WIRELESS COMMUNICATION NETWORK IMPROVEMENTS FOR THE UIUC ROBOTICS LABORATORY TESTBED

BY

FREDDY GONZALO PESANTEZ DIAZ

Ingeniero Eléctrico, Universidad de Cuenca, 1997
Magister en Gerencia Empresarial, MBA, Escuela Politécnica Nacional, 2005

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana–Champaign, 2009

Urbana, Illinois

Adviser:

Professor Mark W. Spong

ABSTRACT

This thesis introduces some improvements to the robotic testbed in the Robotics Laboratory in the Coordinated Science Laboratory at the University of Illinois at Urbana-Champaign. The main goal was to improve the wireless network. Not only this goal was successfully achieved, but the complete tesbed was also improved with the introduction of new hardware and software.

The challenges of this research were twofold: first, it was necessary to keep functionality of the previous platform, and second, it was desired to maximize the performance of the new system. The legacy serial port used in the existing platform presents a serious constraint in terms of the performance of the system. Two main implementations are proposed: The first one is completely "transparent" to the user, meaning that the experiments can be run without the user noticing any change. This implementation can be used even with the same software that is running currently. The second implementation changes the serial port at the side of the control computer with a more efficient Ethernet port. This implementation demands a change of software in the control computer. This software was made in Labview and was developed to allow the same functionality as before but with improved communication performance.

The system achieved should not be considered merely as an improvement in wireless communications. Rather, it is a new logical layer introduced between the control computer and the robots, which will diversify enormously the experiments that can be performed in this new testbed.

ACKNOWLEDGMENTS

First, I would like to thank my advisor, Professor M. W. Spong, for his support. Thanks are also due to the staff at the Coordinated Science Laboratory for their friendship and help.

Last but not least, I would like to thank Sandra, Juan Fernando and Ana Paula for everything, especially for being my inspiration during this time.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Distributed control algorithms for networked systems rely on good communication systems. A better communication system often allows better performance with simpler and more reliable distributed control algorithms. In this thesis we develop a new wireless communication interface for control of multi-robot systems in the Robotics Laboratory in the Coordinated Science Laboratory at the University of Illinois. The current communications protocol for the wireless network used in this laboratory works only in broadcast mode from the master computer, which limits its range of possible applications. The current system receives data only on the serial port. Moreover, the wireless transmission is simplex (one-way only); it is limited to receiving and transmitting raw data only and cannot perform operations on the data. Consequently, the entire computational burden for data processing and distributed control is carried out by the digital signal processing (DSP) board on the robots. The introduction of suitable new hardware for communications will therefore yield the double objective of improving wireless communications and adding extra processing capacity to the multi-robot network.

1.1 Motivation

Decentralized control algorithms increase bandwidth demand on the communications platform. The IEEE802.11 protocol is a good approach for providing increased wireless bandwidth as well as channel control, power control, and other advantages. It is a well-know wireless interface widely used in the industry, it achieves a good throughput and it is simple to configure and understand.

The system proposed here is based on the use of an embedded system that contains an 802.11 interface, which will be used for wireless communication, and an Ethernet port, which will be used for the interface to the master computer. Changing the serial port to an Ethernet port is also an important step to obtain better performance of the tesbed.

The introduction of 802.11 was an important improvement in wireless communications when it was introduced. The protocol incorporates collision avoidance algorithm that provides a basic and efficient way to allow reliable and fast wireless communications.

1.2 Description of the Testbed

The basic tesbed configuration is depicted in Fig. 1.1. There are four computers, each with a dedicated camera located in the ceiling of the Laboratory. Every time one of the computers has data available to be transmitted to the robots, it sends this information to a central computer (computer C in the figure), which is connected to the wireless interface and has a program running that packs the information with the proper headers and broadcasts it wirelessly to the individual robots.

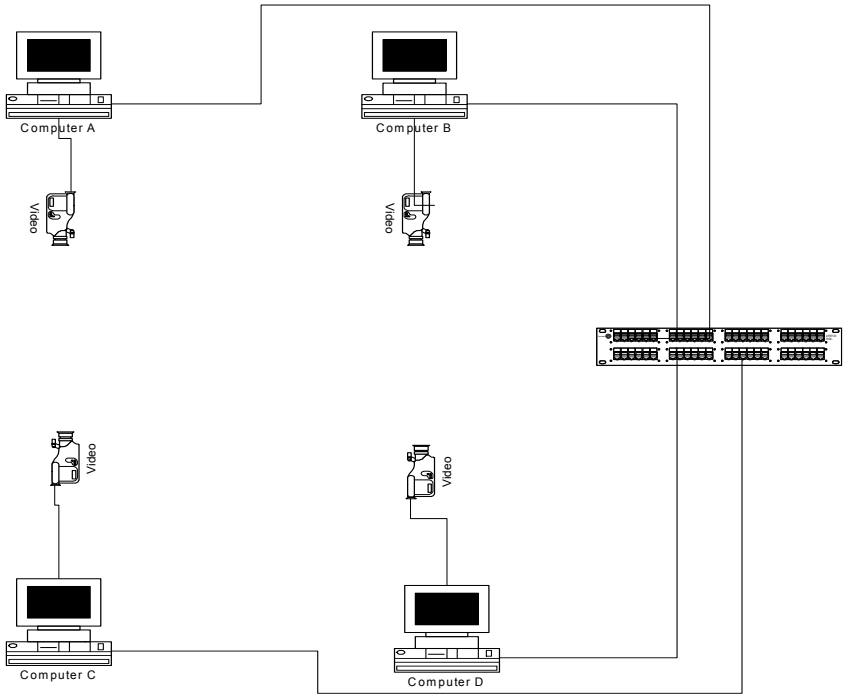

Figure 1.1: Basic testbed configuration.

1.3 Elements of the System

This section describes the hardware and logical elements used in the project.

Hardware: The main element introduced is an embedded computer called Gumstix. Gumstixes are embedded computer systems running a reduced version of Linux (Open Embedded Linux). The processor is a Marvell PXA270 with XScale, speed 400 MHz, 64 MB RAM and 16 MB Flash. It has various interfaces that can be configured according to the application. For our specific application, the Gumstix is been equipped with an 802.11 interface, three serial ports and one Ethernet port. The small size and low power requirements of the Gumstix make it suitable for mounting on the robots.

As for the logical components, it is helpful to examine them in terms of the OSI layers [1]: Application, Presentation, Session, Transport, Network, Data Link, and Physical.

Physical and Data Link layers: Ethernet or IEEE802.11 takes care of the Physical and Data Link layers. Both behave similarly, the first over wired media and the second wirelessly. Both protocols implement a simple and effective collision avoidance algorithm (Data Link layer).

Network layer: At this level, the communication works with IP protocol. Each node needs to be assigned an IP address in order to be able to communicate and be uniquely identified in the network.

Transport layer: This layer uses UDP (User Datagram Protocol). UDP sockets provide logical separation in simultaneous communications, allowing unicast, multicast or broadcast communication, further information about UDP sockets can be found in [2].

One of the most important features of the OSI communications layer protocol is that each layer is independent of the others. This feature is particularly useful in our application as it means that the UDP protocol will work regardless of the physical interface to which it is attached.

Session, Presentation and Application layers: This refers to programs written in C/C++ that run in each Gumstix and are detailed in Section 2.4.

1.4 IEEE 802.11

One of the key changes to the testbed is the implementation of the IEEE802.11 protocol for the wireless link. In this section we highlight various characteristics of this protocal to illustrate the performance improvement over the current protocol. There is an extensive literature about IEEE802.11. Further information can be found in [3], [4] and [5].

