Bluetooth and Filesystem to Manage an Ubiquitous Mesh Network

Nicola Corriero - Emanuele Covino - Giovanni Pani
University of Bari
Departiment of Computer Science
Italy, Bari, Via Orabona 4, 70125
Email: {ncorriero,covino,pani}@di.uniba.it

Eustrat Zhupa
University of Vlora
Sheshi Pavaresia, Skele,
Vlora, Albania
Email: ezhupa@univlora.edu.al

***Abstract*—Hixosfs is a filesystem that lets you manage a network of Bluetooth devices. We have analyzed this filesystem in contexts of great movement and file sharing systems which requires high performance time. We study the system both for the management and to extract statistics.**

***Keywords*-Ubiquitous mesh; ad hoc networks; sensor networks; Bluetooth; Tracking.**

I. INTRODUCTION

Ubiquitous multimedia computing will change the way we operate and interact with the world with the development of numerous interesting ubiquitous multimedia applications.
Our purpose is to create a layer among operating systems, middleware and user devices to facilitate the modeling of the system and to make possible the efficient use of the embedded devices. The main idea is the use of bluetooth technology to create an ad-hoc indoor network. The system has been implemented using embedded systems linked to each other using bluetooth connection or an ad-hoc wireless network. In each of these, a hixosfs filesystem based database has been used to improve the performance.

For such reasons the choice is to try an innovative solution: hixosfs.
Hixosfs is a Linux filesystem that can be included in the standard Linux kernel. Differences respect to the widely used ext2 filesystem are additional features for storing and efficiently retrieving data.

We introduce scenarios, the critical points of these scenarios, other approaches and our solution. We explain the architecture of our system and all of its components. We explain how it's possible to build an alternative simple approach to this problem with a common filesystem. Finally some comparitions with other approaches.

II. SCENARIO AND CRITICAL POINTS

In this work we present an update of the scenario presented in [12]. Two scenario: bluetooth markentig, file sharing in a mesh network. Bluetooth marketing. In this context, there are mobile devices that stop or pass near a place of interest (library, cinema, stadium). Some device has bluetooth and is available to receive data from our system. Our device must periodically scan the environment looking for some device bluetooth and try to establish an exchange of data used to send content usually advertising media.
The second scenario is a laboratory or a library where people want to share information such as music files. When each user enters within range Action Bluetooth antenna automatically start the sharing of files in a profile stored on their devices (Openmoko).
In the context described above it's possible to identify some critical points of the system. In such points we see the differences with the other solutions offered by the market.
In changing contexts is necessary to have softwares which react in short time. The case of bluetooth devices is lampant. Each device stays in the antenna covered area for few seconds. During this short period the antenna should verify the presence of device services (so verifying the presence of Push) and establish a connection with it if it has not been contacted before. Checking if the device is already in the database of the contacted devices can be time consuming. Contexts on move suggests to use intelligent systems that can decide in a small time.
Client-server systems are not recommended to manage contexts where we need speed of reaction. Embedded system that can decide for themselves are advised.
Despite everything you need to use servers to handle large informations about the contexts.
Send sms bluetooth to someone on the move or turn on the light in an environment requiring considerable reaction. Embedded systems are designed for a permanent use and to carry out little well-defined tasks.

III. OTHER APPROACHES

Normally these situations are handled using PCs with large primary and secondary memories that make possible the use of every operating system and every mean for saving and managing information.
Other approaches of the system require the use of complex databases over servers and/or embedded databases over embedded machines.

In the embedded systems we have evident problems of memory that during the time have solicited light-weight and high-performance ad-hoc solutions. Sqlite [8] is an application that implements all the functionalities of a database

 ISBN: 978-1-61208-100-7

using simple text files. This enlighten the execution load of the system and facilitates the integration of the system inside an embedded system. However, the system installation produces a certain load to the mass memory.
Currently Linux fs as ext2 [7], ext3, reiserfs allows to manage with metainformation related to a file with *xattr* feature. Patching the kernel with xattr you have a way to extend inode attributes that doesn't physically modify the inode struct. This is possible since in *xattr* the attributes are stored as a couple attribute-value out of the inode as a variable length string. Generally the basic command used to deal with extended attributes in *Xattr* is `attr` that allows to specifies different options to set and get attribute values, to remove attributes to list all of them and then to read or writes these values to standard output. The programs we implemented in our testing scenario are based on this user space tool.

