the CARS system this would also be fine in for the marshalls who remain stationary for a large portion of the event, or if they are moving it is not very fast, but for the case of the units inside the cars this would mean that on a large percentage of the handoffs there would be a network outage while the handoff was performed. This is because the area covered by a wireless access point is only up to roughly 300 meters outdoors so the maximum time that a car is in range of a single access point at an average speed of 90mph (which is reasonable estimate for a rally car on a stage) would be the maximum diameter of the coverage (600m) divided by the speed at which the car is traveling in meters per second (1mph = 0.44704 m/s), ie

$$600 - (90 \times 0.44704) = 14.91 \tag{1}$$

This means that a car will only be in range of an access point at this speed for around 15 seconds. Therefore a handoff time of 1.1 seconds means that the time a car is fully in range of an access point would be the maximum possible time in range (15 seconds) minus the handoff time (1.1 seconds) which takes it down to under 14 seconds, so if we calculate the distance between access points based on this using the formula

$$Time\ in\ range \times (speed\ in\ mph \times 0.44704) \tag{2}$$

we get that the car would be in range for approximately 560 metres which is under half a mile, so on a 10 mile stage there would have to be an absolute minimum of 20 access points along the stage. For this reason it was deemed that 802.11b would not be a satisfactory network medium for the CARS system.

Due to the slow handoff of 802.11b, and the fact that the GPRS network is widely available (and even if not available can be made so with a temporary mast) meant that GPRS was chosen for use with the CARS system.

5.5 Network Protocols

A number of different network protocols were looked at for the system to use. These protocols are discussed in the following sections. All of the protocols discussed in the following sections operate on top of the Internet Protocol (IP)[13]. Because of this the various components of the system will only need to have a connection to the internet to be able to utilise them.

5.5.1 TCP

The first protocol that was looked at was the Transport Control Protocol (TCP). TCP is a connection oriented protocol which means that it sets up a connection between two devices and keeps that connection open for network communication to take place. This is generally known to programmers as a TCP socket. The client and server can then communicate over this socket by sending packets of data to each other.

TCP is good because it offers guaranteed delivery of packets which for the CARS system would mean that as long as a user is connected to the system they are guaranteed to get all the packets sent out by the server.

A big problem with TCP however is that it doesn't operate well over low bandwidth wireless networks because of its slow start sending and back off mechanisms. When it starts sending a stream of packets over a socket it starts slowly and then sends packets more and more quickly as time goes on, when a packet is lost it assumes this is because of network congestion and backs its sending speed off to a low level and starts the slow start procedure again. This is fine for wired networks as the assumption of congestion is generally true, however, for a wireless network the most common cause of packet loss is not network congestion but loss due to interference which occurs regularly. This means that whenever a packet is lost due to interference TCP backs off and starts its slow start mechanism again, because of the high rate of packet loss on a wireless network this is very inefficient as TCP would generally not get very far into its slow start procedure before it encounters another packet loss and therefore has to start again.

Another problem with TCP is the way in which it would handle with network outages. When a TCP connection is established it sets a timeout that is usually determined by the Operating System (OS), this timeout means that when the TCP connection is broken it waits this period of time before another connection of the same type can be opened. For the CARS system where it is possible that this could happen relatively often and the periods of disconnection would be relatively small, with the system needing to be able to immediately reconnect when the network is available again, this would not be satisfactory as it would result in prolonged periods of disconnection where it could otherwise be avoided.

5.5.2 UDP

The User Datagram Protocol (UDP) is a connectionless protocol, an implicit connection is not set up between two devices as with TCP. UDP is a very minimal protocol and does not provide nearly as many facilities as TCP. UDP has no facility for guaranteeing delivery of information as there is no response for a received packet, nor is there any retransmission of packets. UDP is often used in applications that

5.5.3 Remote Procedure Calls

Remote Procedure Calls [14] (RPC's) were looked at for use with the CARS system as it is a method of distributing an application over many different heterogeneous platforms. RPC's conceal the operating systems and locations of the platforms from the programmer and allow them to call functions remotely without them being aware that they are doing so. This keeps the complexity of the application to a minimum whilst allowing developers

to deploy applications without having to worry about where certain services are located because RPC abstracts low level information such as the network address of a resource hidden from them. RPC also offers guaranteed delivery of packets due to in built systems that check and retransmit packets to make sure the procedure call is carried out and the results returned.

5.5.4 Protocol Analysis

A deciding factor in which network protocol to use was to do with one of the overall system requirements. Section 3.5 specifies that "A competitor shall not gain any competitive advantage through using the system". This means that if one competitor encounters a sustained period in which they do not receive information from the server and another competitor does not experience this on the same stage the second competitor has clearly gained an advantage as if an accident had occurred they would know about it whereas the first competitor would not. There is also a psychological aspect to this requirement that should not be overlooked. Section 3.1 states that a visual signal should be available to the co-driver to notify them when the system is connected and when it is not. A competitor who can see the system is connected knows that if an accident happens they will be informed about it and therefore is reassured by this that the stage ahead is clear and they can therefore drive to their full potential, whereas someone who knows the system is not connected may be more wary of the stage ahead as they are not sure that it is clear and as a consequence they more drive more cautiosly than they otherwise would resulting in time being lost.

The number of packets that a competitors application received on its own is not enough to decide whether they could have received an advantage because there are certain situations where this data is unreliable, for example, if there were two competitors, A and B that were both using the system and both of them had received 50% of the total information during a stage but A had received its 50% from the start of the stage until half way through it and B had received its 50% in a pattern of one in every two pieces of information being sent out, B has clearly had a much better spread of the data and although they have only received half of the total they have had much more reliable information on aggregate and for a more sustained period of time and has therefore gained a fairly large advantage throughout the stage.

Protocol choice could affect this requirement as some of the protocols mentioned above offer guaranteed delivery of packets between two connected end points. This means that any packet sent by the server to a client in the online user database would definitely get there as long as the connection does not go down in the period of time between it being sent and received. Because of this it would be possible to analyse the data that has been received by a competitor easily and therefore possible to decide if anyone had gained a competitive advantage due to the amount of data received and the spread of this data.

As discussed above, TCP offers guaranteed delivery packets so analysing the data that co-drivers applications receive during a stage would be easy, however, the issues regarding disconnections mean that it would be very unreliable in a system such as CARS because of the likelihood of a disconnection to the GPRS network, for these reasons TCP was not deemed suitable.

RPC also offers guaranteed delivery as it guarantees that the function call will succeed

as long as the network remains connected. the way in which it does this in a lot of cases is by using TCP as the underlying transport protocol on which it operates, this means that it suffers from the same problems as TCP with regards to the disconnection problems, therefore RPC was not used for the underlying protocol for the CARS system.

UDP, although not offering a lot of the services of TCP and RPC was appealing for two main reasons

1. Because it does not send any confirmation packets after a packet has been received and it does not retransmit failed packets the bandwidth overhead is very small which is important when using a low bandwidth network medium such as GPRS.

2. UDP is not connection oriented therefore the possibility of regular disconnections will not affect its ability to send out packets, if a competitor gets connected due to a network outage data can still be sent to it over UDP and the fact that it has been disconnected just means it will not receive the data but as soon as it reconnects it can carry on receiving packets immediately.

Despite the lack of bandwidth overhead with UDP, the fact that it does not provide guaranteed delivery means that checking no competitive advantage has been obtained by anyone competing in the rally could not be resolved using any network information from the protocol.

For these reasons it was decided to use UDP for the network transport because of the low overhead that it has compared to other network protocols, however, because of this decision another method had to be derived for deciding that the system was fair to competitors due to the network coverage that was obtained by each one during a stage, how this was accomplished is described in Sections 6.1.2 and 6.1.1.

5.6 Global Positioning System

The Global Positioning System (GPS) uses a network of satellites positioned in space to get the location of a receiver on the Earths surface. To accomplish this the receiver needs to be able to get a signal from a minimum of four satellites. A receiver can measure the time that a signal takes to reach it because the satellites are constantly sending out a signal and the receiver can replicate that signal and calculate the offset between the two signals. The problem here is that the clocks used on board the GPS satellites are atomic clocks which is why they can provide extremely good accuracy, the clocks built in to the receivers are nowhere near this accurate which is the reason that four satellites are needed to triangulate the position, if the clocks in the receivers were as accurate as the ones on board the satellite only 3 signals would be needed. The fourth signal will not intersect with the first three and this tells the receiver that it is not in perfect sync with the satellites clocks and it the receiver can then calculate how far the fourth signal is from intersecting with the first three and use this to alter its time, effectively meaning that the clocks built into the GPS receivers become as accurate as an atomic clock after the offset has been applied.

The receiver also needs to know how high the satellite is to be able to make its calculation, this is done by pre-programming all GPS receivers with an the positions of all the satellites, this can be done because they are in extremely high orbits (roughly 11'000

miles) and therefore their orbits are highly predictable due to the lack of interference from any atmosphere, however, the exact position of all the satellites is constantly monitored to take into account any gravitational interference from the Sun and the moon, and also solar radiation. If a change is noticed then this is relayed to the receivers by embedding this information into the signal the receivers use to calculate the timing difference, this means they are consistently accurate.