IEEE 802.11 defines the Physical layer (PHY) and the Medium Access Control layer (MAC). The first layer specifies the modulation and signal specifications for transmission at radio frequencies. The second layer determines how the nodes access the medium. 802.11 does not provide differentiated services based on traffic types.

In 802.11, a basic service set (BSS) is defined. BSS is used when there are *n* independent nodes without a wired connection and infrastructure BSS when there are *n* nodes and an AP (access point or coordinator unit).

1.4.1 Timing intervals

IEEE 802.11 supports two schemes: Distributed Coordination Function (DCF) and Point Coordination Function (PCF). DCF is associated with BSS and PCF with IBSS.

PCF: This scheme was defined to support time-bounded services. The priority access to the medium by the stations is determined by a point coordinator (PC normally is the access point). PCF has higher priority than the DCF because it can transmit before a DIFS (DCF interframe space). This time is called PIFS. All the stations communicate only through the AP. The relative lengths and positions of the different interframe spacings are depicted in Fig. 1.2.

DCF: Basic 802.11 MAC operates as a listen-before-talk scheme; each station determines when it accesses the medium. Only data packets and ACK packets are transmitted in this mode. DATA packets are of arbitrary length (up to 2304 bytes). For each successful transmission a confirmation (ACK) is sent immediately by the receiver.

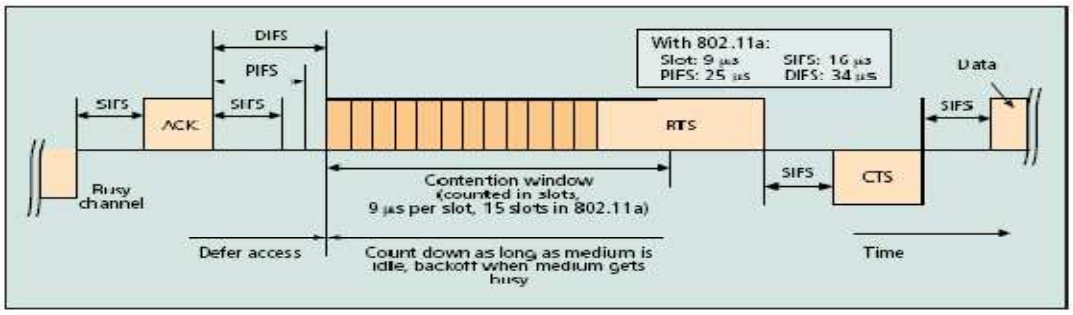

Figure 1.2: Interframe spaces and backoff procedure with random contention window size. Here the transmitting station uses CW=CWmin (15 slots) of 802.11a, and has selected a random backoff time of 12 slots.

RTS/CTS: The Request-to-Send/Clear-to-Send mechanism is used optionally, mainly when large data packets have to be transmitted. This mechanism inhibits transmissions of other nodes until the transmission of one or multiple frames (if the big one is split in smaller packets) of data and ACK are transmitted and received between the sender and receiver. Wireless is a shared media transmission, so for instance in Fig. 1.3, when station A is transmitting, station B must remain quiet to avoid collisions. In this way the so-called *"hidden node problem"* is avoided. It is important to note that SIFS is shorter than DIFS, which always gives CTS responses and ACK packets highest priority for access to the wireless medium.

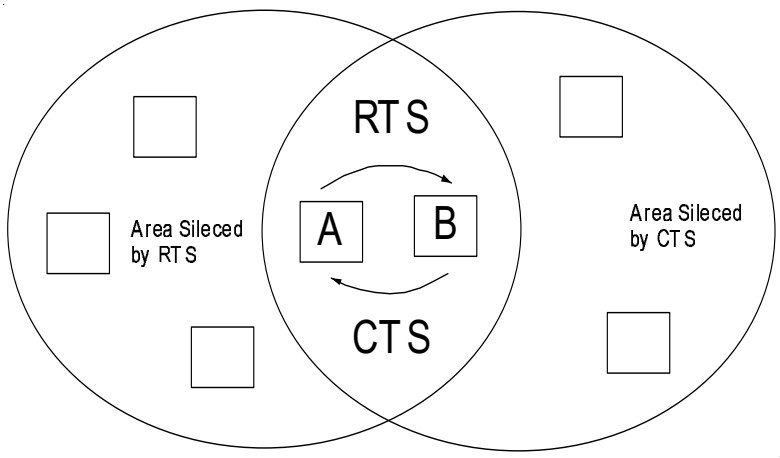

Figure 1.3: The RTS/CTS exchange silences two neighborhood: those of A and B.

1.4.2 Virtual carrier sensing

DCF uses a virtual carrier sensing (NAV) mechanism, which is used by both the basic access method and the RTS/CTS access method. A node can assume that a channel is busy if the physical carrier senses that the medium is busy, or if the NAV is set to a non-zero value. NAV is a register in each terminal that saves the time needed for an ongoing transmission between two other terminals. The information about the time needed is included in the RTS/CTS packets, and then no collisions can be assured because all terminals would listen to any of those two packets in the interference space.

Figure 1.4 shows how virtual carrier sensing works. There are six stations trying to gain transmission privilege; station 6 cannot detect the RTS frame of transmitting station 2, but station 6 receives the CTS frame of station 1. Although station 6 is hidden to station 1, it refrains from medium access because of the NAV.

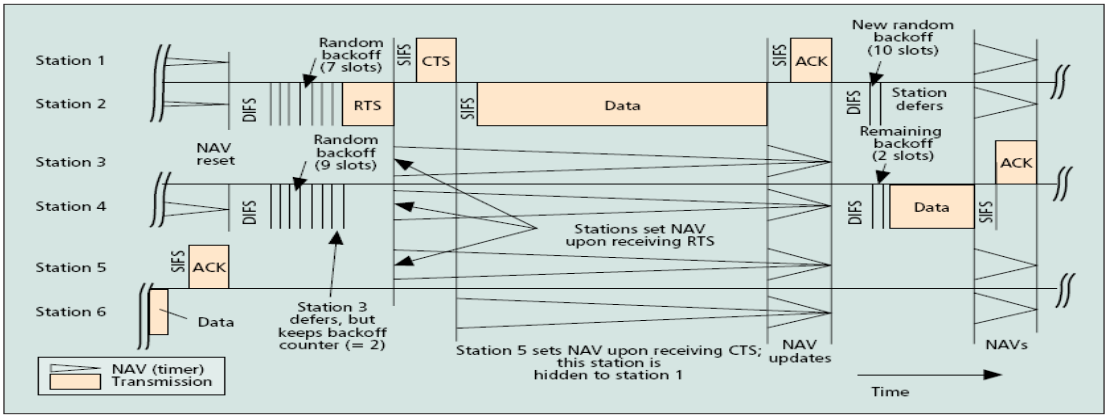

Figure 1.4: Timing of frame exchanges and NAV setting of the 802.11 DCF.

1.4.3 Collision avoidance

DCF is based on carrier sense multiple access (CSMA), in which every node "detects" the channel for a certain time and begins sending a message only if, at the end of this time, there was no transmission. To diminish the probability of collision, 802.11 has a collision avoidance (CA) algorithm. The basic idea is that, after the station detects the medium idle for a certain time (DIFS), it performs a so-called backoff procedure. Every station has a backoff counter and a backoff stage. The maximum value the backoff counter can take is the contention window (CW).

Backoff counter = Int (CW * Random ()) * Slot time.

If a station senses the medium occupied by another station, then the backoff counter is stopped until the medium is idle again for a DIFS period of time and continues counting.

Because of the uncertain nature of the channel and the nature of the protocol (CSMA/CA), a station that has transmitted a MAC protocol data unit (MPDU) needs to know if the destination received the MPDU message; therefore this protocol uses ACKs. As soon as the destination station gets the message, it sends an ACK to the source station. If the source station does not receive the ACK response within a prespecified time, the source station concludes there was an error and may transmit the MPDU again.