The last approaches we have compared is to avoid to use a particular filesystem to manage these scenario. We have compared our solution with an ext2 filesystem and some script in bash.

IV. Our Solution

An ad-hoc filesystem created inside the kernel will be used to handle the data provided by the various sensors of the network. The communication of the devices will be realized using an ad-hoc network implemented with aodv. Finally, the devices will have a linux distribution created ad-hoc with the necessary programs only. Python has been choosed as a programming language for the scripts to make the system independent from the final device. For this reason it has been necessary to install only a python compiler in the devices, saving a lot of space.

A high performance software infrastructure is needed in dynamic systems if we consider the time factor. Nowadays the embedded systems are spreading very rapidly. Little computers designed to spend limited energetical and economical resources, but with precise and well defined duties. On the other side, modern operating systems are designed to use all the resources. Our suggestion is to use the embedded devices like bluetooth antennas, placed by the entrances. The limited dimensions of the embedded devices make possible an adequate use of the space.

V. System architecture

A. Hixosfs

Hixosfs [10] is as an ext2 Linux filesystem (fs) extension able to classify and to manage metadata files collections in a fast and high performant way. This is allowed since high priority is given to the storing and retrieving of metadata respect tags because they are managed at fs level in kernel mode.
The *hixosfs* core idea is that information regarding the content of a metadata of a file belong to the fs structure itself.

Hixosfs has been used to tag multimedia files and bluetooth devices as well as user profiles. In this way all the load has been transfered to the kernel that handles and organizes the hixosfs files as occurrs. The servers and the clients contain partitions that can read and set the hixosfs tags so to manage the database.

In the case of bluetooth file, hixosfs extends the inode definition with a struct tag:

```
struct tag {
#ifdef CONFIG_HIXOSFS_BLUEZ
char macbyte1[3];
char macbyte2[3];
char macbyte3[3];
char macbyte4[3];
char macbyte5[3];
char macbyte6[3];
char devicename[40];
char scanningdate[9];
unsigned int tag_valid;
#endif
}
```

The struct tag has four fields for a total of about 100 byte of stored information, theoretically an inode can be extended until 4 kb then it's possible to customize it with many tags for your purpose. It's convenient to choose tags that are most of the time used in the file search to discriminate the files depending their content. We choose here what was able to maximize the time of search musical files by most commonly used criteria as album or author name and so on.

In our experiment we understood the limits of the originary idea of hixosfs that suggests the compiling of an ad-hoc filesystem for each type of tag. In our system was necessary to compile 3 different filesystems with 3 different types of tags (music, user profile, bluetooth). For such a reason we decided to use a generic version of hixosfs with a generic structure in which is possible to insert file representative tags case by case.

```
struct tag {
#ifdef CONFIG_HIXOSFS_TAG
char tag1[30];
char tag2[30];
char tag3[30];
char tag4[30];
unsigned int tag_valid;
#endif
}
```

In our partition there was only a hixosfs filesystem mounted in RAM with three folders containing files provided by three running processes: bluetooth devices detecting, musical files manager and user profiles.

An example of file handling.
`chbluez -m` sets the mac address of the device.
`chbluez -n` device name (when detected).
`chbluez -d` date of the first detection of the device in the system.

 ISBN: 978-1-61208-100-7

B. Bluetooth

There are two types of services used for file transfer:

- Object File Transfer
- Object Push

Most of the bluetooth devices implement ”Object File Transfer” service, in which for file transfer is necessary an authentication using a unique PIN number on both client and server.

