Low-cost Distributed Video Surveillance with Discarded Mobile Phones

Alexandru E. Şuşu University Politehnica of Bucharest, ETTI, Bucharest, Romania

***Abstract*—**

Wireless video surveillance is an important option suitable when it is difficult to have cable Internet and electricity infrastructure, which is the case, for example, for construction yards or vehicles.

We present iCam, an open-source Internet-enabled video surveillance project, in which we use mainly discarded camera phones. The mobile application is written mainly in Python on most phone platforms, namely Android, Symbian, Windows Mobile and iOS, and also on Raspberry Pi, allowing us to share code between the platforms and cutting down our development costs.

We used iCam for almost seven years on several smartphones in order to provide video information with high availability from remote sites at low-cost rates, thanks also to a simple form of Bluetooth tethering we employ.

We present considerations on the robustness of the phones for long-term usage and also on their memory, CPU and energy consumption.

To the best of our knowledge, we are the first to have approached such long-term smartphone usage for surveillance and mobile cross-platform development in the Python language.

***Keywords*-wireless media sensor nodes; video surveillance; camera phones; cross-platform mobile development, Python; fault tolerance; power management, energy consumption profiling.**

I. Introduction

With the proliferation of smartphone devices and development of cellular communication, real-time mobile video streaming has become a mainstream option.

In this paper, we present the *iCam (inexpensive Cameras)* project, an open-source, Internet-enabled video surveillance solution. iCam uses a web service, either our dedicated open-source server application or Google's *YouTube* video platform, which receives video content from devices like mobile phones equipped with one or two cameras, normally no longer operated by the owner, or *single board computers (SBCs)* such as Raspberry Pi [1]. The phones connect to the server via a mobile Internet connection, such as *4G, 3G, EDGE, GPRS*, or via *WiFi* while the SBCs normally connect via an *Ethernet* cable. We offer a couple of remote viewers for the recorded media. Our solution implements also a simple form of detection of *out of the ordinary* events, such as motion in video or big noise, being able to alarm the user by calling him on his private number, for example.

The project is especially useful for places where the user does not have cable Internet infrastructure but has mobile network coverage, such as rural areas or vehicles. If we use several devices at the same location, in order to avoid paying a mobile data connection for each, we can form a simple Bluetooth network and delegate only one device as a gateway to send the media information to the web server. We can also use a WiFi router to create a network.

We choose to implement a client-server architecture. This implies that the server, which normally does not share the deployment site infrastructure, offers normally better connectivity and availability for the remote viewers and the client devices.

The iCam clients are written mostly in Python, making the respective application easy to port on any platform supporting Python. We made iCam available on smartphones running Symbian, Android, Windows Mobile and iOS, and also on Raspberry Pi's Raspbian OS, and experimented with most of them in the last seven years. We note that even if Symbian phones are no longer manufactured, they can still be bought from the second-hand market, being very robust and well-designed.

II. Related Work

Similar research projects use mobile phones as media sensor nodes and build networks with them. We present a few of them:

- the *Facet* project [2] consists of a setup somewhat similar to ours. They interconnect *Nokia 6630* camera phones, running a dedicated *Java 2 MicroEdition (J2ME)* application, through Bluetooth and GPRS, creating small and long range networks, respectively. The nodes communicate in order to perform object tracking. However, they use a peer-to-peer architecture, while iCam uses basically a client-server architecture.
- the project described in [3], [4] proposes to create a sensor network out of the owners' mobile phones to assess various characteristics of the environment where the users are located, such as noise levels or weather conditions. Such a network is able to serve information to interested people who are not able to get it otherwise, eventually for a certain small cost. The architecture of their system is quite elaborate for the data collection task and takes into consideration the mobility and the variable availability in the network of the devices since the phones are normally used for activities like standard calling by their owners.

Individual mobile phones are used in [5], running *Python for Symbian S60 (PyS60)*, Nokia Research Center's port of Python to the Symbian S60 platform, and *J2ME* applications. They compare the quality of the audio recording of a series of sample signals for both applications on a few *Nokia N95* phones and also on a dedicated phonometer. The phones do not communicate among themselves: the audio data is processed locally and the recording performance is read from each device manually.

Other software projects use smartphones for video surveillance: for Android, we have *MobileWebCam*, which is also open-source, IP Webcam, Mobile Hidden Camera, Spy Camera Advanced. Other applications for Symbian are SensyScan, Bluetooth SpyCam, VRCC (open-source), Bambuser. iCam differentiates itself from these alternatives through the following: i) it is an open-source cross-platform solution working for most smartphone platforms and Raspberry Pi; ii) it performs video recording; iii) it can do *Bluetooth tethering*, which allows Internet traffic sharing from one device to other iCam nodes for more cost-effective networking; iv) it has versatile remote viewers and it uses more complex computer vision analysis.