If there is a collision or a lack of ACKs received in the source for any MPDU, then congestion is inferred. In this case, the contention window (CW) is increased by 1, so the probability of collision decreases. CW must be between 0 and CWmax. When CW is increased, the probability of collision decreases, but the waiting time increases, so it is necessary to adapt the CW in such a way that the delay will not increase excessively while the probability of collision stays low.

After any successful transmission, each station starts a new backoff process (randomly selected and downcounting). This process is started even if there are no more MPDU to be delivered. This is often referred to as a post-backoff. The post-backoff process ensures that there is always at least one random backoff process between two consecutive frames interchange.

All the parameters described above can be changed and configured to change the communications characteristics.

1.5 Linux Hardware Configuration

One of the aspects that makes Linux is a powerful operating system is that it is easy to configure and interacts well with the hardware. In this specific application the hardware systems used and configured in Linux are: Ethernet interface, wireless interface and serial port.

There are two implementations for this application; basically they differ in how the serial port is used.

1.6 Serial Port

Serial port programming is important for this implementation because it is used in one of the implementations at the master Gumstix side, and both implementations need the serial port to read and write to the robot serial port. The serial port allows two kinds of flow control: software and hardware. Software flow control is used in this application. The ASCII character 253 is always assumed as the beginning of a transmission frame. Frames have different sizes. The configuration of the serial port used is: speed 57 600 bps, 8 bits, non-parity, 1 stop bit.

1.7 Scripting in Linux: Startup of the System

A shell program is a series of Linux commands. Shell script is just like a batch file in MS-DOS but has more power. The shell script can take input from the user or from a file and output it to the screen, or to a file to execute the commands sequentially. Shell scripts can also create user-defined sets of commands that can save time and automate routine tasks. Further information can be found in [6].

The shell script runs at the startup of the Gumstix and is in charge of configuring the interfaces and running the programs needed for the application.

CHAPTER 2

IMPLEMENTATION

All the elements mentioned in Chapter 1 are used to develop the platform. These elements present advantages and restrictions in some cases that have to be considered in the design process of the platform.

Two implementations were performed. The first one uses serial ports at both ends of the platform and is modular in the sense that it allows easy access for implementing modifications or algorithms. The second one uses an Ethernet port at the control computer side and, in general, to access the platform it is necessary to modify the program code. The second implementation is basically designed for experiments where the communication is not relevant.

2.1 Design Consideration (Asynchronous Communication)

Two of the most important aspects to keep in mind during the design process are the communications speed and reliability, in that order. The packets transmitted to the robots come in sequence and are being actualized continuously so that, if one packet is missing for any reason, the robot is still able to receive the next one. As long as the packet loss is below certain a threshold, the control algorithm performance is not diminished.

2.2 TCP or UDP

If transmitting fast is one priority of the system, then UDP is the first option for wireless data transmission. There is a lot of research [3] showing that the performance of a TCP links drops significantly, mainly because of the inability of TCP to differentiate the cause of the packet loss. TCP was created basically for wired networks. In wired networks the main cause of packet drops is congestion. When TCP finds congestion, it reduces its transmission window dramatically and remains in this reduced transmission stage for a relatively long period of time. This phenomenon reduces significantly the overall throughput. This procedure makes sense in a congested network by yielding fewer transmissions attempts by each station, partially alleviating the problem; however, if the packet drop is not caused by congestion, throughput reduction turns out to be excessive.

In a wireless network the transmission medium is shared and is dependent on a lot more parameters than in a wired network (interference, noise, etc). If this uncertainty in wireless networks is interpreted as a congested network, then the final throughput is reduced significantly and unnecessarily. There are various attempts to solve this problem through both minor and major modifications to the basic TCP protocol. However, these modifications to TCP are not available in the Gumstix, so we will use UDP in our application here.

As an "unreliable" transmission protocol, UDP does not guarantee performance of the end-to-end communication. Rather it is a "best-effort" protocol, meaning that if one packet is dropped for any reason, it is simply missed. UDP relies on upper-layer protocols to check communication reliability.

2.3 Multitask OS

The Gumstix embedded computer runs Linux Open Embedded as its operating system. Linux OE is becoming a standard for most embedded systems, because it is highly modular. The user can load the applications needed and avoid overloading the processor with useless or unnecessary parts of the operating system for some specific application. Another useful characteristic of Linux is that it is a multiuser and multitask operating system.

2.4 Software

Figure 2.1 shows the topology of the Gumstix mounted on the robots. At the master Gumstix side, there are two software schemes proposed, based primarily on the type of interface used:

Using serial port, which is called implementation 1

Using Ethernet port is called implementation 2

2.4.1 Implementation 1

One Gumstix is going to be placed in each robot and another acts as the master for the communications (refer to Fig. 2.1). The combination of the existing robots plus the Gumstix is called a new robot because of the new characteristics available in each robot and for differentiation.

The system implemented should not been taken only as an improved communication system. Actually the platform should be considered as the introduction of a new logical layer that provides extra processing capacity to the robots, making it easier to implement distributed and networked control algorithms with them.

The Gumstix increases the available processing capacity and is relatively easy to program using C/C++. Initial programming presents some difficulties because it has many steps and the information is spread out among many places, but this information has been collected and is presented in the Appendix of this thesis. In order to install C/C++ it is necessary to generate a metafile with a script language and use secure copy protocol (SCP) connection to download the program to the Gumstix.

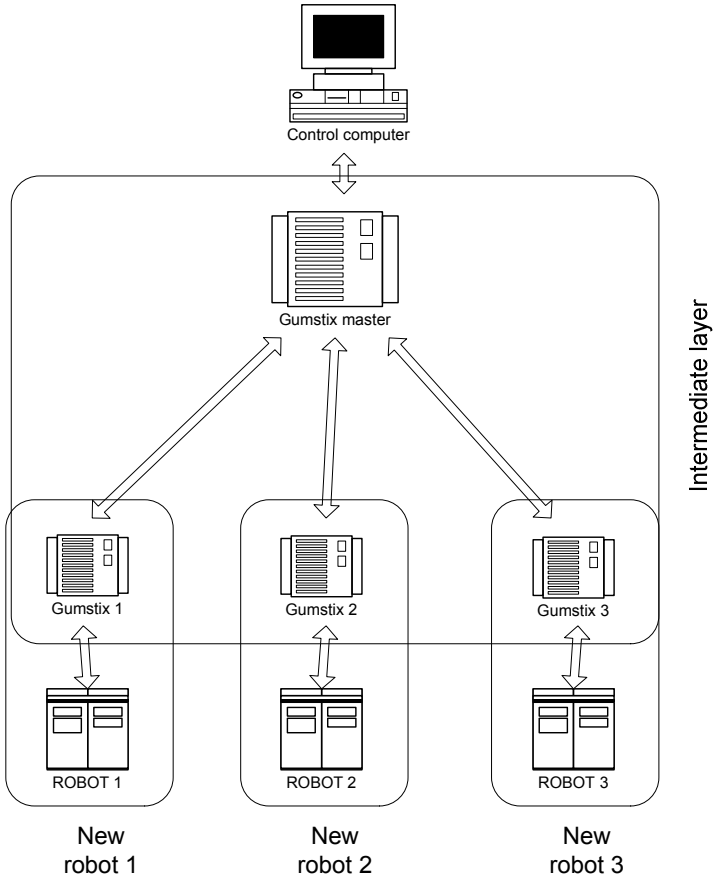

Figure 2.1: Modification proposed to the wireless communication using the Gumstix.