Around 40% of the devices implement the ”Object Push” service in which is possible to send files without using any PIN number. However, to receive a file is necessary to accept a connection request. This technique facilitates the interaction cause you don't need any PIN number, even if in the majority of the devices there is a hardware lock in case of repeated connection requests via the Push protocol.

C. AODV

We are choosing to use the routing AODV (Ad hoc On demand Distance Vector) protocol, because it's suited for route discovery and routes maintenance within ad hoc network.

To this purpose a Linux kernel and a mini distribution have been compiled and analyzed in the case of an handhelds HTC blue-angel and for Openmoko. Then it has been extended including the Manet support, specially compiling the aodv-uu, the AODV protocol implementation realized by the Uppsala University[4]. At this point we are ready to generate a mesh network. Etherogeneous nodes like handhelds, work station, notebook, mobile phone, router, having wifi connectivity with Linux operating system and aodv-uu module installed, can contribute to the creation of a decentralized newtork, working in a mobile context.

In our system we have cross-compiled code that implements the AODV protocol for ARM architecture [1]. The implementation used provides a daemon that constantly sends “ hello”message searching for other nodes. This way when the user comes within range of another device to authenticate.

D. Openmoko

To test hixosfs filesystem for music files we have choose Openmoko Freerunner[5], an open source project with the aim to port Linux on mobile device. Inside Freerunner Linux os already works, we had just to patch the kernel with our modifications and we choose to create hixosfs partition inside a minisd with 2 gb of stored music file.

VI. Evaluation of system's performance

The testing scenario was implemented in a shop and in a music laboratory. Every user was provided a Openmoko[5], a bluetooth mobile, gps and wireless with Linux.

Inside the buildings there were embedded systems with bluetooth antennas for identifying the users and send bluetooth sms.

A. Cinema

The aim of the experiment was sending advertisement sms via bluetooth. The text of the sms was of multimedial type and was addressed to promote objects of the shop itself. Each user was invited to switch on the bluetooth device of his mobile.

The antennas were in communication with the server for sharing of the contents to send and at the same time to check the mac addresses in order to avoid sending the same content for more than once to a given mac address.

The system was handled in remote with a little web based application for managing the synchronization of the bluetooth advertisement campaign with respect to the contents and the hours. Every antenna was sending daily reports to the server and all the data were handled in a hixosfs filesystem. The reports were compressed in files of a hixosfs partition in which for every campaign were registered: message sending, message not accepted, disinterest of the user for the system.

The data of the experiment show that the system must be promoted better. Only 10% of the users were aware of the connection request and only 80% of those accepted the message. The main problem (90%) that came out with a observation of the log files is the fact that people didn't knew the system so not knowing what kind of messages they were receiving, they choose to ignore the request or just not accept.

B. Laboratory

The motivation of the experiment was to share some multimedia files inside the laboratory based on the profile of each user.

With every user there is a Rfid tag associated, an Openmoko mobile with a bluetooth device. In the server there have been loaded the musical preferences of each user. The entrance of the lab is managed by an ”intelligent” system that performs an authentication via a Rfid. Inside the lab there is an ad-hoc wireless connection implemented with aodv protocol. When the user is detected by the system, he is profiled and some folders are shared containing musical tracks coherent with the user profile. At the same time every user shares musical tracks with the system. All the musical tracks of the users are not on the server but in the device of each user. Tracks are saved in hixosfs partition of the flash memory where each file has been tagged. The music files were tagged for the multimedia contents and for the profile. Each file shared by the user A with profile P_A has been tagged with the elements of the profile P_A.

The user B, with profile P_B that has at least one element same as in the profile P_A, has the file sharing from the user A. With such approach it was possible the to implement the probably interesting file sharing between users with similar profiles and avoiding probably not interesting file sharing between users with different profiles.

 ISBN: 978-1-61208-100-7

VII. SCRIPTING

The final Linux distribution will contain only the python interpreter and the essential commands without adding programs for database management. The system therefore includes a Linux kernel (2.6.23) patched with hixosfs and a shell in Python through which you can either start the daemons either use hixosfs. Pybox is a shell created to enlighten the work load on the embedded systems. Pybox has been written completely in Python and it occupies less space in the device having interesting performances. The Pybox project implements the rewriting in Python of the basic commands of the system. To use Pybox in our embedded systems we rewrote the commands interfaced with hixosfs.