After a receiver has determined its position it is displayed to the user as their longitude and latitude values, this is a system that divides the earth surface up using lines of latitude (these are lines which go around the earth in the same direction as the equator, lines above the equator are northern latitude, below are southern latitude) and lines of longitude (these lines go around the Earth in the same direction as the prime meridian, to the right are West longitude and to the left are East latitude), this is shown graphically in Figure 7.

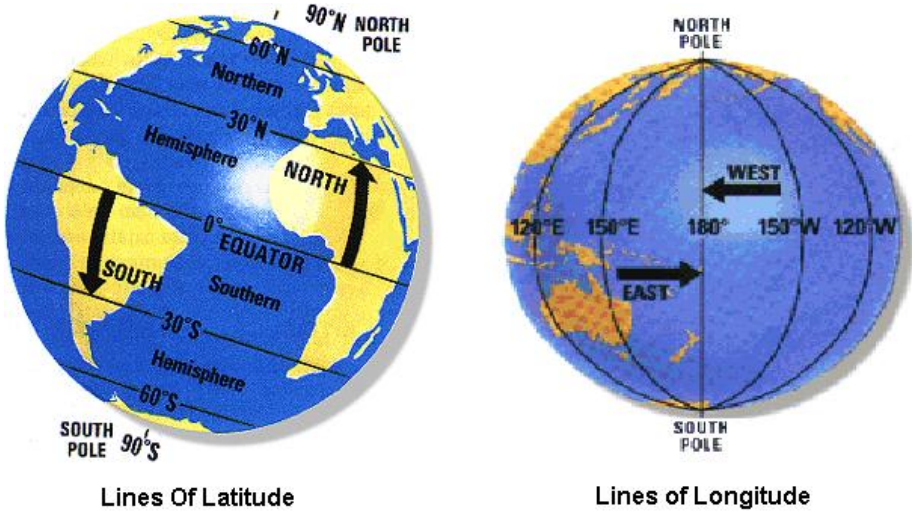

Figure 7: Lines of longitude and latitude.

The system used for numbering the lines of longitude and latitude is degrees, as the earth is an almost perfect sphere. By having a degree of longitude and a degree of latitude it is possible to plot any point on the earths surface.

GPS is the only system of its type and for this reason it was chosen for the CARS system so the locations of the cars on the stage can be monitored by fixing GPS receivers inside the cars.

6 Implementation

This section describes how the system was implemented. Section 5.1 describes the hardware used and why, Section 5.2 discusses the platforms on which the system was implemented.

6.1 CARS Protocols

As discussed in Section 5.5.4 the system was designed to run over the UDP protocol, however, there were two protocols implemented on top of UDP for the CARS system, the following two sections describe how these protocols work.

6.1.1 Client To Server Protocol

Communications from a client to the server take place using this protocol. It has a distinct format that all packets sent follow so the server is able to distinguish between them and decide what to do upon receiving them. The layout of the packet is described below, the fields in a packet are all seperated using the "|" character.

PACKET : 1|2|3|4|5|6|7|8|9|

Above is the layout of the packet, the list below explains the fields.

1. Message Type, described in more detail below
2. Senders IP address. Automatically set.
3. Senders unique system ID. Automatically set.
4. Time of message (Format - int, 24hr clock eg 10:52 would be 1052). Time is always set when the packet is created, does not need to be set in the program.
5. Car number, in case of position update (type 1), the car number reporting, in the case of an accident report (type3) the number of the car that has crashed, not used for a request to stop the stage (type 2).
6. In case of message type 1 (Position update) GPS co-ord longitude else 0
7. In case of message type 1 (Position update) GPS co-ord latitude else 0
8. In case of message type 3 (Accident Report) accident severity, values explained below
9. Stage data - at the end of a stage this contains a string of 1's and 0's showing the distribution of packets the client received.

The 1st section of the packet is the type, this is what the server uses to distinguish what the rest of the information in the packet relates to, the type can be one of four values as described below.

1. Position update
2. Request to stop the stage
3. Accident Report
4. "Hello" Message from the client to notify the server they are online
5. End of stage report from client
6. Emergency service team responding to a request to attend an accident scene or to say an accident has been cleared (these are distinguished between by field 6 being a 0 for the first instance and a 1 for the second)
7. Retirement message from co-drivers client[1]

The 8th segment of the packet describes the severity of an accident if the packet was of type 3, this can be one of four values depending on how bad an accident is, these are described below.

1. Non Severe
2. Severe, Requires medical assistance
3. Severe, Requires fire assistance
4. Accident cleared

Using the information in these packets makes it possible for the server to efficiently parse the packet and decide what needs to be done depending on the information contained, how this is accomplished is described in Section 6.2.3.

6.1.2 Server To Client Protocol

Communications taking place from the server to the client all take place using this protocol. It uses a similar system to the client to server protocol but the packet sections contain different information, it is used so that when a client receives packet it can use a parser to analyse the packet and decide what actions to take. The packet format is described below (all values are separated using the "|" character as for the client to server protocol)

PACKET : 1|2|3|4|5|6|7|

Above is the layout of the packet, the list below explains the fields.

1. *Packet type* - The type of the packet, explained further below
2. *IP* - The IP address of the client

[1]Not yet implemented, see Section 8.2

3. *Driver details** - Medical details of the driver involved in an accident, if the packet contains a position update then this field contains the longitude coordinate
4. *Co-Driver details** - Medical details of the co-driver involved in an accident, if the packet contains a position update then this field contains the latitude coordinate
5. *Car information** - Extra information about the car, if the packet contains a position update then this field is blank
6. *Car Number*
7. *Sequence Number* - If the packet is being sent to a co-drivers client (indicated by the car number not being equal to zero) this is a number (incremented every time a packet is sent to each individual co-drivers client) that contains a number so the co-drivers client is able to detect when it has missed packets.

* Only set if the packet is an accident report to the emergency services, or if it is a position update

As with the client to server protocol the first section of the packet is its type, this can have one of three values depending on what the data in the packet contains

1. Position Update
2. Stahe Abort
3. Accident Report

How the clients make use of the information in these packets is described in section 6.3.1.

6.2 Server

This section describes the implementation of the server.

6.2.1 Databases

As mentioned in Section 5.3, Berkeley db only allows one data item to be stored along side its unique key (it can also be configured to store data with multiple identical keys). Because of this the data had to be marshalled to be placed into the database and unmarshalled when taken out. This was accomplished by creating a `struct` that contains all the fields that need to be stored in the database. The structs for the stage database, the competitor database and the connected user database are shown in Figures 8 and 9 (a `compDetails struct` is used for the user details and the function that marshalls it only takes the relevant information from it, the IP, the ID and the car number, when it puts the information in the database).

By using structs such as these it is possible to set the values of each item in the code, then use a function that takes the `struct` and marshalls it into a suitable array of characters, that can be easily unmarshalled, and store them in the database along with their unique key. The way this is achieved for the stage database `struct`, shown

```
typedef struct StageDetails
{
int stageNo ; //Number of the stage
char *bmpFile ; //Path to stage bitmap file
char *gpsTopLeft ; //GPS
char *gpsTopRight ;
char *gpsBotLeft ;
char *gpsBotRight ;
}stageDetails ;
```

Figure 8: Stage Database Struct.

```
typedef struct competitorDetails
{
char *driverName ; // Driver name
char *driverMedDetails ; // Driver medical details
char *coDriverName ; // Co driver name
char *coDriverMedDetails ; // Co driver medical details
int carNo ; // Car number
char *carMake ; // Make of car
char *carRegistration ; // Car registration page
int engineSize ; // Engine capacity in cc
char *carInformation ; // Car information
} compDetails;
```

Figure 9: Stage Database Struct.

in Figure 8, is by putting the items into a single char array using the | character as the seperator so that it can be unmarshalled by tokenising the array and storing the values in a `stageDetails struct` where they can easily be accessed. The format in which the array is stored in the database is shown in Table 2. The competitor database is marshalled using a similar `struct` containing the information specified in Section 4.3.1, this is marshalled from a `compDetails struct` to an array of characters using the | character, the `struct` for the online users (a `packStruct`) is marshalled using the same technique containing the information shown in table 4.

| stageNo | bmpFile | gpsTopLeft | gpsTopRight | gpsBotLeft | gpsBotRight |

Table 2: Marshalled `stageDetails struct`.

6.2.2 Interface

The user interface for the server is very important because it is the only was to give feedback to the user about where the competitors are on a stage when the system is

| driverName | driverMedDetails | coDriverName | coDriverMedDetails | carNo | carMake | carRegistration | engineSize | carInformation |

Table 3: Marshalled `compDetails struct`.

| ID | IP | carNo |

Table 4: Marshalled packDetails `struct` for an online user database entry.

online and it also givers the user the functionality to input competitors into the database and also input the stage information.