An established solution for remote viewing IP cameras is *http://sensr.net*, a media platform storing and presenting content from smart cameras.

We mention also the very popular *Skype* and *Google Hangouts* services, which handle as well video streaming over smartphones, their focus being different than ours: they stream in real-time video from the smartphone to one or more devices, the accent being put

on Quality of Service during calls, which normally take in the order of minutes.

There is also the well-established market of professional video surveillance, which encompasses solutions such as analog or digital cameras with very good optical sensors and infrared lamps, with robust cases, able to perform automatically *Pan-Tilt-Zoom (PTZ)*, which can change the orientation of the camera and zoom in and out optically the view. These cameras can transmit the information in real-time to a *Network Digital Video Recorder (NVR)*. Then, a central monitoring desktop, web browser or mobile application, which can be run from any location, can connect to the NVR to present the existing videos to the user.

Note that wireless video surveillance is less popular than solutions with cable, due to bandwidth and cost limitations, but is very useful where deploying wires can be cost-prohibitive: on construction yards [6], [7], at the countryside or where mobility is required such as in buses or cars.

A thorough discussion of professional video surveillance is beyond the topic of this paper. We invite the interested reader to visit various dedicated websites, most notably *IP Video Market Info* producing catalogs such as [8], presenting the latest trends in video surveillance.

Compared to professional video surveillance solutions, iCam runs on discarded phones with rather low processing power or other embedded devices such as Raspberry Pi, all having photo cameras with a lower quality and with a casing that is not very robust to withstand, for example, high humidity and strong winds. These are two important reasons why we do not consider ourselves an actor in the field of professional video surveillance. However, in Section V we argue that these devices we employ are robust and provide good recording capabilities.

III. System Architecture

As depicted in Figure 1, the iCam system is comprised of the following components:

- the **device client** running on the smartphones we no longer use for standard voice communication or on other embedded devices such as SBCs. This is a cross-platform application that runs on Symbian S60 and partly on Symbian UIQ, Android, Windows CE, iPhone and Raspberry Pi. We also have a less maintained version written in J2ME for feature phones. The iCam phone client can record video, audio and take photos. This media is uploaded on *YouTube*, or on the dedicated iCam server, from where the user can view it in real time or later.
  Note that a device running iCam can be used also as a gateway relaying to the Internet the media generated by the Bluetooth clients "subscribed" to it, as we discuss in Section IV-A.
- the **server**, whose main functionality is to receive the media content from the iCam clients, store it and later send it to the *iCam remote viewers*. We recommend using Google's *YouTube* platform as a server, which provides free, simple to use and secure storage. We also provide an open-source *dedicated iCam server*, written in PHP, which allows full control on the implementation of the protocols used.
- the **remote viewers**, which allow the user to view the media uploaded from the iCam device client:
  - the *iCam WebViewer*, a generic web browser viewer, which can display the most recent video content from any of the mobile devices registered for *YouTube* or for the dedicated iCam server;
  - the *iCamViewer*, a more complex and easily extensible *Central Monitoring System (CMS)* desktop application, which displays multiple views of the media uploaded to the server and performs video analysis on it.

The source code of these components and the mobile app binaries can be found at *http://github.com/alexsusu/icam-mobile-revival*.

Using the terminology from [8], the iCam phone platform has the following characteristics: i) the camera device (e.g., the phone): fixed (no panning or tilting), color, normally lower than *Standard Definition*

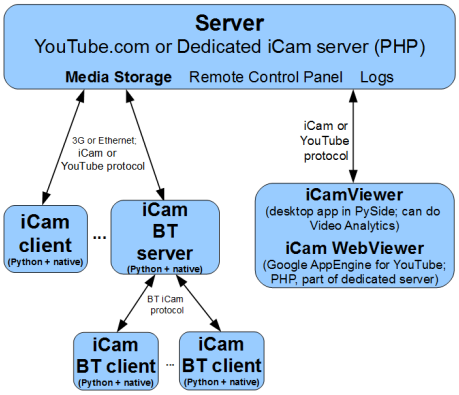

Fig. 1: System architecture - components with implementation details and protocols employed.