The main idea was to find a way to exploit the multi-tasking capability of the Linux operating system. This characteristic was exploited by making several small programs that run in parallel instead of only one big program. This modularity makes it easy to

modify the structure and add software needed for performing tasks according to some control algorithm.

The Gumstix communicates with the robot using the serial port at 57 600 bps. In the future the DSP boards will be replaced by new boards now available on the market, which will have Ethernet interfaces. The software has been developed in anticipation of this change in the near future, so it will be completely compatible with these changes in hardware. The change to an Ethernet interface will improve dramatically the performance of the communication. The serial port is not only relatively slow but also presents a lot of problems using buffers and is really inefficient. In fact, the biggest problem faced during the software implementation was dealing with the serial port.

There is also a drawback associated with a multitask processing operating system: it is impossible to know exactly which program or task or process is executed first and which one later. The software must assure a way to take care of this problem. It was solved using a buffer in every step to store the information until it can be transferred to the other end. The buffers not only synchronize the processes but also carry out inter-process communication among programs. Each shadowed block in Fig. 2.2 represents a small program. All programs run simultaneously and use the files robot1.txt, robot2.txt, Master1.txt, and Master2.txt as buffers to allow asynchronous communication. The information is generated at one end; let us assume it is the control computer (refer to Fig. 2.3). The computer sends the information through the serial port. The Gumstix master reads the serial port and saves the message in the buffer temporarily (robot1.txt).

At this point the buffer is split in two: robot1.txt and robot2.txt. If any process attempts to affect the data by any control algorithm, it is easy to take the information from robot1.txt, perform whatever the user wants to do and save it in robot2.txt. Then the UDP socket will take robot2.txt and send it to the wireless interface. Currently data are just being copied from robot1.txt to robot2.txt, but the change can be implemented very easily.

The wireless link uses IEEE 802.11, which takes care of the collision avoidance thanks to the algorithm intrinsically implemented in this protocol. Queue priorities are easily implemented by making slight changes to the basic IEEE 802.11.

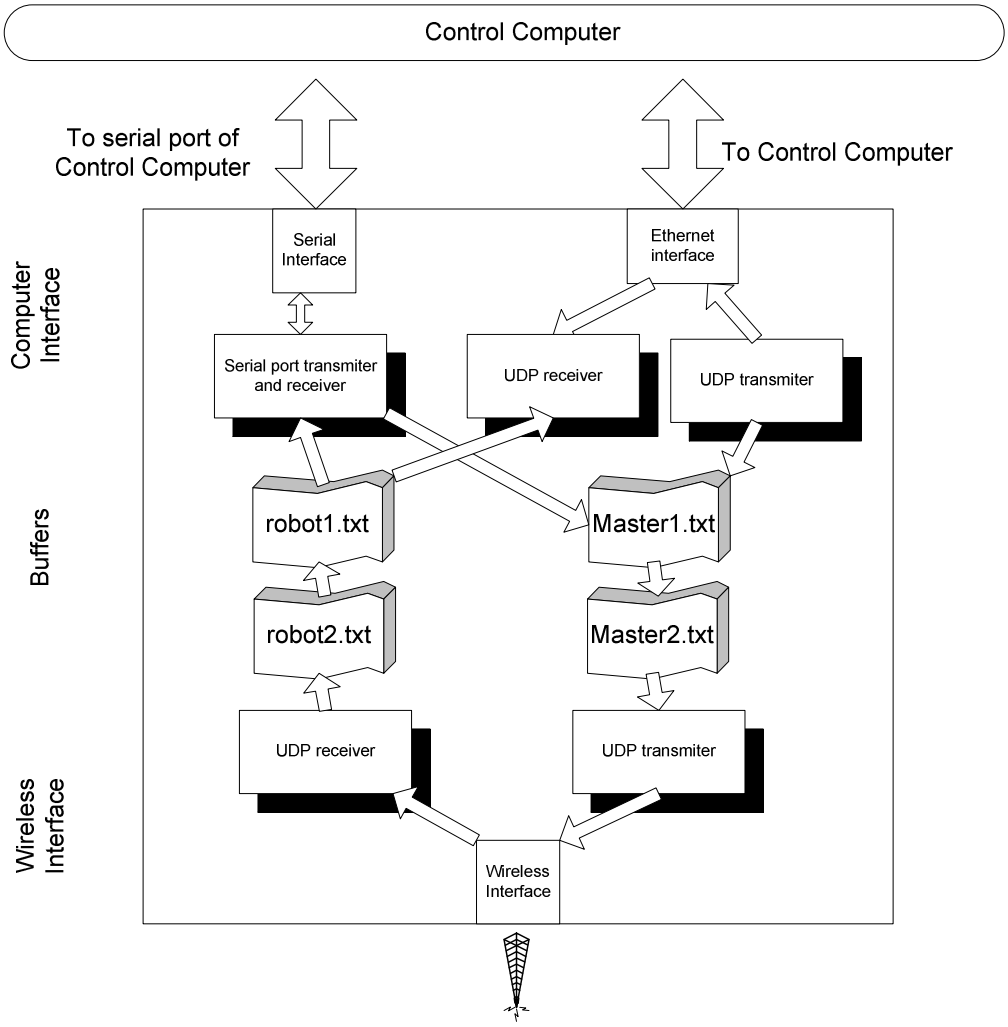

Figure 2.2: Functional diagram of the software in the Gumstix, Implementation 1.

At the other end, the wireless interface of the slave Gumstix receives the information with the UDP receiver program; this information is saved temporarily in the buffer called master2.txt. Here the buffer is also divided in two, making it easy to manipulate the data at the slave side as well; currently master2.txt is copied to master1.txt. Finally, the serial port program manager takes master1.txt buffer and sends it to the serial port of the robot. This concludes the transmission from end to end in one direction.