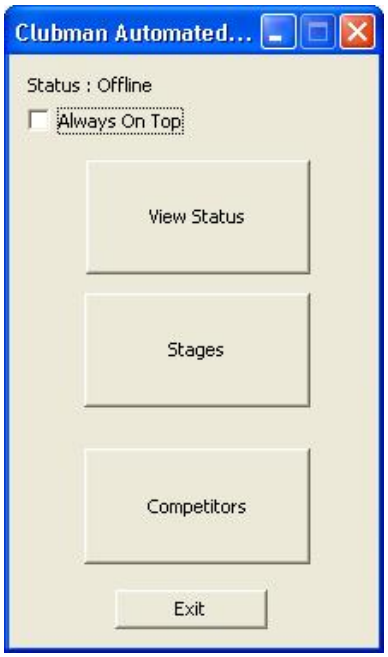

Figure 10: Main application window.

Application start up greets the user with a small main window shown in Figure 10 that gives them four main options

1. *Competitors* - Go to the competitor window to add, delete or modify competitor information
2. *Stages* - Go to the stage window to add, delete or modify stages
3. *View Status* - View the status of a stage
4. *Exit* - Exit the CARS program

When a user clicks the competitors button, a new window appears allowing the user to configure the entries in the competitor database as shown in Figure 11.

This presents the user with 3 buttons for configuring the database along with a list view of the information already stored in it for quick viewing. If a user clicks on the

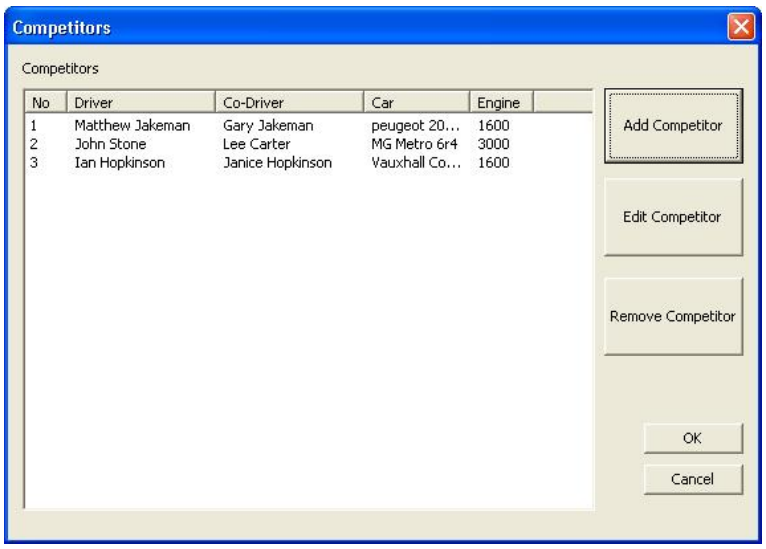

Figure 11: Competitor window.

button to add a competitor they are presented with a window like the one shown in Figure 12 but with no information filled in the edit boxes, using this they can complete the fields with the relevant information and store it in the database.

Editing competitors is achieved by either double clicking on the entry in the list view of the competitor window or selecting an item and clicking on the edit competitor button. Once this has been done a window like that shown in Figure 12 is displayed containing all the information from the database so it can be edited and replace the existing entry when complete. To remove a competitor from the database it simply has to be selected and then the remove competitor button clicked, this will confirm with the suer that they really want to delete the entry, if the user confirms it then the entry is removed from the database.

Editing stages is much the same process as for competitors, when the stages button is clicked in the main window (Figure 10), a new window is displayed with a list view containing all the stages contained in the database, and the options to add a stage, edit a stage and remove a stage as shown in Figure 13. Figure 14 shows the window that the user can use for adding or editing a specific stages information.

When a stage is in progress, the server user has to have the status window open, this is accomplished by clicking on the "view status" button on the main window to open the window shown in Figure 15.

This displays a number of options and pieces of information to the user

1. Bitmap image of the stage that displays the position of all cars on a stage
2. A button to toggle the online/offline status of the server
3. An abort stage button that notifies all clients if a stage needs to be aborted
4. A button to change the current stage of a rally that is in progress
5. A list view that displayes all the requests to abort a stage received from marshalls

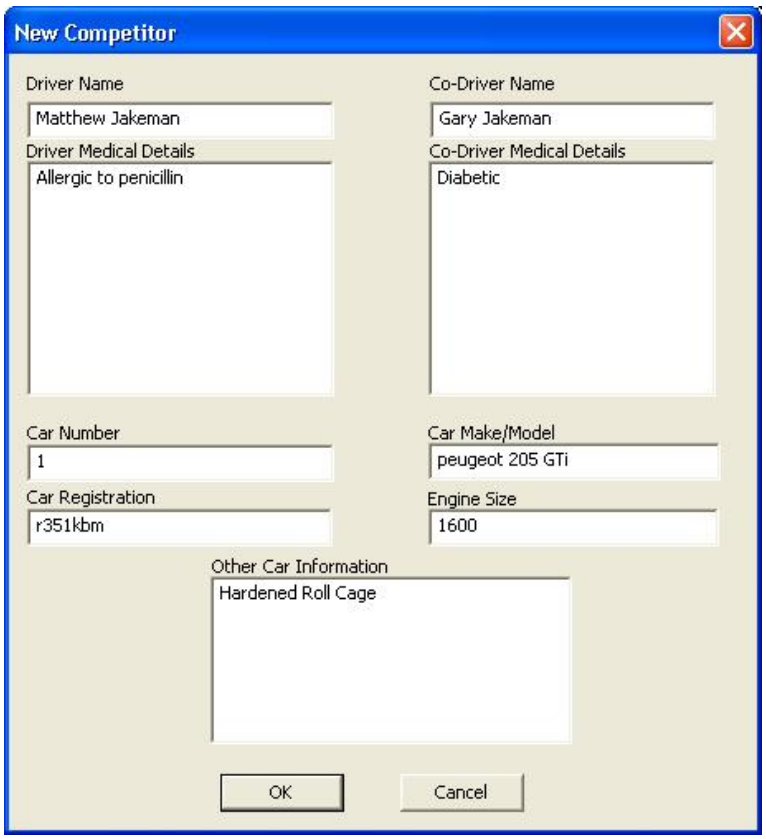

Figure 12: Window for adding or editing competitors.

6. A button that brings up another window that shows the user which clients are currently connected

7. A close window button that takes the user back to the main window

This is where all of the interaction between the user and the server takes place during stages.

6.2.3 Communications

The communications portion of the server is implemented using Microsofts WinSock API[15]. This allows a developer to use standard system calls to create network connections (sockets) and send/receive data over them.

When a user is in the status window and clicks the "Go Online" button seen in Figure 15 a socket is created to listen on port 20002 (chosen because it is not a commonly used port for other applications[16]). This is a UDP socket so it is not connected in any way but will accept packets sent to it by the clients. After this has been created successfully a thread is created to listen for incoming packets, when a packet is received a function is called to parse it and take the appropriate action depending on the type of packet received. The protocols that are used have already been discussed in Section 6.1, here will be discussed the actions the server takes depending on the packet it receives from a client.

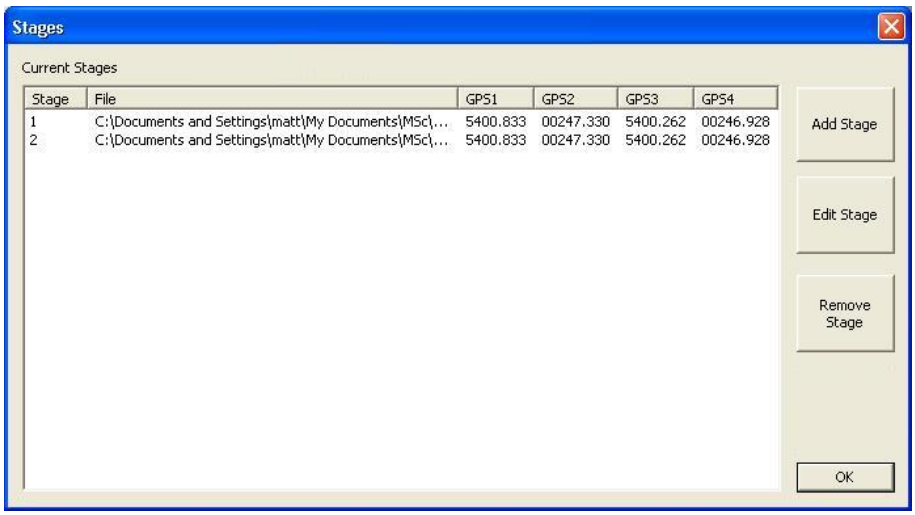

Figure 13: Window for adding or editing stages.

Determining the type of packet that has been received is accomplished by taking the string of data from the packet and tokenising it using the | character as the separating value, this returns a pointer to an array of characters up until the first occurrence of |. Then when called again returns a pointer to an array of characters up until the next occurrence etc. By doing this the packet can be split up into the different fields and this data can then be stored in a packet structure `packStruct` as shown in Figure 16.