(SD), with IP (Internet Protocol); ii) wireless connectivity, employing one or more of the 3G/GPRS, WiFi and Bluetooth technologies; iii) as Video Management System we have either the dedicated iCam server (or *YouTube*) and Viewer, which are software solutions; iv) storage for the media can be internal, on the memory card of the phone, and, normally, through "networked" storage, namely *YouTube*, or the dedicated *iCam* server; v) the video recording is performed normally as *segmented videos* (but possibly also *based on motion*) at low resolutions such as 176x160, 15 fps, 320x240 or 640x480, at 30fps, since the discarded, older smartphones we use have lower recording capabilities; vi) video analytics: on the desktop viewer and, also possible to run as a subtask on the smart camera device - see Sections III-C and III-A; vii) video viewers: for Remote PC Viewing we have *iCamViewer*, for Mobile Viewing we have the *iCam WebViewer* and even the *YouTube* standard frontend.

A. The Device Client

The client can run on smartphones or Raspberry Pi SBCs. The iCam phone application transforms the mobile phone in an IP surveillance camera. Such a device can be used either for indoors, outdoors spaces or even vehicle video monitoring. On the other hand, we use Raspberry Pi mostly as an iCam Bluetooth server.

The iCam client is responsible for forwarding the media content from any of the cameras and microphones installed on the device. Also, if running on a decently powerful CPU, the client can also process itself the captured data, for example, in order to send only relevant videos to the server.

The functionality of the clients is similar for the various OSes, differences being noted at the level of UI design or where the runtime platform does not offer required features - e.g., older Android SDKs allow access only to the main camera, and *iOS SDKs* versions 1 and 2 do not allow video recording at all. iCam running on Raspberry Pi offers no UI, so we have to manually modify its configuration files.

The application normally uploads data to the server. For this, you need to enable *YouTube* on your Google account. Also, the client can work offline and store the captured media on the flash memory card.

The application is initially configured to record 30 seconds long videos with the main camera of the device, normally of resolution 176x160x15fps on Symbian or 640x480x20fps on Android, and then pause for two minutes ($pauseInterval = 120$). This means we offer a *segmented* streaming video solution since the videos are sent as individual files of a certain length, e.g., 30 seconds, an application parameter we call *videoRecordDuration*. This is well suitable for the case when one device acts as a Bluetooth server for a few clients, sharing a low-bandwidth Internet connection, such as 3G, in rural areas.

When a video is being recorded, the viewfinder is turned on during this period. In the case of the Symbian S60 iCam client, the application displays a timer with how many minutes and seconds

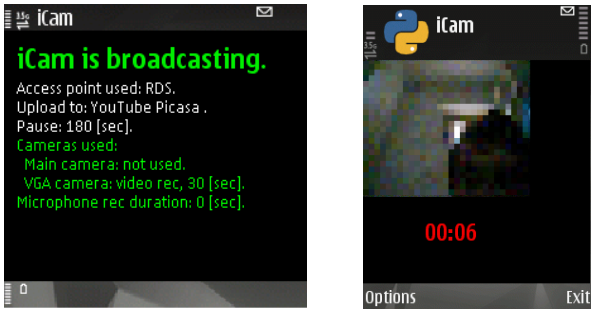

Fig. 2: Snapshots of the iCam Symbian S60 application. At the left, the main window of the iCam application on an S60 phone. At the right, iCam is performing video recording.

Fig. 3: A *Nokia 6120* phone running the iCam application, with a mirror mounted to enhance the FOV of the VGA camera.

we have recorded so far. The moment the recording is finished, the video is uploaded to *YouTube*, which takes in the order of a few seconds; the video can be stored in the iCam *Media* subfolder as well if specified. You can see in Figure 2 snapshots of the iCam Symbian S60 application.

Besides video recording, we can also take photos every $pauseInterval$ seconds, store and send them to the iCam server. Another operation mode is the *Photo Burst* mode, in which the viewfinder is always on and iCam receives frames continuously, with a rate depending on the image size and on the performance of the system.

We can perform with iCam a primitive form of local *image analysis* during the *Photo Burst* mode: we can enable a simple motion detection algorithm on the phone, which searches for meaningful difference among consecutive frames and uploads only the respective image(s) to the server. Note that we can perform complex *video analysis* on the *iCam* remote viewer also, as we discuss in Section III-C. We motivate the usefulness of such a two-tiered *video analytics* solution in the following way: on the resource-constrained phones, such as the ones manufactured before 2009, we allow running some *very simple* motion detection algorithm in order to filter out uninteresting frames or videos and forward only what is meaningful. Such an approach can prove beneficial for the network bandwidth and, also, for the energy efficiency of the device. In-situ image interpretation is reported also in other works, for example, the Cyclops very low-power smart camera nodes [9].