The system is thus platform independent. Using Python has facilitated the porting of the system on ARM architecture (Openmoko). System startup will mount a partition of the flash memory is partitioned with hixosfs and through openvpn also a remote folder (mounted via NFS) for sharing the system log. Periodically, in fact, the system performs a backup on the remote server.

A. Directory tree

The user process interface has been extended with the two programs *stattag* and *chtag* to access or modify the new inode information for one file. Ad hoc user mode tools have been implemented to extract metadata and populate the whole fs in one step. One example is the command *addbluez* that extract mac address of bluetooth device to fill the inode.

To create a directory tree with tagged files, there is the command *orderby*, with syntax:

```
$ orderby [-mac | -names | -data]
```

The command *scan -bluez* can find inside fs files with a specific content of tags and has the syntax.

```
$ scan -bluez -option [ value | ALL ]
Options are:
-datestart DATE -dateend DATE
-date DATE
-mac MAC
-name NAME
```

For example: where *00, 19, 2D, 14, C9, 93, ...* are folders.

In the foils of the tree (representing each device) have been created files to save the events:

- `to send` - to lock the device for other sending;
- `sent` - to identify the interaction (with positive or negative result) with the device;
- `ping` - to save all the times in which the device is in the scope of the bluetooth antennas;
- `problem` - to save all the problems of each mac address.

```
$ scan -bluez -date all -mac AA:BB:CC:DD:EE:FF
```

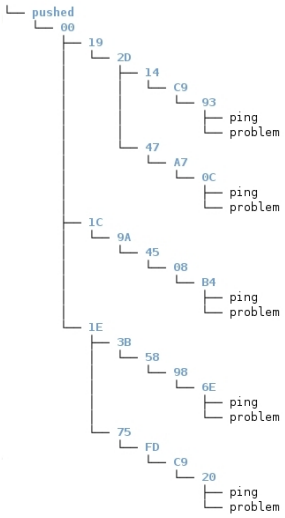

Figure 1. *orderby -mac*

is the command to find all activity of mac AA:BB:CC:DD:EE:FF.

B. Bluetooth

Lightblue [6] is a layer in python for handling bluetooth devices. This layer offers high level functionalities .

For each functionality there is an appropriate error handling. For example in case of denied connection the system returns an error code.

Script for sending a bluetooth sms.
Input mac address, Push channel, path of the file in th system, filename to display in the ending device.

```
import lightblue
def sendsms ( mac , channel , pathfile , nomeoutput ):
        client = lightblue.obex.OBEXClient(mac, channel)
        client.connect()
        putresponse = client.put({"name": nomeoutput},
        file(pathfile, 'rb'))
        client.disconnect()
        return;
```

Script for handling errors caused during bluetooth sms sending. In case of error a file is created with all the information regarding mac address, error and date of the error.

VIII. EXT2 SOLUTION

We have created the same result as hixosfs folder tree inside a simple ext2 fs. We have writen some simple bash scripting to create our data log with a tree of folders and we have created simple bash scripting to select data from this tree.

A. Bash scripting

The particular organization of data in folders allow easy reference data using sample scripts in bash. The following

 ISBN: 978-1-61208-100-7

script allows us to scan the " database" the number of unique devices discovered on a certain day every hour.

The system has a policy to store the log output of this script every day to provide simple statistics for the owner of the shop.

devicexhour.sh

```
echo -e "hour\t device"; data=$1
for hour in {00..24};
do
  echo -en "$hour\t "
  count=$(grep $data-$hour pushed/*/*/*/*/*/*/ping |
      cut -c 1-29 | uniq | wc -l);
  echo $count
done
```

The following script allows us to count how many minutes each device is near of our antenna. With this information you can determine the periods of day when most people stop near the antenna and then inside the shop.