By doing this the type can be extracted and stored as a short integer and then switched to perform different actions depending on the value. The different packet types will now be discussed and the actions that are taken for each type.

1. *Position Update* - If the packet is of this type (ie type is 1) it means the packet contains a position update from a co-drivers client informing the server of the GPS co-ordinates of the car. The server then continues to tokenise the string and store the information in a `packStruct` so it knows which client had sent the update, the number of the car, time of the report and the co-ordinates of the car. The server then cycles through the database and re-sends this information out to all the clients connected to the server (how this is done is explained later in this section) so they are able to update their displays to reflect the positions of the car from which the server received the packet. After this has been completed it calls a function to update the servers display with the cars position by placing a button on top of the stage diagram.

2. *Request To Stop Stage* - This inidcates that a marshall has requested that the stage is stopped. On receiving this the server pops up a message box to the user informing them that the request has been received, it then tokenises the rest of the packet and stores it so it can update the list view on the status window with the IP address of the marshall that sent it, the ID of the marshall and the time at which it was sent.

3. *Accident Report* - An accident report type packet means that a marshall has reported an accident occurring on the stage, when this is received the server stores the

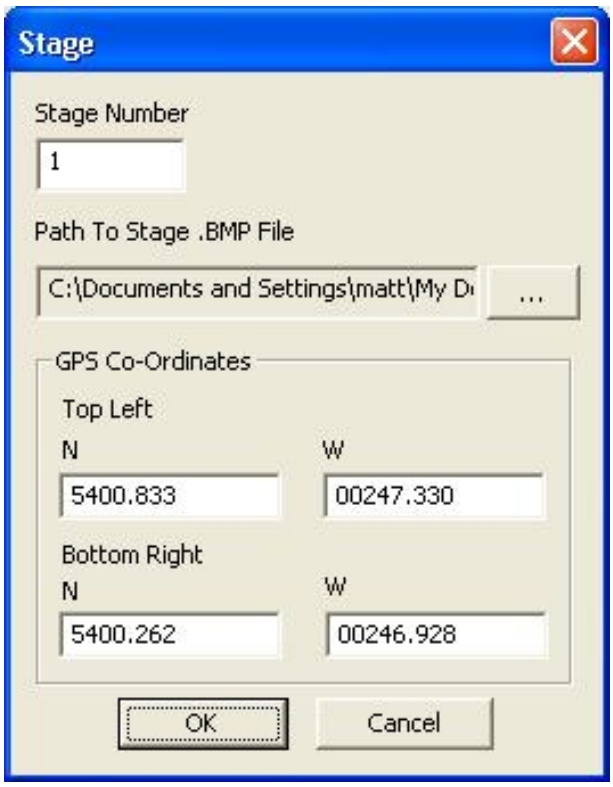

Figure 14: New / Edit stage window.

rest of the packet in a `packStruct`, it then gets the information about the competitors involved in the accident by retrieving the data from the competitor database based on the number of the car in the accident and storing it in a `compDetails struct`. If the accident has the severity two or three, the emergency services are now required to be informed of the accident along with the medical details of the competitors and the information regarding the car it cycles through the online user database and sends this information out to any emergency services clients connected to the system (signified by the first two characters of the clients ID being "ES").

4. *Hello* - This indicates that a new client has come online with the system and is informing the server of this. The server tokenises the packet and stores the information in a `packStruct`, it then adds an entry into the online user database containing the IP address of the client that sent it, the clients ID and, in the case of a co-drivers client (specified by the first two letters of the ID being "CD") it also stores the car number.

5. *End Of Stage Report* - This is sent to the server when a client has finished a stage to indicate the distribution of packets it received[2].

6. *Emergency service team attending accident* - This message is received from an emergency service client in response to an accident report being sent out, to inform the

[2]This has not yet been implemented but the way in which the fairness could be check is discussed in Section 8.2

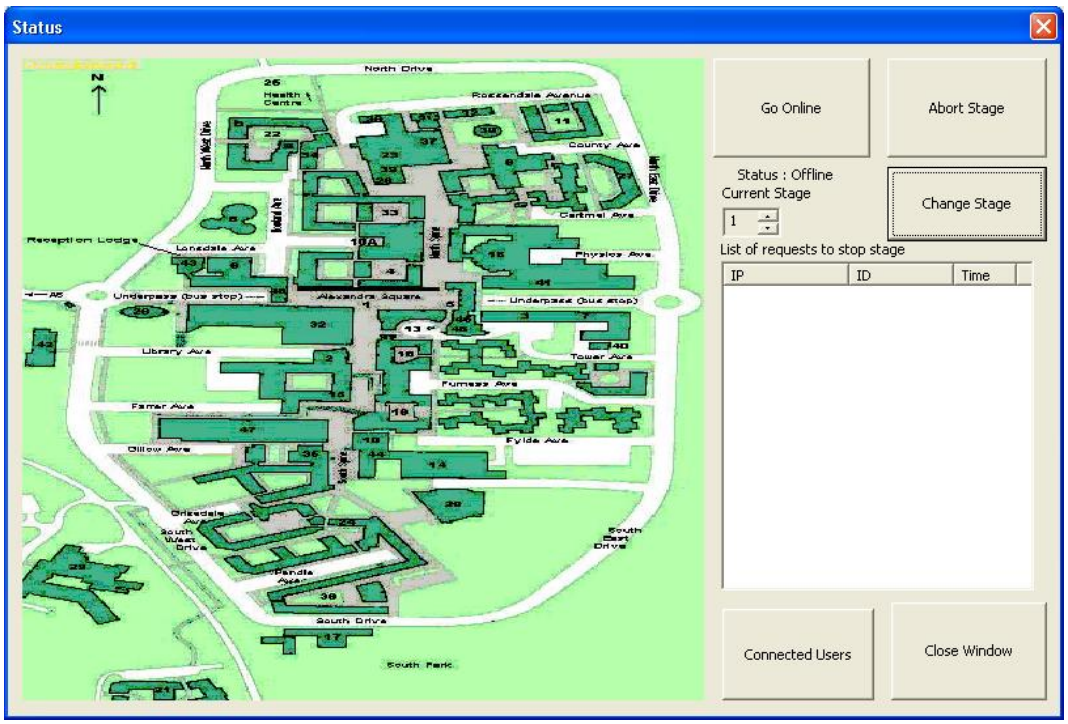

Figure 15: Stage status window.

server that the specified emergency service team are attending an accident (if field 6 is 0) or an accident being cleared (is field 6 is 1). When the server receives this message it displays a message box to the user informing them of who is attending if field 6 is set to 0 or displays a message box saying the accident has been cleared if field six is set to 1.

To enable the server to send packets out it creates a UDP socket when the server is initially started, this socket is global to the application and can be used calling a function named SendToClient(), this function takes an `outPack struct` and creates a packet based on the information contained in it, then sends it to the client specified in the `struct`. The only times the server needs to send information to the clients without the packet being generated by the incoming packet parser is when a stage is aborted and when the stages are changed. When the "Abort Stage" button is pressed a packet is generated using the SendToClient() function and sent out, when the stage is changed the system sends out a packet informing the clients of this in the same manner.

6.3 Clients

This section describes the implementation of the clients.

6.3.1 Communications

All three clients have similar mechanisms for parsing the packets that are received, this has to be done according to the server to client protocol rules laid out in section 6.1.2.

```
typedef struct packetStruct
{
short type ; //Packet type
char *IP ; //IP address
char *ID ; //Client ID
long time ; //Packet time
short carNo ; //Car number
char *longitude ; //Longitude
char *latitude ; //Latitude
short severity ; //Accident Severity
char *stageData ;
}packStruct ;
```

Figure 16: Packet structure.

```
typedef struct outgoingPacket
{
short type ;
char *IP ; //IP address of recipient
char *data ;
char *data2 ;
char *data3 ;
short carNo ;
} outPack ;
```

Figure 17: Client packet structure.

The clients all use the WinSock API much like the server to create connections and send/receive data. All 3 clients have a thread that listens for incoming packets and parses them when received, the three packet types that can be received are discussed below along with the actions each client takes upon receiving them

1. *Position Update* - All 3 clients act the same upon receiving a packet of this type. First they tokenise the packet to fill a client `outPack`, as depicted in Figure 17. Once this is complete the values retreived from the data and data2 (the longitude and latitude), along with carNo (the car number) are placed into a `packStruct` (as seen in fugure 16), by doing this it is possible to use the same function (slightly modified with regards to placing the window over the bitmap) as in the server for placing the car on the screen using the same algorithm explained in Section 6.4.

2. *Stage Abort* - When a client receives a packet of this type, again, all 3 clients act the same by simple displaying a message box on the screen to inform that the stage has been aborted.

3. *Accident Report* - The three clients behave differently in the case of an accident report, the way in which each client handles the packet is described below.