In Figure 3 we can see a *Nokia 6120* smartphone manufactured in 2007, with the charger connected to it, which ran continuously the iCam application for almost seven years. Note that none of the devices have a robust case to protect them against special environmental conditions, so we install them under roofs, which offers shelter against the rain, snow and wind. Since the cameras occupy normally fixed positions in the phones, we can perform with some difficulty panning and tilting together with the phone, especially because there are two cameras on the device. Therefore, we can use simple mirrors to enhance the *Field of View (FOV)* of the cameras.

Besides the video and audio media information, iCam can report data readings from the GPS, accelerometers, battery charge status, charger connect status, GSM line status, phone timer and so on. The non-media information is currently reported only to the dedicated iCam server as text logs, as we discuss in the following subsection.

The source code and the binaries of the iCam clients for Symbian, Android, Windows Mobile and iOS can be downloaded from *http://github.com/alexsusu/icam-mobile-revival*. Also, the Android iCam client can be downloaded from Google Play, its package name being *com.MobileRevival.iCam*.

B. The Server

The main responsibilities of the server are to receive, store and send the media content from the iCam clients and relay commands from the users to the clients.

The iCam client is normally configured to upload media to Google's *YouTube* server by using the *YouTube Data* [10] and *Google APIs* or, until October 2016, the deprecated *GData API* [11]. The other alternative is to use the *dedicated iCam server application*, written in PHP, which is open-source and can be installed on a *LAMP (Linux, Apache, MySQL, PHP)* server in order to be used as a personal video storage and online viewer solution. The server can receive any type of files, of any length, basically at any rate. It can store metadata information, such as the other sensor readings of the phones. Being open-source, we can experiment with various transmission protocols and we can fine tune all the components to our needs.

To change the settings of the iCam phone application you can use its UI or you can send commands to iCam via the web, a useful alternative especially if you do not have physical access to the phone. The dedicated server allows the user to send commands remotely to the iCam clients by visiting the *Remote Control Panel* web page. The client downloads a newly submitted configuration basically every $pauseInterval + videoRecordDuration$ seconds. We designed iCam to check settings at this rate to avoid generating more Internet traffic, which results mainly in a smaller energy consumption.

The dedicated iCam server supports the software update of the iCam clients by storing the latest *lib.zip* file containing the new Python application code. The *iCam* client checks for a new version of the application on the server only when it starts.

C. The Remote Viewers

A very interesting solution to view the media stored on the server is *iCamViewer*, a complex and highly extensible application using the *PySide* LGPL-licensed Python library, with bindings to the Qt library [12]. It displays simultaneously in a split screen format the videos from the various devices selected. We depict in Figure 4 the *iCamViewer* application running. Note that the bottom right view is from an analog *InfraRed (IR)* camera; this camera is connected via coaxial cable to a small LCD, positioned in the FOV of one of the cameras of a *Nokia N82* phone running iCam, therefore accomplishing a simple analog-to-digital image conversion.

iCamViewer can perform various simple real-time transformations of the rendered videos, such as rotate, flip or color corrections. Also, as already discussed, *iCamViewer* performs image and audio analysis employing *OpenCV* [13] and *SoX* [14], respectively, in order to detect interesting events - note that these tasks can also be delegated to the dedicated server. Once the analysis is finished, it tags the videos based on the interesting events detected. If it detects an *out of the ordinary* event such as a high-intensity sound or a lot of motion, an alarm can be triggered: for example, a warning window is displayed on the PC or a specified phone number can be called. Such events can be the alarm audio signal of a gas leak sensor if done at the right intensity, or the *Passive InfraRed (PIR)* sensor actuating a light bulb positioned in the FOV of the camera phone in case of an intrusion.

Another alternative is the *iCam WebViewer*, which is available as a part of the dedicated iCam server or as a Google AppEngine application, which plays the most recent videos uploaded in the *YouTube*

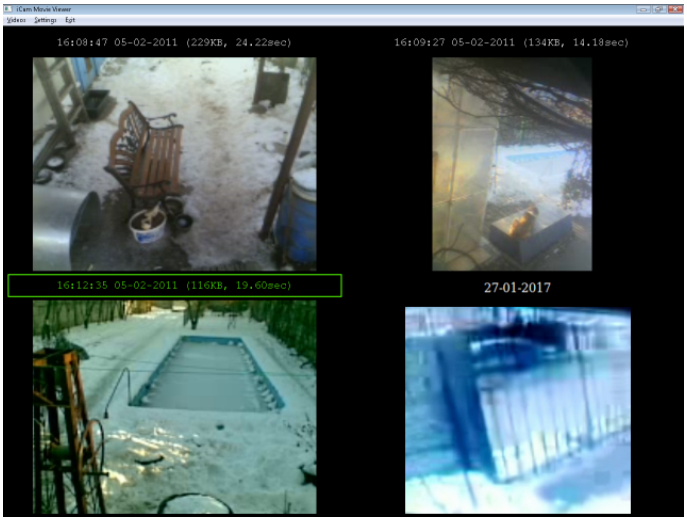

Fig. 4: The *iCamViewer* desktop application, presenting videos from three mobile phones installed at a rural location close to Bucharest.