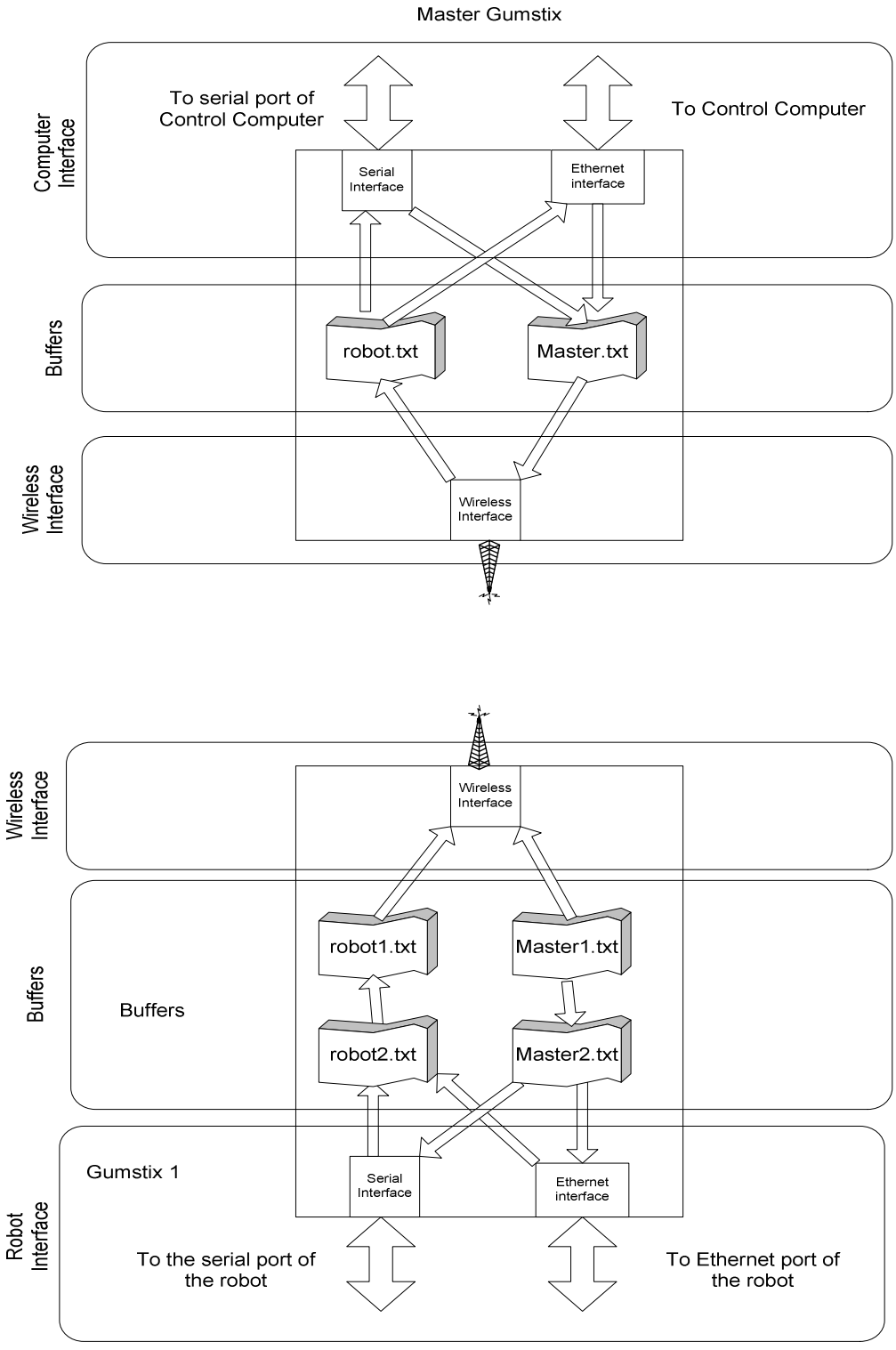

Figure 2.3: Configuration of the wireless network from one end to the other.

The software written for both Implementation 1 and Implementation 2 is briefly described below.

UDP listen: There are two UDP listening sockets. The first takes care of the incoming packets from the wireless communication. The second UDP is reserved for the future implementation of an Ethernet port in the DPS board.

UDP sending: This program is similar to the UDP listen: one for the wireless communication sending, and one for future implementation.

Because UDP allows multiple sockets to run in the same communication channel, there are various logical links using different UDP ports in the same wireless communication channel.

Serial port send and receive: This program receives and sends through the serial port.

2.4.2 Implementation 2

Implementation 2 uses the Ethernet port for the interface with the control computer (Fig. 2.4). It communicates with a Labview program running on that computer, which is also a modification to the platform. The programs running in the second implementation are:

Udpreceive: This program receives UDP packets for the Ethernet interface and sends them wirelessly thought the wlan0 (wireless Lan 0) interface.

Gumserial41: This program sends information between the wireless interface and the serial port. The process is repeated at both sides, having an overall behavior identical to Implementation 1.

The communication system is flexible enough to have two quite different behaviors: it can work as a black box, completely transparent to the user, or it can be modified very easily, depending on the user needs.

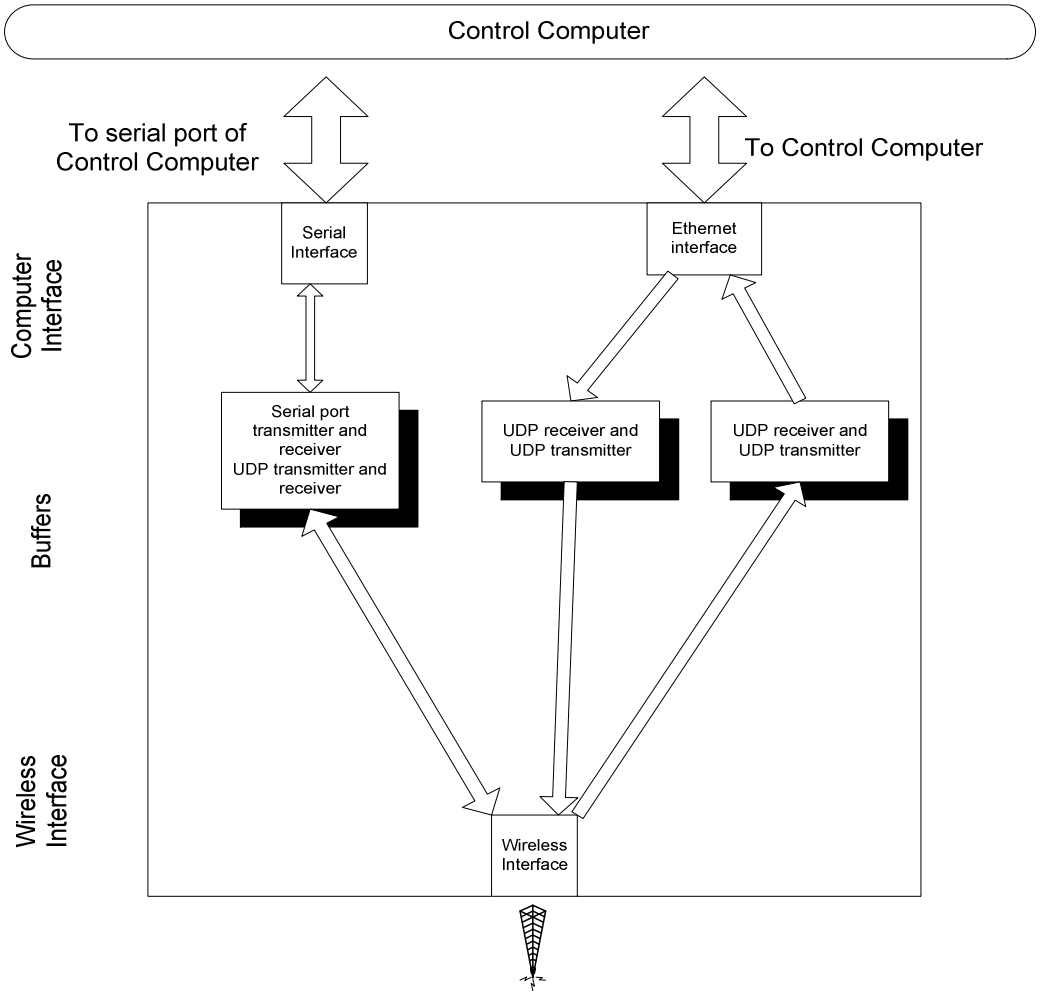

Figure 2.4: Functional diagram of the software in the Gumstix, Implementation 2.

2.5 IP Assignment

UDP is used as the Layer 4 communication protocol. UDP relies on IP as a Layer 3; this means an IP address needs to be assigned in order to be able to route the packets from source to destination. The proposed IP assignment is depicted in Fig. 2.5.

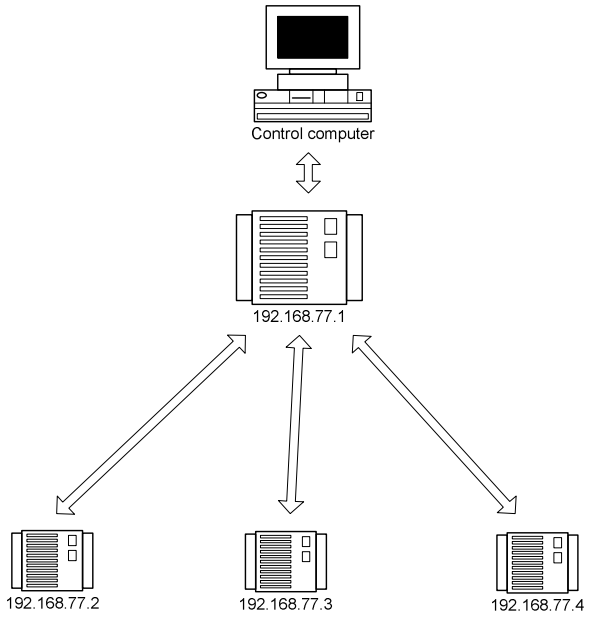

Figure 2.5: IP address assignment.