(a) *Emergency Services* - When an accident report is received by the emergency services client it tokenises the packet and stores the information about the driver, co-driver and the car in an `outPack struct`, then displays a message box to the user informing them of the accident and asks if they will attend the scene. It then displays a button at the bottom of the screen asking the user if they wish to see the accident details, when they click this button a window is displayed containing three edit boxes with the three lots of details in it obtained from the `struct` earlier filled.

(b) *Co-Driver* - The co-driver client takes the accident report packet, tokenises it and fills an `outPack struct` with the information about the accident, the fields in the `struct` concerning the information about the driver, co-driver and car (fields three, four and five) do not have any information in the when received by the client so they are ignored. After this is complete an audible warning is played for the co-driver to alert them of the fact that an accident has taken place and a message box is displayed telling the co-driver what car has crashed and where[3].

(c) *Marshall* - The marshalls client takes the packet and tokenises it filling an `outPack struct` (as with the co-driver client, fields three, four and five contain no information). A message box is then displayed to inform the marshall which car has had an accident[1].

The clients can all send packets to the server but as the packets sent are client dependent this is discussed in Sections 6.3.2, 6.3.3 and 6.3.4

6.3.2 Marshall

The marshalls client has to allow the marshalls to monitor the positions of cars on a stage and also report accidents to the server, how this is accomplished is described in this section.

When a marshalls client is started the first thing it does is to go online, it does this by calling a function that creates UDP socket to receive packets over, a thread is then created that receives packets on this socket. This thread is responsible for receiving the packets and calling a function which parses the packet as described in section 6.3.1 and takes the required actions.

To send the packets from the client to the server a global UDP socket is created when the system goes online. This allows a function to be created that can use this socket and send packets using the WinSock API. One function `SendToServer()` was implemented that took an argument in the form of a `struct packStruct` and converted this into the correct packet format and then sent it out over the socket.

The packets that get sent from a marshalls client depend on when an accident gets reported. The main screen of the marshalls application shows the same map as the server with a button at the bottom for reporting accidents. When this is pushed the marshall can enter the car number involved in the accident and select one of the four severity levels explained in section 4.2.2. Once this is completed a `struct` of type `packStruct` (as used

[3]This is the current state of implementation but the aim is to have the car involved in an accident highlghted red

by the server and shown in Figure 16) is filled with the relevant information and sent to the server for analysis. This is the only type of packet the marshalls client sends after the initial "hello" message.

6.3.3 Co-Driver

The co-drivers client works in much the same way as the marshalls client on startup, the packets that it needs to send to the server however are different. The co-drivers client has no need to report accidents to the server, its main purpose is to display the stage map to the co-driver, the position of all the cars on the stage and report the location of the car back to the server (as well as allowing the co-driver to time the stage).

Reporting the location of the car on the stage is the main priority for the application. It accomplishes this by being connected to a GPS unit, this unit is constantly taking readings of the location which are available to the application via a Windows dll called "IPAQSerialGPSDLL.dll". This allows the iPaq to read the location from the GPS unit as an array of characters that can then be parsed to give the type of location being returned (longitude/latitude, grid reference etc). This has been done in a function called `getLocation()` that returns a `coOrds struct` (shown in Figure 18) with the current coordinates of the GPS receiver to the application.

```
typedef struct coOrdStruct
{
WCHAR longitude[15] ;
WCHAR latitude[15] ;
}coOrds ;
```

Figure 18: Coordinates structure.

The `getLocation` function is called every 2000ms using a Win32 API provided system rather than using a thread because of its simplicity. This then puts the location information into a packet and sends it via UDP to the server.

6.3.4 Emergency Services

The emergency services client sends very little information to the server apart from its initial "hello" message unless an accident occurs. On receiving a packet saying an accident has occured, the user has to reply saying whether they are attending an accident, the client uses the SendToServer() function described in Section 6.3.2 to send the packet by setting the value of the sixth field in the packet to 0. The only other time a packet needs to be sent by this client is to inform the server that the accident has been cleared, in this case the value of field six needs to be set to 1.

6.4 Positioning Algorithm

In order to plot the position of a car on a stage map a positioning algorithm was developed. This algorithm takes the GPS coordinates of the car received from a client and translates them into their representative points on the bitmap image. The formulas derived to

accomplish this are shown in Equations 3 and 4. The variables used in the formulas are described below (accompanied by Figure 19)

- Mx_1 and My_1 are the map (GPS) of the top left hand corner of the stage map
- Sx_1 and Sy_1 are the screen co-ordinates, these are put in for reference only as they are both 0 and not actually needed because they will always be 0 and so can be hard coded in as this.
- I_x and I_y are the GPS co-ordinates of the object that needs placing on the map (in this case a car)
- Mx_2 and My_2 are the map co-ordinates of the bottom right of the map
- Sx_2 and Sy_2 are the screen co-ordinates of the bottom right of the map (these can be calculated by getting the size of the rectangle in the code)

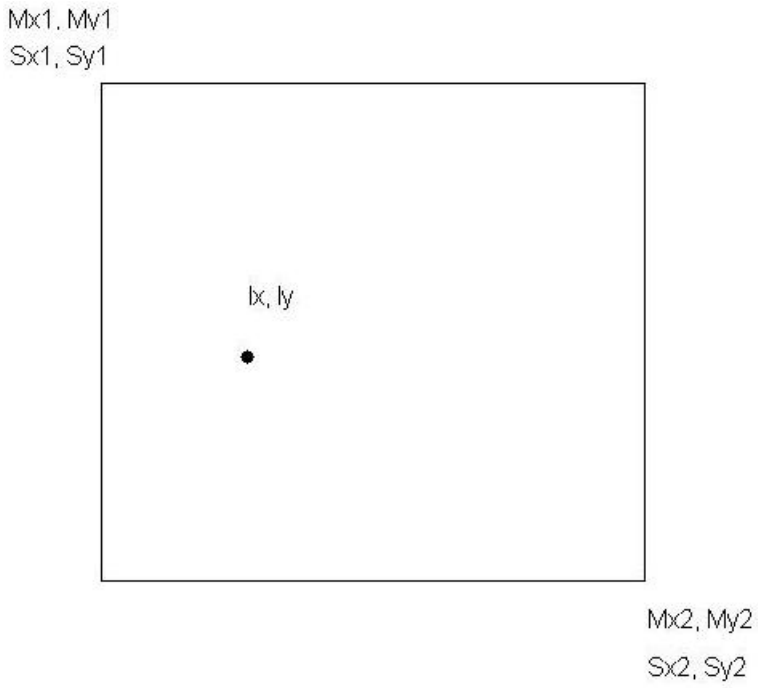

Figure 19: Algorithm variables display.

Using these variables it is possible to derive a formula that will correctly plot the position of the car, the formulas that war derived for the x and y co-ordinates are shown below.

$$Mx_2 - \left(\frac{Mx_2 - I_x}{Mx_2 - Mx_1}\right) \times Sx_2 \tag{3}$$

$$My_2 - \left(\frac{My_2 - I_y}{My_2 - My_1}\right) \times Sy_2 \tag{4}$$

The formula for obtaining the x co-ordinate will now be discussed, the formula for the y co-ordinate works in the same way. The formula takes the co-ordinate of the car that needs to be placed on the map (I_x) and subtracts it from the map co-ordinate of the bottom right corner (Mx_2), this is then divided by the value obtained by subtracting the map co-ordinate of the top left (Mx_1) from the map co-ordinate of the bottom right (Mx_2). This then gives a ratio between 0 and 1 that represents how far across the map the car is, to obtain the pixel value this represents it is then multiplied by what is effectively the width of the map, due to the fact that the left hand side is always 0 (Sx_2). This then gives the x co-ordinate on the picture where the object representing the car needs to be displayed. Once this has been completed for both the x and the y co-ordinate the position can be marked on the map and displayed to the user.

7 Evaluation

To evaluate the system two types of testing were carried out.

1. *User testing* - This is discussed in Section 7.1, it was undertaken by testing the system on a number of users not familiar with rallying to discover how easy the system was to use.

2. *System Testing* - This is discussed in Section 7.3, it was undertaken by testing how well the system could perform in various manners.

For evaluation purposes, the CARS system was tested on the Lancaster University campus using a map of the campus from the universities web site[17]. The first challenge was calculating the GPS co-ordinates of the top left and bottom right corners of the map, to do this there were two possible methods.

1. Go to the two corners on the map with a GPS unit and manually take the readings.

2. Go to places on the map that are well defined and take readings and calculate the positions of the corners from these.

As the two corners are not near any features the first option would not be a good method as they would not be accurate so it was decided to use the second method. The points that were taken around the campus are shown in Figure 20.

The numbers on the map represent the points at which GPS readings were taken, these are listed below.