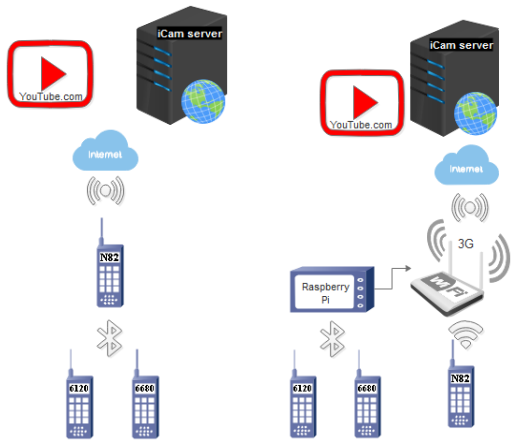

Fig. 5: Network diagrams of two deployments. At the left, the diagram for a setup used until 2014 in a rural area, with a *Nokia N82* acting as a gateway. At the right, the diagram for our current configuration, with a *Raspberry Pi B+* and a WiFi 3G router used as gateways, at the same location.

user account for the specified phones and cameras. It is working on popular browsers, such as Mozilla Firefox, Google Chrome and Opera. This viewer is also suitable for mobile platforms, which cannot currently run the more complex *iCamViewer* application.

IV. More Details

One of the main ideas of our project is to "recycle" older camera phones by transforming them into intelligent Internet surveillance nodes.

Today's smartphones are discarded roughly every two years, so it is not uncommon for people to have an older, yet functional smartphone. In fact, the big volume of mobile phones discarded worldwide becomes a serious environmental concern [15], especially because of the rechargeable batteries. From 1980, nearly 800 million cell phones were disposed [16]. Also, statistics [17] in the US show that in 2007 and similarly in 2009, 140 million mobile phones were disposed of, out of which 10% were recycled, the rest being trashed.

A. Mobile Internet Connection Sharing

Mobile Internet comes at a cost - for example, a medium quality 3G connection of virtually unlimited traffic, starts from 4 Euros per month in Romania. Such a connection reaches a throughput in the order of Mbps until a certain data volume of 5 GB, for example, is trafficked and then the bandwidth is limited down to 128 Kbps.

Therefore, as already discussed, we have an incentive to share one 3G connection on as many devices as possible. This has been also reported, for example, in [18], where one device is used as a gateway to "consolidate" the Internet traffic, while the others connect to it via WiFi or Bluetooth, which leads also to better energy consumption and response times.

In our deployments presented in Figure 5, we delegate one device to act as a gateway to the 3G network and employ normally Bluetooth *class 2* for local networking because all our phones are equipped with this radio technology; if possible, we employ WiFi *802.11 b/g* (or *n*), which is less common on the older phones we use and about four times more power-hungry, albeit about ten times faster [19]. By doing so, only the gateway has to pay for the connection. The delegated Bluetooth server device relays to the Internet the data received from the Bluetooth client phones.

The Bluetooth communication is implemented in a simple manner by transmitting *OBEX (OBject EXchange)* messages. The phones implement this protocol by using the Bluetooth only as long as data is transferred and automatically disconnecting immediately afterward, which results also in good power management. The OBEX profile is a widespread Bluetooth standardized service, which uses underneath the Bluetooth *RFCOMM* transport protocol, designed to exchange data as files. Because of this, one might see in the messaging folder of the Bluetooth server (or client) device the OBEX messages received via Bluetooth that are not processed yet by iCam.

Sending data via Bluetooth to the gateway in order to be further sent to the Internet is a particular implementation of *Bluetooth tethering*. Although less employed, we can also send data from the web server to the Bluetooth clients - we usually do this when sending iCam commands to the clients and, obviously, use the gateway to relay the command to its destination.

To deploy such a standard star topology Bluetooth network, we need to install iCam on all the devices and configure the application to act as a Bluetooth server or client, accordingly.

Note that for the phones we deployed, the range of communication for Bluetooth class 2 transceivers is at most 10 meters, influenced by factors such as walls in-between and other interferences.

TABLE I: Devices used with iCam. By (*) we mean the iCam support is poor.