Static IP assignment was chosen because the number of robots is limited, so it is not necessary to have a big pool of IP addresses; however, if for any reason the user wants to experiment with dynamic IP address assignment, protocols like DHCP are easily implemented, as in any other Linux platform.

2.6 Modifications to Other Elements of the Testbed

As mentioned previously, the first implementation uses the serial port at the Gumstix master end to interface the control computer. This option uses the same software that is running currently. The second option uses an Ethernet interface, so it needs a software upgrade to send information to the Ethernet port instead of the serial port. The Ethernet port has a lot more bandwidth and also uses the transmission and reception buffers much more efficiently. The software implementing this upgrade was written in Labview. The images generated in the four cameras in the ceiling of the laboratory are processed on four computers and sent in UDP packets. Labview software collects these packets, adds headers and sends them to the Ethernet port of the new communication platform. The procedure that the platform uses to take the information from the Ethernet interface and send it to the other end was described previously.

2.7 Wireless Management in Linux

All the parameters of the wireless communication can be configured in Linux. The basic parameters necessary for communication and those that were implemented are the following:

- Channel used: Channel number 2
- Grupo: “B16”
- A static IP address assignment
- UDP is used in the following way:
  - Broadcast transmission from master to 3 slaves
  - Unicast tranmission from slaves to master

The option of broadcast from slaves is easily implemented. If this feature is implemented the other robots can receive information from other slaves without the intervention of the master.

2.8 Startup Sequence

After the system starts up, a script named “gumscript” is executed. Following are the tasks performed for gumscript:

- Enable wireless interface Channel 2, group called “B16”.
- Asign IP address to the corresponding ports (wireless link wlan0, Ethernet port eth0).
- Execute software needed for communication:
  - Serial port management: Gum
  - Enable UDP wireless sender
  - Enable UDP wireless receiver
  - Enable UDP Ethernet sender
  - Enable UDP Ethernet receiver

Initially, the possibility of using different channels for the communication among the robots was explored, but it turns out that the channel change introduces a huge delay in the communication. If, for some reason, it is necessary to use different communication

channels in the future, it is recommended that two separate Gumstix be used, one for each channel. Otherwise the communication may actually be degraded instead of improved.

CHAPTER 3

CONCLUSIONS

There is an increasing interest in exploring applications where the wireless communications and the networked controls are highly interrelated. More sophisticated network control systems demand more communications resources. One question arises naturally: To what extent can the design of the communication and control systems be done separately? Is there any possibility that combining both systems in an early stage of the development of a system could improve the final result? These kinds of questions can be addressed with the new platform implemented through this thesis.

A few examples of systems as described above and that are currently intensively studied are [7],[8]:

- Intelligent transportation systems (intelligent highways)
- Military systems
- Environmental sampling
- Air traffic control
- Formation control

According to R. M. Murray [7], for a long period of time the abstraction that the controls systems always had a completely reliable communication system, with no delays, worked well; however, this model needs to be adjusted in order to represent the kind of applications currently being developed: “Future applications of control will be much more information rich than those of the past and will involve networked communications, distributed computing, and higher levels of logic and decision making …”. Murray proposes an architecture for networked control systems that is shown in [Fig. 3.1]; this architecture is able to deal with increasing bandwidth requirements and is flexible enough to not force starting a new design from scratch every time a change is needed.
The most relevant aspect to this thesis is that the topology achieved with the modifications proposed perfectly matches the topology in Fig. 3.1 and the criteria pointed out by Murray. The final platform integrates the current DSP board already running in the robot with the Gumstix. The DSP boards will continue taking the sensors signals and

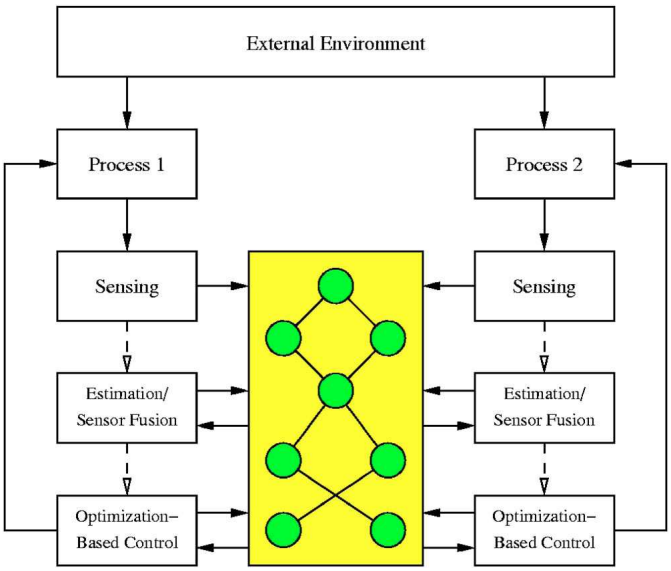

Figure 3.1: Topology proposed for networked control platforms [7].

will pass them to the Gumstix; the Gumstix is able to receive the information from the DSP, perform tasks on the data, and communicate with other Gumstixes or the master in both directions, in a very simple manner as is described in Chapter 2.

To achieve this architecture, IEEE802.11 was used as Layer 2 protocol, IP as Layer 3 and UDP as Layer 4, which becomes a really powerful combination in terms of transmission capacity and configuration setup. In general those three protocols are currently widely used in the industry. In Chapter 2 it was explained why those logical elements were used for this application, but in the future new protocols can be tested and even some new protocols could be created that match better with the application.

IEEE802.11 running on Linux OS is a good combination in wireless communications because of the robustness of this OS and for the simplicity of hardware configuration and operation.

The improved system presents multiple advantages: it is easy to manipulate, easy to understand, and easy to program using C/C++ . The most important of all is that there are many doors opened for future experiments and research.

The possibilities that can be explored with the new platform are many:

- Networked control systems
- Interaction and best routing protocols for multirobots formations

- Effects of delays in networked control systems
- Minimum power transmissions
- Nearest neighbor communications algorithms
- Effects of dynamic changes of topology in networked protocols
- Dynamic IP assignment
- cooperative control
- Distributed autonomous mobile robotics

In summary, the platform has been designed using widely known elements; it is important because it will allow making changes in the platform and implementing networked control systems easily. If the user prefers, he can use the platform in a similar way as it was working previously, without even noticing any change. The previous robots are fully used and the near future changes have been taken into account in such a way that the Gumstix software will not need to be greatly modified, or even not changed at all, with a change of the main board of the robots.

With the changes implemented in this thesis, the same testbed, without dramatic changes, has increased processing capacity, has significantly improved wireless communications, and can now be used in a wide variety of new research fields.

APPENDIX

This appendix summarizes all the configurations and information needed to make the platform work. Most of this information was taken from the manufacturer web site [9]:

A.1 Basic Gumstix Configuration

A.1.1 Setup a serial connection with the Gumstix

Connecting with Kermit

These directions are for the Kermit communications package (cKermit on Linux).

1. Connect the null-modem serial cable to the serial port on your computer and to the central serial port on the Gumstix. (It is assumed the three serial port extension card is used; if not, check the specifications of the expansion card used.)

2. Launch kermit with kermit -l /dev/ttyS0, where /dev/ttyS0 is the serial port you are using on the host computer. It might be necessary to do this as root if regular users can not access the serial port. (Use kermit -l /dev/ttyUSB0, if USB-serial converter is used.)