With the next script you can count the number of unique devices to date input. With this script you can establish a trend throughout the year the inflows of people inside the shop.

devicexday.sh

```
count=0; data=$1;
for i in `ls pushed/*/*/*/*/*/*/ping`;
 do
    grep $data  $i> /dev/null;
    if [ $? -eq 0 ]; then let count=count+1;
    fi
done
echo $data = $count
```

This is the most important script inside our antenna-device. This script runs and creates a tree of folders as in Figure 1 like hixosfs without hixosfs.

pushed_no_message.sh

```
while [ true ]
do
contatore=1
numrighe=0
sdptool search OPUSH > ./opush
fallito=`cat opush | tail -1 |
    grep 'Inquiry failed' | wc -l`
if [ $fallito -ne 1 ]; then
{
   cat ./opush >> log/logopush
   date >> log/logopush
   cat opush | grep -B 1 Service | grep Searchin |
     awk '{print $5}' | sort -u > pushmac
  for i in $(cat pushmac); do
    echo -n $i; cat opush |
    grep -A 9 $i | grep Channel; done
    | grep :..: > macechannel
   numrighe=`wc -l macechannel | cut -c 1-2`
}
...
if [ $numrighe -ne 0 ]; then
{
sed -i s/Channel:/\\n/ macechannel
..
for i in $(seq $numrighe)
do
...
mac=${array[$j]}
channel=${array[$i]}
temp_dir=$(echo $mac | sed -e 's/:/\//g')
...
  mkdir -p pushed/$temp_dir
  touch pushed/$temp_dir/ping
  DATA=`date +%d-%m-%y-%H-%M`
guardadata=`grep $DATA pushed/$temp_dir/ping |
  wc -l`
if [ $guardadata -eq 0 ]; then
{
echo $DATA >> pushed/$temp_dir/ping
...
```

IX. PERFORMANCE MEASURES

The presented testing scenario is similar to the system proposed in [12]. The differences are the extra time to test the product and more detailed statistics.

In this section we present performance measurements made on *hixosfs*. The monitored operation is reading tags from musical file o bluetooth file.

Here we show that the disk performs better for indexing and retrieving music over standard file systems but we didn't study so far the loss of performance respect other uses of the fs. The idea in fact is that hixosfs will be used only for a disk partition containing a musical data or bluetooth device file collection to better organize and query such data and not for the whole disk.

We measured in fact the time required to perform this operations by *hixosfs* and we compared it with the time needed in the case the data are stored in a common fs and accessed by a Sqlite db.

Finally we have compared hixosfs solution with a simple ext2 filesystem managed by simple bash scripting.

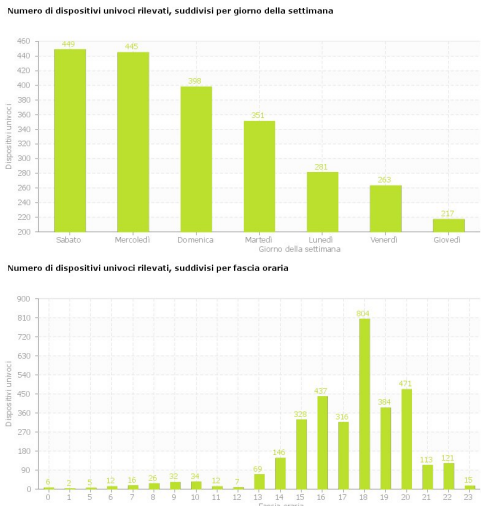

Figure 2. *Unique devices per day and hour*

A. Sqlite vs hixosfs vs ext2

After three months of testing in a cinema, our filesystem contained about 20,000 unique mac address scanned by the system. We have recreated the conditions both for the sqlite database and in an ext2 filesystem to compare systems.

First we compared the search for single mac address in the db.