Device Name	OS	CPU [MHz]	RAM [MB]	Radio
Symbian phones				
Nokia 6680	v8.0a S60 2.6	220	20	BT 1.2, 2G
Nokia 6120	v9.2 S60 3.1	369	50	BT 2.0, 3G
Nokia N95	v9.2 S60 3.1	332	56	BT 2.0, 3G, 802.11 bg
Nokia N82	v9.2 S60 3.1	332	120	BT 2.0, 3G, 802.11 bg
Nokia E7	Symbianˆ3	680	245	BT 3.0, 3G, 802.11 bgn
Samsung G810 (*)	v9.2 S60 3.0	330	69	BT 2.0, 3G, 802.11 bg
Motorola A920 (*)	v7.0 UIQ 2.0	168	32	BT, 3G
Raspberry Pi B+	Raspbian	700	512	eth, (USB BT 2.0)
Raspberry Pi 2 B	Raspbian	4x1000	1024	eth
Banana Pi	Bananian	2x1000	1024	eth, BT
Samsung GT i5500	Android 2.1	600	170	BT 2.1, 3G, 802.11 bgn
Samsung Galaxy S	Android 2.1	1000	512	BT 3.0, 3G, 802.11 bgn
UTOK 450D	Android 4.2.2	2x1200	512	BT 3.0, 3G, 802.11 bgn
HTC Touch Cruise	Windows Mobile 6.0	400	64	BT 2.0, 3G, 802.11 bg
iPhone	iOS	412	128	BT 2.0, 2G, 802.11 bg

B. Examples of Deployments

We experimented mostly with iCam on a site in a rural area close to Bucharest. There we employed two different network configurations,

which we depict in Figure 5: i) one used from 2011 until 2014, with the *Nokia N82* acting as a Bluetooth server and 3G gateway, with the phones *Nokia 6120* and *Nokia 6680* connected to it; ii) another network configuration used from 2014 onwards, with a WiFi router with 3G USB stick capability, a *Raspberry Pi model B+* connected to the router, acting as an iCam Bluetooth server for *Nokia 6120* and *Nokia 6680*, and the *Nokia N82* phone connected via WiFi directly to the Internet, since the WiFi router is more reliable than the Raspberry Pi.

In Table I we present most of our devices we installed iCam on. However, we tested the Symbian and Android applications on many other smartphones we do not mention here. From Table I we notice the low processing requirements of the iCam client for the devices because the CPU frequency can be as low as 220 MHz.

Other deployments are simpler than those above, relying, for example, on *Nokia N95*, *HTC Touch Cruise* or *Samsung GT i5500* smartphones and a WiFi router at a location in Bucharest, or just a mobile phone connected via its 3G connection to the Internet.

C. Differences between Mobile Phones and WSNs

The devices running iCam are basically an instance of *Wireless Media Sensor Nodes (WMSNs)*, which is a very general term for smart cameras such as dedicated nodes, camera phones, digital photography or surveillance cameras equipped with radio transceivers [20], [9].

Note that cell phones are different from dedicated *Wireless Sensor Nodes (WSNs)*: i) they consume considerably more power and have to be recharged from the electric grid every 2-6 days, since phones have an LCD display and powerful radio transceivers; ii) they are designed to normally be handheld devices, while WSNs can reach sizes of the order of millimeters.

The average power consumption of a Nokia smartphone with a standard operating iCam application is around 400 mW as we can see in Figure 7. This is considerably more than for WSNs, which take on average less than 50 mW for a sustained mode of operation. For example, a more recent device such as Sparrow 3 [21], which uses the Atmega128RFA1 chip integrating an 8-bit microcontroller and the radio transceiver on the same die, consumes **at most 45 mW**.

Note that mobile devices and WSNs can cooperate for environmental, medical or even parking assistant applications [22], [23]. However, in this project we want to use the mobile phone as a camera sensor node. After all, the smartphones are the Swiss army knife of consumer electronic, where the digital communications, *Portable Digital Assistant (PDA)* functionality, photography and location capabilities converge [24].

We note also that, due to mass production, old mobile phones can be cheaper than dedicated sensor node solutions, something observed also in [2]. The cost of a second-hand Symbian (e.g., *Nokia 6680*, *N95* or *N82*) or Android smartphone (e.g., *Samsung i5500* or *LG P500 Optimus One*) can be in the range of 20-40 Euros. For dedicated sensor nodes we have the following prices:

- the CargoNet node [25] had a cost of around 40 USD, without including its humidity sensor;
- the Gardena smart sensor with temperature, soil moisture sensors and WiFi transceiver costs 119 Euros.

We mention that the Raspberry Pi SBCs cost from 5 USD (model Zero) to 35 USD (model 3), but this offering does not include any sensors, radio transceivers or power chargers.

D. Comparison with Commercial Surveillance Cameras

While there is a very large offering for video surveillance, we consider our solution to come closest to simple indoors IP cameras.