1. N 5400.612 W 00247.301

2. N 5400.818 W 00247.227

3. N 5400.818 W 00246.944

4. N 5400.628 W 00246.944

5. N 5400.459 W 00246.949

6. N 5400.338 W 00247.170

A line was drawn between points one and two and a right angled triangle was drawn from these points using the first line as the hypotenuse. The distance between points one and two was then measured and from these measurements a ratio of map distance to the difference in GPS co-ordinates was calculated, this could then be used to calculate the top left corner of the maps location which is calculated to be N 5400.833 W 00247.330. The ratio that was used to determine this was 1 mm on the printout of the map was equal to a change of .00229 units either way for the GPS co-ordinates. Since the ratio had already been calculated, determining the co-ordinates of the bottom right of the map was a simple matter of measuring from one of the known points listed above, the result was that the co-ordinates were N 5400.262 W 00246.928.

This information was then entered into the server so that it was able to plot cars on the campus map. This meant that the system could be tested on the University campus,

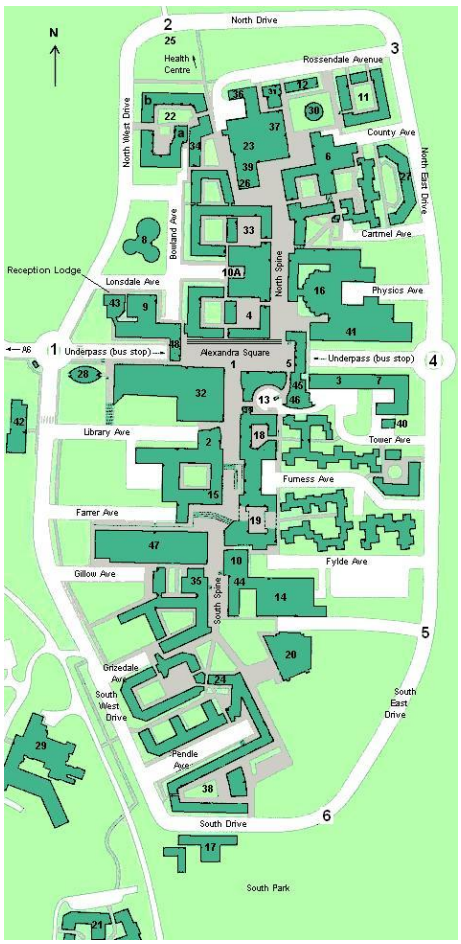

Figure 20: Lancaster University Campus Map.

however, as no GPRS equipment was available for testing this system it had to be tested in areas that had another form of wireless network coverage, the only areas that had any form of wireless network were run over IEEE 802.11 so this was chosen, the main difference here was the bandwidth, but as the tests were only being carried out using a very small number of clients this did not affect the testing.

7.1 User Testing

This section describes user tests carried out on the system. Due to time constraints only two different people were used for the testing with various backgrounds and varying knowledge of rallying, the most knowledge either of the users had of rallying was one person who had been to one amateur rally event.

The testing was carried out by giving the subject a brief introduction to how rallying worked and a brief introduction to the system. The user was then monitored while interacting with the different clients (a co-drivers client was not made available as only two iPaq's were available for testing so the marshalls client was configured to send location updates from the GPS unit to the server) and the server to see how they used the system and how easy they found the interface to operate. Then the user was questioned at the

end of the session to get their feedback, the results are shown in the sections below. The tests were not carried out in an in-car situation because of the need for wireless connectivity over IEEE 802.11. Instead they were conducted with the user holding the device and moving around to get a picture of how the display worked and how the updates were displayed on the screen.

7.1.1 User 1

User 1 had no experience with rallying but an extensive background in various aspects of computing. They had no problems understanding the basic system and rallying concepts.

The user thought the server was clean and easy to use for adding entries to the competitor and stage databases, the windows were straight forward and the fields when adding to the database were clearly marked. They also thought the status map was slightly better on the server than on the clients because of its increased size, however they did not like that the message when a request to the abort the stage was received was only put in the list view and not made more prominent to the user.

The user took to the system straight away and was walking around to watch how the position changed on the screen, the user then started playing with the accident reporting feature of the marshalls client to see how the accidents could be reported and what feedback was received on the emergency service client if medical assistance was required. After this the user checked all aspects of the data displayed on the emergency service client to check it could all be accessed and viewed, he also checked the marshalls screen to check that the information had not been relayed there. After the user had continued using the system for about a further 5 minutes without really trying any more of the features the test was stopped and the user interviewed.

The user very impressed with the way severity of accidents is handled and the overall display that is used to relay the position of the cars. Was slightly disappointed with the update speed of the position of the cars as it seemed like the system may have crashed when in fact it hadn't. The user also suggested having the ability to zoom in on certain parts of the stage map to get a clearer view of exactly where an accident had occurred.

7.1.2 User 2

User 2 had been to an amateur rally before as a spectator so knew the basics of how the events were organised and also had a background in computing.

First of all this user acted quite differently to the previous user, they started looking at all the interfaces of the clients and the server before attempting to interact with the system. After they felt they had a grip of what each part of the system did they picked up the marshalls client and started to walk around with it. After a few minutes watching the position being updated on the screen they reported an accident of type non sever to check what happened. When they had finished this the reported a severe accident requiring fire assistance and observed the response on the emergency services client, the user then turned the emergency service client off. The user then took the server off line and started to use the features of the server to add new competitors to the database, then the user took the server back on line and started using the clients again. At this point the user had noticed an error with the system and the test was stopped, the user was then asked about the system.

The users first comment was regarding the error that had happened (when the user had gone back on line the emergency service client was still listed as being on line), the user asked why this happened and it was explained that the reason for this was that because the iPaq had been turned off rather than the application being explicitly shut down no message had been sent to the server to inform it that the user had gone offline, the user first suggested trying to fix this. As with the last test, this user thought the update was also a bit slow. The user was very impressed by how easy to use the interfaces were and how little clutter there was on the screens that made actions that could be taken easy to identify.

7.2 User Testing Analysis

Overall the user feedback suggested that the main problem with the system from an interface of view was the delay between updates, although this was not much of a problem when moving small distances, the large distance that a car could move on the map in two seconds meant that the gap between updates in term of screen are would be large. Another problem was that of the packet not being sent to the server when an iPaq was switched off, although this may not be desirable if the ipaq has the auto power off option switched on and the iPaq turns off but is only off for a short period of time.

Overall the users seemed to use the interface easily and be able to navigate their way through the windows and menus with ease.

7.3 System Testing

One of the interesting tests on this system is how many clients it would be able to support during an event. To test this it the average size of different packets was calculated and how often they would need to be sent. By doing this and comparing it to the available bandwidth (with the server both on a wired broadband connection and on a wireless GPRS connection) it is possible to give an estimate as to how many can be supported. To do this, two main variables need to be known

1. Available bandwidth over the network
2. Size of the packets

GPRS networks can supposedly handle up to 170kbps of data downstream, but in reality it is around the 50kbs with an upstream of somewhere between 10-28kbps. Because of the different data rates for upstream and downstream and the different types of packets sent either way the estimations were made in two sections. Section 7.3.1 describes the upstream evaluations and Section 7.3.2 describes the downstream evaluations. All these evaluations are from the point of view of the server as it has the most information flowing in and out of it and is therefore the bottleneck in the system. The tests described below all assume 100% receipt of pacets even though in reality on a wireless network this is highly unlikely and this would result (in the upstream tests) of more bandwidth being available, how much however cannot be tested accurately as the packet loss would be sporadic and effected by external interference. The formula for calculating the number of clients the system can support is shown below and is the same for all cases

$$Cars\ Supported = \frac{\left(\frac{Network\ Bit\ Rate}{8}\right)}{Packet\ Size} \tag{5}$$

This formula relies on the fact that the clients send position updates to the server every second (the delay defined in the application is actually 2), it was chosen to be tested like this because some of the user testing said that the delay between updates was not fast enough so the update delay was reduced to 1 second for the tests to create a picture of how final application would react. If results are required for different update delays the results simply have to be multiplied by the delay required (for example if a delay of half a second was required the results should all be multiplied by 0.5)

7.3.1 Upstream

The calculations for how many cars the system could support based on the upstream were carried out using three test values for available bandwidth (10, 28 and 256kbps, the first two are values that could be available from a GPRS connection and the third is from a typical ADSL line) and three values for the average packet size. The values for the average packet size were derived by taking the smallest possible packet, the largest possible packet and an average of the two as described below. The packet sizes were calculated by taking the values that a position update packet could be as this is the type of packet received more than any other during a rally. This is due to the fact that each co-drivers client sends one every two seconds compared to for example the end of stage report which in the case of a 10 minute stage would add another 300 bytes on to one packet per stage, other packets such as accident reports can also be ignored due to their similar size and their infrequent occurrence.

- *Smallest Packet* - The smallest size packet is calculated with the following values in a `packStruct struct`

    1. *type* - This field is always one byte long due to it being represented as a short
    2. *IP address* - Taken to be 8 bytes for the shortest possible IP address (eg 10.0.0.1)
    3. *ID* - Taken to be 1 byte
    4. *time* - The shortest this can be is four bytes when the time has no second value associated with it (ie it is an exact minute, although in practice this is impossible)
    5. *Car Number* - This field is always one byte long due to it being represented as a short
    6. *Longitude* - This value is constant at 9 bytes (The maximum accuracy value)
    7. *Latitude* - This value is constant at 9 bytes (The maximum accuracy value)
    8. *Severity* - This field is always one byte long due to it being represented as a short

	Bandwidth		
Packet Size	10kbps	28kbps	256kbps
68 Bytes	19	53	482
56 Bytes	23	64	585
44 Bytes	29	81	745

Table 5: Results for upstream system testing.