 ISBN: 978-1-61208-100-7

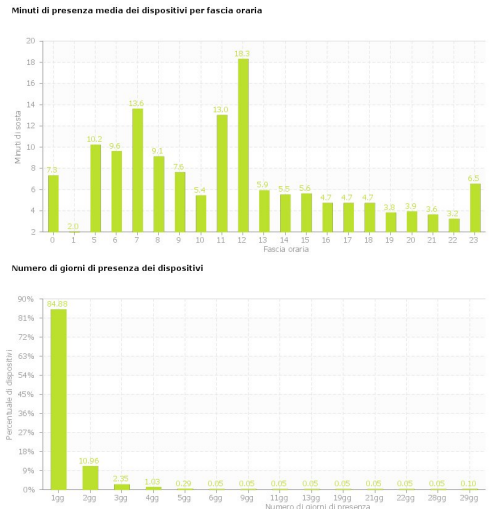

Figure 3. *Average waiting time near antenna*

A simple table `mac` have been created with:

- *id smallint*
- *mac varchar(20)*

We have insert into table 20.000 mac address generated by a casual ad-hoc algorithm. We have created 20.000 files inside a hixosfs partition and we have ordinated (by `orderby` command) a tree of directory to store all informations about every device. We have compared the time to execute "scanbluez" that works with hixosfs and "bashbluez" a simple bash scripting that works with ext2 filesystem.

We have compared time to execute a simple search of one mac address (case A in Table I) .

During testing we had the need to see statistics on the state of the system eg

- the number of devices that have agreed to receive SMS advertising
- the device number that comes back regularly and how often
- problems related to non-submission of the SMS advertisement

Our system provided the files inside the folder which contains for each device problems, and pings sent. We created the database sqlite a second table "sent"

- *id smallint*
- *data varchar(20)*
- *problem varchar(20)*
- *flag smallint;*

We have populated this table with 150,000 records of problems and on random times and we have compared the statistics to extrapolate from the system (Figure 2). (case B in Table I) .

Case\System	sqlite	scanbluez	bashbluez
A	0m0.093s	0m0.004s	0m.010s
B	0m.183s	0m,012s	0m.098s

Table I

X. Conclusion and Future Work

We presented hixosfs as a new solution to manage bluetooth marketing context. We tested this idea in a couple of real context. We improved the tests in [12] applying to cases where there are dependencies between data, when access to data in databases is perhaps the most performant.The system presented use also a platform-indipented code in Python to manage bluetooth connectivity and hixosfs user space program. We compared the idea with the standard approach and we noticed how the approach is more performant. In the future we need to improve the file system, adapting it to a file system lighter than ext2.

References

[1] Corriero, Cozza, Pistillo, and Zhupa. *Wifi Mesh for HandHelds in Linux*, 978-989-8111-60-9, pag 380-383. ICWN 2008.

[2] Corriero and Cozza. *The hixosfs music approach vs common musical file management solutions*, ISBN: 978-989-674-007-8, pag. 189-193. SIGMAP 2009.

[3] Rubini. *The "virtual filesystem" in Linux* . Kernel Korner. http://www.linux.it/∼rubini/docs/vfs/vfs.html.

[4] AODV-uu homepage - Uppsala University, http://www.it.uu.se/research/group/coregroup, 12.2008

[5] Openmoko. http://www.openmoko.org, 03.2010. Home Page.

[6] Bea Lam. LightBlue: a cross-platform Python Bluetooth API. 03.2010. http://lightblue.sourceforge.net/

[7] Card, Ts'o, and Tweedie. *Design and Implementation of the Second Extended Filesystem*, http://e2fsprogs.sourceforge.net/ext2intro.html.

[8] Sqlite. http://www.sqlite.org/. Home Page. 03.2010

[9] Kernel. Torvalds. www.kernel.org. Home Page. 03.2010

[10] Hixosfs. Hixos. www.di.uniba.it/ hixos/hixosfs. Home Page. 03.2009

[11] Corriero and Zhupa, *An embedded filesystem for mobile and ubiquitous multimedia*, MMEDIA2010.

[12] Corriero and Zhupa, *Hixosfs for ubiquitous commerce through bluetooth*, FutureTech2010.

 ISBN: 978-1-61208-100-7