For example, the DLink DCS-932L cloud camera, with no PTZ capabilities or the DLink DCS-5009L, with PTZ, are equipped with WiFi and cable Ethernet connectivity, have both a video resolution of at most 640x480 pixels, a microphone and a small IR lamp. Their price is 50, respectively 90 USD. They run firmwares released under an (L)GPL license, which can be downloaded from *http://tsd.dlink.com.tw/downloads2008list.asp?OS=GPL* - note that most other vendors do not open-source their firmwares. The cameras can store the media on their flash card or upload it via a standard protocol such as FTP to a server. The cameras can be accessed remotely via Dynamic DNS, through a simplistic web interface implemented on the device. To view the media they also offer a web interface on the company's *mydlink* website, a Windows desktop application named *D-ViewCam* or the *mydlink Lite* smartphone application for Android, iOS and Windows Phone, all being able to perform real-time video streaming from the camera. They can perform motion and audio detection, allowing to configure the detection parameters.

Other solutions range from very simple analog cameras up to very complex digital cameras equipped with very good optical sensors and IR lamps such as the *EarthCam GigapixelCam X10*. This camera has a 24 MegaPixels sensor, 17x optical zoom, 4G Internet and WiFi interfaces, PTZ, thermostatically regulated and corrosion-resistant enclosure and various accessories such as solar panels for autonomous operation. The price can range from 15 USD for very simple analog cameras up to a few thousands of USD.

E. Power Management in the iCam Phone Client

The phones running iCam need to operate for months, continuously. Even if they are equipped with a battery, this normally depletes completely in the order of 1-2 days when running iCam with the original configuration. Therefore, we need to connect the phones to the electric grid, to ensure a continuous operation mode.

If a power grid outage occurs, the iCam phone client is designed to enter in energy conservation mode: it uploads the media less often, hopefully preserving enough energy in the battery, until the electricity comes back. Also, when the phone battery charge gets below a certain specified level, which we call *BATTERY_LEVEL_THRESHOLD*, and the charger is not connected, iCam puts the phone in power saving mode by making $pauseInterval = 10$ hours, until the battery is well charged again.

iCam can adapt its functionality also depending on the time of the day with an apriori established scheme: for example, during the day we can take photos at a certain time interval and at night we can record videos, where the audio component becomes very valuable. This can also result in energy saving, for example during the night. Note that professional video surveillance cameras are normally switching automatically to night mode, which means that the infrared optical filter is no longer used and the infrared LEDs are turned on when the light intensity gets below a certain threshold.

We took particular care in iCam to power manage the LCD, by turning off the viewfinder immediately after the phone finished recording. An interesting trick we discovered on Symbian S60 phones is that we can make the keypad of the phone lock automatically in order to completely disable the backlight of the LCD while iCam is running.

F. Fault Tolerance

To cope with the iCam phone client software or device OS related malfunctions we rely on watchdogs to restart the application or the system.

The PyS60 environment for Symbian S60 has some bugs itself, and this makes a complex application such as iCam to misbehave in certain occasions. Even worse, we have experienced occasionally failures of the Symbian OS or even of the hardware. To address the software errors, we developed a watchdog to restart the iCam application if it deadlocks or crashes or the OS. The watchdog checks if iCam is deadlocked through a very simple communication mechanism: iCam creates an empty file around every 30 seconds, which the watchdog looks for to erase it. If the watchdog does not find this file, then it assumes iCam has deadlocked. Also, the watchdog takes care to restart the phone, if too many unsuccessful application restarts have happened recently - in this situation Symbian probably experiences some degradation and requires a restart itself.

This watchdog ensured iCam functions well for almost seven years, continuously.

In case the watchdog for S60 fails itself, we developed another small Symbian application, which simply restarts the phone if it receives a call from a number that is in a whitelist hardcoded in the program.

Raspberry Pi uses a hardware watchdog that restarts the system if it becomes unresponsive. Running iCam on Raspbian OS poses little concerns since the Linux distribution itself is very reliable, the only exception being an occasional failure of the Bluetooth driver, which we address with a system restart.

V. Interesting Findings

During the extensive experimentation with the iCam mobile application, we made a few mentionable discoveries.

A. Availability and Some Failure Statistics

The iCam client, together with the software or hardware watchdog installed on the system, achieves a high degree of availability.

Firstly, the Nokia smartphones we deployed were able to function in a continuous operation mode for almost seven years, in outdoors conditions, with temperatures ranging between -25° and $+40^\circ$ Celsius. No special casing was added to the phones, as can be seen, for example, in Figure 3. Also, during this period we changed the original Nokia rechargeable batteries only after 5 years, even if they were not new at the beginning and the phones were charged continuously from their power supplies.