3. The following configuration works when logging into the serial console:

set speed 115200
set reliable
fast
set carrier-watch off
set flow-watch off
set flow-control none
set prefix all

Add the following if files need to be transferred to the Gumstix (for example, when replacing the filesystem image):

set file type bin
set rec pack 4096
set send pack 4096
set window 5

If USB-serial converter is use, it might be necessary to add the following line in order to make the file transfer work:

set transfer protocol zmodem {rz} {rz} {sz –w 10000 %s} {sz –w 1000 –a %s} {rz} {rz}

Maybe these lines can be added to a file called .kermrc in order to avoid typing it every time the Gumstix is accessed.

The set line /dev/ttyS0 can be added to .kermrc, but be warned that set line may fail silently. If kermit reports an error of "?SET SPEED has no effect without prior SET LINE" but .kermrc sets the line with set line /dev/ttyUSB0, the cause could be that the line is invalid. For example, if Gumstix appears at /dev/ttyUSB0, but it is connected to /dev/ttyUSB1, "set line" fails silently, and you get the error above. "Set line" may also fail because the device does not exist yet; USB devices are created when the device is plugged in.

4. Connect to the port by typing "connect."

5. Plug the power adapter into any power jack on the Gumstix (there might be various). When connected and powered, a message from U-Boot appears followed by the normal Gumstix boot sequence.

6. Log in for the first time with username "root" and password "gumstix."

7. When finished, return to the kermit prompt by typing [CTRL-\] then pressing "c". Typing "quit" will exit the program.

A.2 Bitbake

Bitbake is a metafile used to generate the installers of the programs needed to run in the Gumstix. Bitbake is used on a Linux computer; once the installers are generated, these are sent to the Gumstix with can SCP connection.

The following is a sample of a Bitbake recipe:

Bitbake recipe:
user.collection
|

```
`-packages
 |
 `-helloworld
   |
   |-files
   | |
   | `-hello.c
   |
   `-helloworld_1.0.0.bb
```

$ bitbake helloworld

A.3 Transmission of the User Programs to the Gumstix Using SCP Connection

First IP addresses must be assigned to the Gumstix which is going to receive the program and the computer with the software in such a way that they are communicating (it can be tested using the ping command). After that, the following command has to be typed:

$ scp ~/Gumstix/Gumstix-oe/tmp/deploy/glibc/ipk/armv5te/helloworld_1.0.0-r0_armv5te.ipk root(at)192.160.0.2:/home/root

This example uses helloworld as the program to be transmitted to the Gumstix with IP address 192.160.0.2. The program will be copied in /home/root.

A.4 Installation of the User Program in the Gumstix

The following commands will install and run for first time the program helloworld:

```
$ cd /home/root
$ ipkg install helloworld_1.0.0-r0_armv5te.ipk
$ hello
hello, world
$
```

A.5 How to Enable the Libraries of C++ in the Gumstix

Gumstix OE does not support by default the C++ compiler, if the user wants to use C++ as the programming language, the corresponding libraries must be installed with the procedure detailed below.

There are several solutions:

1) Create a custom image with your application, and libstc++ will automatically be added to the image (http://bec-systems.com/web/content/view/79/9/).

2) Install libstdc++ over the network using opkg:

opkg update

opkg install libstdc++6

This requires that an opkg feed to be set up for your distribution.

3) Copy the packages from your OE build dir to your target and install.

buid dir> scp <tmpdir>/deploy/../ipk/libstc++6...ipk root@<my Gumstix ip>

Gumstix> opkg install libstc++6...ipk

scp /home/freddy/Gumstix/Gumstix-oe/tmp/deploy/glibc/ipk/armv5te/libstdc++6_4.1.2-r10_armv5te.ipk root@192.168.78.X:/home/root

A.6 Hardware Configuration

The following is the hardware configuration in Linux:

- lo no wireless extensions
- eth0 no wireless extensions
- wlan0 IEEE 802.11b+ ESSID:"B16"
  Mode: Ad-hoc Frequency:channel 2
  Bit Rate:100 Mb/s Tx-Power:18 dBm Sensitivity=187/255
  Retry min limit:7 RTS thr:off
  Power Management:off
  Link Quality=54/100 Signal level=39/100 Noise level=1/100
  Rx invalid nwid:0 Rx invalid crypt:0 Rx invalid frag:0
  Tx excessive retries:0 Invalid misc:0 Missed beacon:0

The above configuration can be done with the following command:

```
$ sudo iwconfig wlan0 mode ad-hoc essid "B16"
# Wait 5 seconds to complete channel scan. Then *force* to channel 2
$ sudo iwconfig wlan0 channel 2
```

This completes the Layer 2 setup for the wireless card.

Next, IPv4 addresses have to be assigned to each port used. Each node has to use a different IP address to be able to communicate.

For the master Gumstix node the IP address would be 192.168.77.1. Execute the following command, replace X with the node number, ie. from 1 to n:

```
$ sudo ifconfig eth0 192.168.77.X
```

It is strongly suggested that if some problem occurs in the communication, the first thing to do is verify ping connectivity.

A.7 Compile .cpp Programs Using g++ with Real-Time Timers

If you use gcc, just make sure you add -lrt to your list of arguments. The software uses a real-time timer as a watchdog to avoid infinite waiting for response in the serial port, so it is necessary to compile Gumserial4.cpp and Gumserial41.cpp using the option –lrt.

A.8 Setting Up a Build Environment

This information was taken from the Gumstix web site [8]; it is copied here for completeness and because there is an upgrade in the Gumstix computer hardware and software. The following information is no longer available at the web site.

This procedure is to be executed on a computer running Linux, and is necessary only the first time in order to build a programming environment.

First, source files have to be checked out for the Gumstix OpenEmbedded (OE) build system. With a typical DSL connection this step should take about 20 minutes and requires at least 10 GB of free space on hard drive.

A.8.1 Prerequisites

The build machine should be running a fairly recent Linux distribution. The following distributions are known to work:

* Ubuntu 7.04, 7.10
* Fedora 6, 8
* Debian
* SuSe
* Gentoo
* Centos 5

Depending on your distribution, you will need to install additional software. The packages needed are:

* gcc
* patch
* help2man (Centos 5 package available from atrpms repository)
* diffstat
* texi2html (texinfo on SUSE)
* makeinfo (texinfo on Ubuntu)
* ncurses-devel (libncurses5-dev on Ubuntu)
* cvs
* gawk
* python-dev
* python-pysqlite2 (python-sqlite2 on SUSE)

The system will prompt for the missing packages.

Note: When using an Ubuntu distribution, it is likely that /bin/sh is linked to /bin/dash. If this is the case, then it will need to change /bin/sh to link to /bin/bash. Neglecting to do this will cause file corruption and the build image will not boot! Run "sudo dpkg-reconfigure dash" and answer "no" when asked whether you want to install dash as /bin/sh.

Source checkout

The only package that is absolutely required to get started is the subversion source code control package. Check to see if your machine has a working copy of subversion:

```
$ svn --version
```

If the version information is printed, that is fine; if a "command not found" message appears, a subversion package for the distribution used has to be installed.

Check out the source files for the Gumstix build system:

```
$ mkdir ~/Gumstix
$ cd ~/Gumstix
$ svn co https://Gumstix.svn.sourceforge.net/svnroot/Gumstix/trunk Gumstix-oe
```

After this command completes, the Gumstix build system source code will be in the home of the user directory in Gumstix/Gumstix-oe.

Environment setup

Gumstix OE requires some environment setup in order to function properly. There are a couple of ways to handle this requirement.