9. *Stage Data* - As this test is assuming the packets are all position updates this is one byte long

- From the data above the smallest packet size is (1+8+1+4+1+9+9+1+1) + 9 = 44 bytes. The last nine is added because of the separating | characters.
- *Largest Packet* - The largest size packet is calculated using the following values

1. *type* - This field is always one byte long due to it being represented as a short
2. *IP address* - Taken to be 15 bytes based on the longest possible IP address (eg 255.255.255.255)
3. *ID* - The maximum value this can be is 16 bytes as this is defined in the code as the longest ID size
4. *time* - The maximum value for the time is 6 bytes representing hours minutes and seconds
5. *Car Number* - This field is always one byte long due to it being represented as a short
6. *Longitude* - This value is constant at 9 bytes (The maximum accuracy value)
7. *Latitude* - This value is constant at 9 bytes (The maximum accuracy value)
8. *Severity* - This field is always one byte long due to it being represented as a short
9. *Stage Data* - This field is one byte for the same reason as in the smallest packet

- From the data above the largest packet size is (1+15+16+6+1+9+9+1+1) + 9 = 68 bytes. The last nine is added because of the separating | characters.
- *Average Packet* - This is simply the average of the two extreme packet sizes described above which is equal to 68+44 / 2 = 56 bytes

Table 5 shown below illustrates how many clients the system can support for various bandwidths and the average packet sizes described above, this information is also shown graphically in Figure 21.

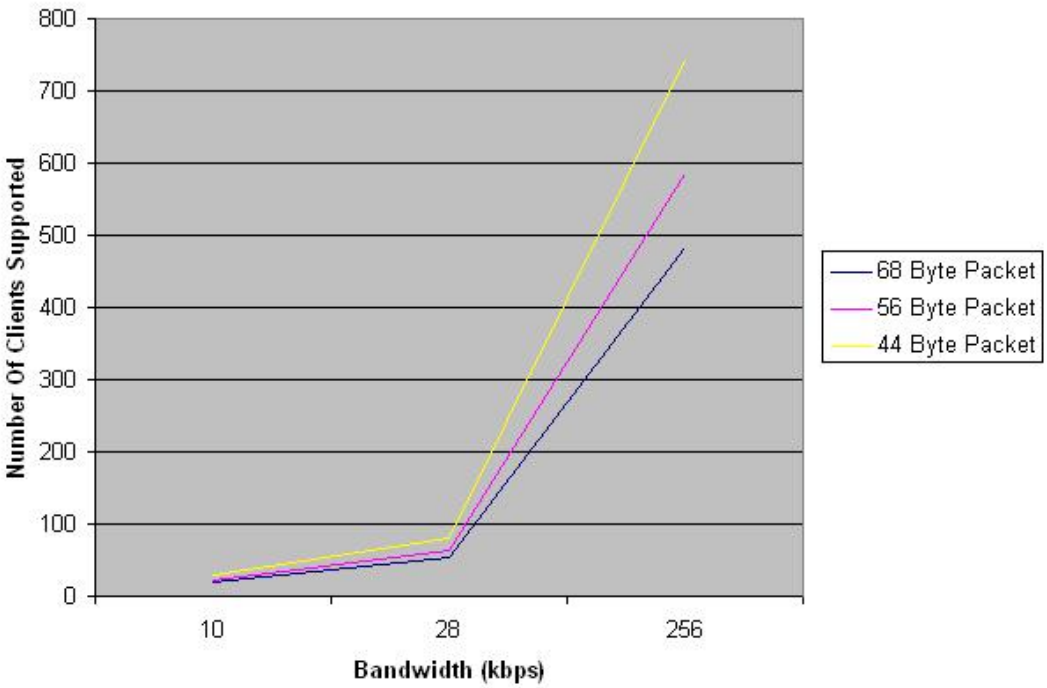

Figure 21: Graph of upstream results.

7.3.2 Downstream

The calculations for how many cars the system could support based on the downstream were carried out using six values for available bandwidth (10, 20, 30, 40, 50 and 128 the first five are values that could be available from a GPRS connection and the six is from a typical ADSL line) and three values for the packet size (smallest, largest and average), explained below

1. *type* - This field is always one byte long due to it being represented as a short
2. *IP address* - For the same reasons as the upstream test for the smallest packet this is set to 8 bytes
3. *data* - This is taken to be 9 as it is a GPS coordinate in a position update
4. *data2* - This is taken to be 9 as it is a GPS coordinate in a position update
5. *data3* - This is taken to be 1 as it is set to 0 in a position update
6. *Car Number* - This field is always one byte long due to it being represented as a short

From the data above the packet size is (1+8+1+1+1+1) + 6 = 19 bytes. The last six is added because of the separating | characters, this is the value for the smallest packet size, to get the largest value the IP size has to be changed to 15 from 8 for the same reasons as in the upstream tests, this is the only value that needs changing and gives a packet size of 26 bytes. The average from this is 23 bytes. Table 6 shows the results from these calculations and the information is also shown graphically in Figure 22.

	Bandwidth					
Packet Size	10kbps	20kbps	30kbps	40kbps	50kbps	128kbps
26	49	98	148	197	246	630
23	56	111	167	222.6	278	712
19	67	135	202	269	337	862

Table 6: Results for downstream system testing.

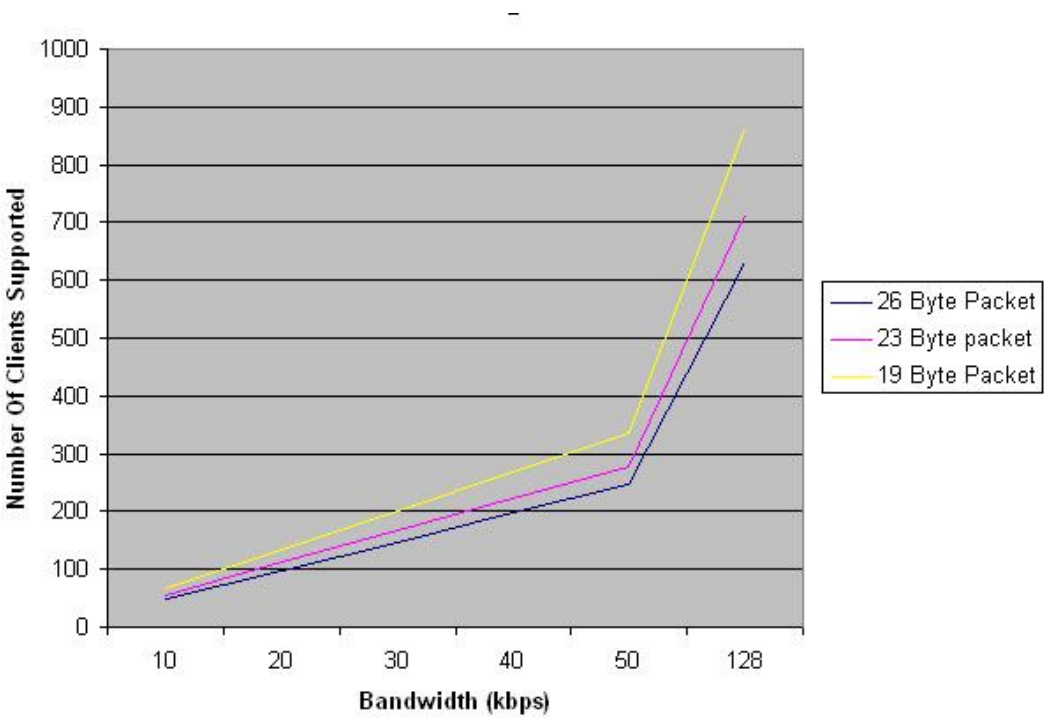

Figure 22: Graph of downstream results.

7.4 Analysis Of Test Results

The test results in the previous two sections show that the number of clients that can be supported by the system increases as the bandwidth does and also increase as the packet size decreases. These results are what would be expected as because as bandwidth increases, the network can handle more data and the lower the packet size, the number of packets that can be sent over it will increase. The most interesting results are how many cars the system can support. The biggest bottlenecks of the system is are concerned with the upstream.

The tests show that the average packet size for the upstream varies from 44 to 68 bytes. These values are much higher than the values for the downstream and this means that the available bandwidth will always be used less efficiently. On top of that the amount of bandwidth available for the upstream will always be less than the downstream, the rates that have been tested (10, 28 and 128kbps) are typical of the type of networks that would be available at the events the CARS system is aimed at. When the packet size is

68 Bytes and the available bandwidth is 10kbps (as could well be the case on a GPRS network) the system can only support 19 clients, despite the fact that at some of the small multi use single venue events the system could be used at this might just be enough to cover the stages (as there are generally only around seven to eight cars on a stage at a time), this would not make the system scalable to larger events.