Option A: bash profile

It is most convenient (and highly recommended) to set this up via your bash profile:

$ cat Gumstix-oe/extras/profile >> ~/.bashrc

With most distributions, adding this environment setup to ~/.bashrc will work for both GUI shells and login shells.

Option B: Command line sourcing

This is done on a per-session basis by sourcing extras/profile prior to issuing any build commands:

$. ~/Gumstix/Gumstix-oe/extras/profile

or alternatively:

$ source ~/Gumstix/Gumstix-oe/extras/profile

Source code caching

As part of the build process, Gumstix OE downloads source code tarballs for the Linux kernel and other software packages. This next step sets up a global system cache for these tarballs so that they only need to be downloaded once. Setting up a global cache may seem like a lot of trouble for a single-user system, but it is often quite useful to set up an autobuilder that runs as a cron job under a separate user account. Occasionally it may even want to do experimental work under a seperate user account. Setting this up now will save disk space and download time later.

On most systems it will need to have root privileges for this step. The process below uses sudo; if your current user is not in the sudoers file, you will get an error message. If this happens, use su and enter the root password before issuing the commands below.

First create a new group called oe and add your login account to the oe group. In the second command substitute login user name where your_username is indicated.

$ sudo groupadd oe

$ sudo usermod -a -G oe your_username

Next we create the directory for the source code cache, set the group owner to the newly created oe group, and finally set permissions on the directory:

$ sudo mkdir /usr/share/sources
$ sudo chgrp oe /usr/share/sources
$ sudo chmod 0775 /usr/share/sources
$ sudo chmod ug+s /usr/share/sources

The first build

If Option A was choosen for environment setup (bash profile method), log out/log back in or close your terminal window and open a new one so that the environment changes enabled above will take effect.

If Option B was chosen (command line sourcing), issue the command to source the environment setup.

The build sytem is set up to build for verdex by default. If you want to build for basix or connex, you will need to edit ~/Gumstix/Gumstix-oe/build/conf/auto.conf. Comment out the line that selects verdex and uncomment the line that selects connex.

To build a basic root file system image that includes Linux 2.6.21, boa, cron, and ntp:

$ bitbake Gumstix-basic-image

If the build machine is missing any essential packages, bitbake will notify you about them at this point. Install the recommended packages and try the bitbake command again.

This initial build will take a bit of time since it will need to download all the source code. When it completes, the root file system image and the kernel image are in ~/Gumstix/Gumstix-oe/tmp/deploy/glibc/images/Gumstix-custom-verdex (or connex if that is the machine type you selected):

$ ls -1 ~/Gumstix/Gumstix-oe/tmp/deploy/glibc/images/Gumstix-custom-verdex/

Angstrom-Gumstix-basic-image-glibc-ipk-2007.9-test-20071101-Gumstix-custom-verdex.rootfs.jffs2

Gumstix-basic-image-Gumstix-custom-verdex.jffs2

modules-2.6.22-r1-Gumstix-custom-verdex.tgz

uImage-2.6.22-r1-Gumstix-custom-verdex.bin

There appear to be two jffs2 images above. However, in reality there is just one: Gumstix-basic-image-Gumstix-custom-verdex.jffs2 is a link to the actual image, Angstrom-Gumstix-basic-image-glibc-ipk-2007.9-test-20071101-Gumstix-customverdex.rootfs.jffs2

The first one is just a short name.

A.9 How to Load an Image OE in the Gumstix

Now that a new root file system is ready and a serial connection with Kermit is established, it is time to reprogram our Gumstix's Flash memory.

The following instructions assume that the steps in Section A.1.1 have been done.

Reboot the Gumstix, hit any key to stop the autoboot process and drop into U-Boot. At this point, verify that the U-Boot version is 1.2.0.

First, tell U-Boot to prepare to receive a file download at RAM location a2000000:

GUM> loadb a2000000

Now break the serial connection and return to the kermit prompt by pressing Ctrl-[CTRL-\] and then "c". Set up the file transfer by typing:

C-Kermit> cd ~/Gumstix/Gumstix-oe/tmp/deploy/glibc/images/Gumstix-custom-verdex/
C-Kermit> send Gumstix-basic-image-Gumstix-custom-verdex.jffs2

After typing in the send command, a screen appears displaying the progress of the file upload process:

C-Kermit 8.0.211, 10 Apr 2004, localhost

Current Directory: /home/username/Gumstix/Gumstix-oe/tmp/deploy/glibc/images/

Communication Device: /dev/ttyUSB0
Communication Speed: 115200
Parity: none
RTT/Timeout: 01 / 02
SENDING: => GUMSTIX-BASIC-IMAGE-GUMSTIX-CUSTOM-VERDEX.JFFS2
File Type: BINARY

File Size: 7491192
Percent Done: 2 /
...10...20...30...40...50...60...70...80...90..100
Estimated Time Left: 00:13:32
Transfer Rate, CPS: 8963
Window Slots: 1 of 1
Packet Type: D
Packet Count: 68
Packet Length: 4096
Error Count: 0
Last Error:
Last Message:

When this process completes, your new filesystem has been loaded into Gumstix RAM. The contents of Flash are still intact.

Now leave kermit's command mode and reinstate the serial connection to the Gumstix:

C-Kermit> connect

Getting the new files ystem copied into Flash memory is a two-step process. First, the old filesystem must be erased, then the new filesystem must be written from RAM to Flash.

To erase the old filesystem, enter:

GUM> protect on 1:0-1
GUM> erase all

The first line protects Flash sectors 0 and 1, which contain the uboot bootloader. *This line is very important!* If this line is omitted, Gumstix will be unusable and will have to be reflashed using a far more complicated process. The second line erases all of Flash except the two sectors that we just protected.

Now, commit the new files ystem to Flash:

GUM> cp.b a2000000 40000 ${filesize}

This copies your new files ystem into Flash (at address 40000). ${filesize} is a reference to the variable *filesize*, which was set by your file transfer.

Now, we use a similar process to load the kernel:

GUM> loadb a2000000

C-Kermit> send uImage-2.6.21-r1-Gumstix-custom-verdex.bin

C-Kermit> connect

GUM> katinstall 100000

GUM> katload 100000

GUM> bootm

This concludes all the basic information needed to make the Gumstix work properly.

REFERENCES

[1] ITU-T Recommendation X.210, "Information technology-Open Systems Interconnection-Basic Reference Model: Conventions for the definition of OSI services," common text with ISO/IEC10731, Nov. 1993.

[2] J. Corbet et al., *Linux Device Driver*, 3rd ed., Sebastopol, CA: O'Reilly Media Inc., 2005.

[3] G. Holland and N. Vaidya, "Analysis of TCP performance over mobile ad hoc networks," in *Proceedings of ACM MobiComm, Seattle, WA*, 1999, pp 219-230.

[4] M. Gerla, K. Tang and R. Bagrodia, "TCP performance in wireless multi-hop networks," in *Proceedings of IEEE WMCSA'99*, New Orleans, LA, 1999, pp 41-50.

[5] N. Gupta et al., "A performance analysis of the 802.11 wireless LAN medium access control," *Communications in Information and Systems*, vol. 3, no. 4 pp. 279-304, September 2004.

[6] V. Gite, "Linux shell scripting tutorial ver 1.0," 2000 [Online]. Available: http//www.freeos.com/guides/lsst/index.htm.

[7] R. Murray "Recent research in cooperative control of multivehicle systems," *ASME Journal of Dynamic Systems, Measurement, and Control.* vol. 129, pp. 571-583, Sept. 2007.

[8] L. E. Parker, "Designing control laws for cooperative agent teams," in *Proceedings of IEEE International Conference on Robotics and Automation*, 1993, pp. 582–587.

[9] Gumstix, "Initial setup for Gumstix operation," November 2008. [Online]. Available: http://www.Gumstix.com.