7.5 System Changes

This section describes any changes that were made to the system from the original design and why.

Originally it was proposed that the server would have all the stage maps associated with it and distribute these to all the clients. Whilst this was technically feasible the strain it put on the system meant that it was very undesirable. This was due to the fact that the bitmaps used to display the stages tended to be large and therefore the amount of bandwidth consumed at each stage change was too much. This would be fine if the server was connected to a wired network, however, at a number of events that the system could be used at this would not be possible and it would be connected via GPRS.

To overcome this it was decided that the stage maps should all be put on to the clients manually and just the co-ordinates sent from the server to the client at the beginning of a rally, this means that the bandwidth used by the system at stage changes is reduced considerably and the network could be used a lot more efficiently. One problem with using this method is that if a stage map has to be changed during a rally for any reason this cannot be done dynamically from the server and all the clients would have to be recalled for the changes to be uploaded.

8 Conclusion

8.1 Summary

The CARS project aimed to test how wireless networks and the Global Positioning System (GPS) could be utilised and combined to provide safety improvements to amateur rallying in the UK. The results that were achieved from this were promising in some areas and not so in others.

The way in which users interacted with the system (reporting accidents, dealing with them etc) and the ways in which the information was displayed back to them (stage maps, locations of cars etc) were met with a general sense of approval from the users that tested the system. However, many of the users who tested the system had little or no background knowledge of amateur rallying and were basing their feedback purely on the interface and interaction aspects. This can be seen as a good thing because it means that the rallying community who could have little knowledge about computing and how systems work should find the system easy to use, the down side to this is that because more people from the rallying community could not be used for testing, the way in which some aspects of the system work may not have been tested with a proper objective view on the most efficient method of relaying all this information. Despite that it is believed that the overall interaction aspect of the project was successful in that it was easy to use and users would not need much prior knowledge of the system to be able to utilise it, the only major flaw from the user testing was the update delay.

GPRS was chosen as the main network the CARS system was to utilise as its wireless network. The implementation of the system (despite only being tested over an 802.11 network) has identified that the clients can manage easily with the limited bandwidth. The server implemented for this system has to deal with a lot more information than the clients due to the fact that it is constantly receiving updates from all cars on a stage and severe limitations were observed from the system testing. Taking the worse case scenario (lowest bandwidth available and maximum packet size on the upstream) shows that only 19 cars can be supported. If the best case scenario for a possible connection over a GPRS link is taken the value is 81 cars which would support most rallies that are run but the 28kbps bandwidth can not be guaranteed. For these reasons it is believed that for practical use the server would need to be plugged into a wired network to be able to support larger rallies.

8.2 Further Work

Another option to overcome the problem with distributing the stage maps (as previously discussed in Section 7.5) would be to use a peer to peer mechanism for distributing the maps. The server could send the map out to a small number of clients that could then create connections with other clients and the maps could then be distributed between the clients rather than from the server to each client. Although the total bandwidth used would not be reduced the bandwidth needed by the server would be. As the server is the main part of the system and requires the most bandwidth to carry out its normal tasks this would create a much better distribution of network traffic and still allow the server to operate efficiently. This was not implemented for the CARS system due to

the complexity of developing a suitable peer to peer system, however if the system is expanded this would be a suitable option if the stage maps have to be distributed from the server.

Allowing co-drivers to retire from an event via their client application would be a desirable addition to the system[4]. This would have to be put into a packet and sent to the server, for this to be implemented the client to server protocol would have to be changed to slightly to add a new "type" value for competitor retirement and the server would have to be able to process this and take appropriate action such as displaying a message box to the user informing them that the car has retired.

To increase the number of cars the system could support, the most obvious way to do this is by trying to reduce the amount of bandwidth consumed by the system. One way of achieving this would be to change the format of the packet, this could be accomplished by using a binary sequence for each packet. By doing this the information contained could be transported across the network in smaller packets. Because the packet size would therefore be less it follows that the total amount of traffic would be reduced.

Another way to reduce the bandwidth used would be by formatting the string for the end of stage report more efficiently, for example the current system sends back a series of one's and zero's, a 1 representing a packet received and a zero representing a packet not received, an example of which can be seen in Figure 23

```
11111110000111111111111111110000111111111111111
```

Figure 23: Portion of a typical string received as an end of stage report.

In this example the string represents that the first seven packets were received (shown by the occurence of seven one's), the next four weren't (shown by the 4 zero's) etc. The size of this string could drastically reduced by using a system shown in Figure 24

```
\17\04\118\04\115
```

Figure 24: Coordinates structure.

This revised pattern consumes much less space (and therefore bandwidth) by using a system of compressing each group of similar digits and tagging them using a "\". The first number after the backslash represents whether the sequence is made up of one's or zero's and the numbers after that up until the next "\" represents how many of occurrences of that number there are in sequence. In the above example the first string is 48 bytes and the second is 17, this represents nearly a 65% saving in bandwidth. This value is by no means definitive for the average saving this system of tagging would provide, however as the pattern for this system is likely to have patterns containing large amounts of repeated sequences this system should provide relatively high bandwidth savings.

[4]Although not implemented at the time of writing, this functionality should be implemented in the near future

As mentioned in Section 6.2.3 the fairness of a stage needs to be tested to check that no competitors have received a significant competitive advantage over others through use of the system. The way that has been devised to check this is described below.

Suppose a packet such as that shown in Figure 23 was received from a client at the end of a stage. Analysing the whole string would not be sufficient as the distribution of the packets received is of great importance. Determining a statistic for the packet distribution could be accomplished by taking the string four bytes at a time, if these bytes are all 1 then the statistic would be 100% for these four packets, if there were 3 ones it would 75% etc. The start of the window is then moved along one byte and another statistic calculated, the average of these statistics is calculated and remembered. The window is again moved on one byte and statistic taken again, then averaged with value in memory. If this is done all the way along it will create a fairness value for the string. For example a string of "1111100000" would come out with an average fairness roughly equal to 11.7 whereas the string "1010101010" would receive a fairness value of 50, this shows that the second string provided a much better distribution of packets throughout. The fairness values retrieved by all clients could then be compared against each other using a system such as a normal distribution to see if any cars gained a competitive advantage.

8.3 Final Remarks

Implementing this system has provided some interesting insights into how a system such as this could (or in the case of the server could not) totally be implemented over a GPRS network. Using GPS to plot the co-ordinates of multiple moving objects and keep track of them at all points in the system was non trivial and provided some interesting challenges with regards to the network and efficient parsing of information, as well as constantly monitoring for accidents and keeping everyone informed of the stage status. As a prototype for this type of system it mostly fulfilled its goals and showed the limitations of a wireless network such as GPRS for some applications.

References

[1] Peter H. Dana. Global positioning system overview. http://www.colorado.edu/geography/gcraft/notes/gps/gps_f.html.

[2] Motor Sports Association. Home page. http://www.msauk.org/.

[3] Tony Newsum. Msa competitors yearbook. Royal Autombile Club Motor Sports Associaten Ltd.

[4] Microsoft Corporation. Embedded visual tools 3.0. http://www.microsoft.com/downloads/details.aspx?FamilyId=F663BF48-31EE-4CBE-AAC5-0AFFD5FB27DD&displaylang=en.

[5] Microsoft Corporation. Activesync 3.7.1. http://www.microsoft.com/windowsmobile/downloads/activesync37.mspx.

[6] Microsoft Corporation. Development tools for mobile and embedded applications. http://msdn.microsoft.com/library/en-us/dnppcgen/html/mobdevtools.asp.

[7] Sleepycat Software. Berkeley database. http://www.sleepycat.com.

[8] IEEE. 802.11 - wireless lan. http://grouper.ieee.org/groups/802/11/.

[9] Gunnar Karlsson Hctor Velayos. Techniques to reduce the ieee 802.11b handoff time. http://winternet.sics.se/workshops/sncnw2003/proceedings/30T-Techniques%20to%20reduce%20IEEE%2080211b%20handoff%20time.pdf.

[10] GSM World. What is general packet radio service. http://www.gsmworld.com/technology/gprs/intro.shtml.

[11] Magnatec Corporation. What is gsm? http://www.magnatec.de/gsm_text_e.htm.

[12] Dale Callahan. Wireless communications and networks. http://www-ece.eng.uab.edu/DCallaha/courses/wireless/CHAP10.ppt.

[13] University of Southern California. Rfc 791 internet protocol. http://www.faqs.org/rfcs/rfc791.html.

[14] Cory Vondrak. Remote procedure call, software technology roadmap. http://www.sei.cmu.edu/str/descriptions/rpc.html.

[15] Bob Quinn. Winsock 2 information. http://www.sockets.com/winsock2.htm.

[16] Unknown. Port numbers. http://www.iana.org/assignments/port-numbers.

[17] Lancaster University. Campus map. http://www.lancs.ac.uk/travel/campusmap.htm.