While using the already 2-4 years old second-hand phones for extensive testing of the iCam application, we experienced various software and hardware failures. We briefly report what are the serious types of failures on these devices:

- a *Nokia 6120*, running continuously from July 2010 - June 2012, experienced five serious system crashes, in which the phone became completely unresponsive, with a black screen. Pressing the Power button did not help, so the only way to reboot the phone was to take out the battery and put it back and then power on the system.
- a *Nokia N82*, running continuously from Aug 2011 - June 2012, damaged the memory card and experienced one serious system crash.
- a *Nokia 6680*, running continuously from May 2010 - June 2012, irremediably damaged two memory cards, erroneously turned off six times. In one occasion, the power circuitry got damaged and we had to take the phone for repair at a GSM service.
- a rather weared-out *Nokia N95* phone's LCD display got damaged irremediably because of faults in the ribbon between the LCD and the motherboard. However, we were able to continue to use the phone like this by employing as display an external analog TV connected to it via its 3.5 jack TV-out connector. Also, the phone started developing from 2014 issues with discharging battery faster even if the standard power supply was always connected. This probably indicated a short circuit, so we put in parallel a more powerful 5V, 2A charger to be able to sustain this current drain. With these adjustments, we are still able to use the phone even today, but only infrequently. In one occasion, the power circuitry got damaged and we had to take the phone for repair at a GSM service.
- a *Raspberry Pi model B+*, running continuously from July 2014 until October 2016, created enough bad blocks due to the Flash memory wear out on its bootable microSD memory card, that the Linux system started failing once every few weeks and we had to replace it with a new one in order to make the system work well again.
- a *Samsung GT i5500* was able to run the MobileWebCam Android application for a total of about five months, between 2014 and 2016.

Also, our iCam system experienced other failures in the last two years: i) power blackouts of the electric grid, which deactivated immediately the WiFi 3G router and the *Raspberry Pi model B+* because they are powered from the grid; ii) the 3G Internet network experienced some minor problems. Such incidents happen once every few months. Also, the *Raspberry Pi B+* experiences rather often problems with the Bluetooth transceiver, which becomes unavailable to receive data anymore; such a problem happens once every few days and we simply fix it by rebooting the Raspbian OS.

Regarding the server, Google's *YouTube* is a great free online video service, with undebatable high availability, allowing virtually unlimited storage in volume and duration. We uploaded on *YouTube* with iCam a total of more than 1.3 million videos from 2011 until 2017. Note that a user can upload on *YouTube* a video every two minutes, on average, although there is no limit on the size of the video. Also, after uploading a video, *YouTube* takes a non-negligible amount of time to process it before posting it for viewing or download. In a reduced number of occasions, this became a problem since this delay was bigger than 15 minutes, disallowing us basically to view the fresh videos.

B. Quality of the Phone's Sensors

We also consider mentionable the fact the main camera of many smartphones has a very good quality for video surveillance. In fact, for example, *Nokia N95*'s 5 MegaPixels (MP) main camera is comparable to a compact camera sensor of 3-5 MP and is able to make video recordings at 640x480 pixels resolution. Even the *Nokia 6680* has a decent 1.3 MP camera. Also, these cameras behave well under dark conditions. If we want to improve the details by night, we can use a powerful visible light lamp or even an (invisible) IR illuminator, since the phone cameras do not filter completely the IR light.

We cannot overlook the good quality of the microphone, which is able to record reasonably strong sounds even at a distance of 20-100 meters in open field. The most sensitive microphone is found on the *HTC Touch Cruise* phone. An interesting analysis of the quality of the audio recording can be found in [5], as already discussed in Section II.

C. Memory and CPU Consumption

We note that for the Symbian OS, our iCam application has a very low memory footprint. Both versions of iCam running on PyS60 1.4.5 and 2.0 consume around 600KB of system memory when just started, with the UI being initialized, being ready to begin broadcasting. This should come as no surprise: PyS60 was advertised from the beginning, in 2002, as a great alternative to quickly run simple scripts on the phones, modestly equipped by today's standards, having about 8 MB of RAM and an ARM CPU at around 150 MHz.

However, the memory iCam consumes on Symbian increases over time when more work is done. For example, when recording a 30 seconds long video at 640x480x15 fps, the RAM memory used increases with up to 40 MB, which is reasonable since the compressed MPEG media file has about 6 MB and after recording we prepare a packet to send over the Internet.

Python uses on Symbian less memory compared to when running on Raspbian OS (Linux) or on Windows Vista for Intel x64, where it takes at the beginning 17,952 KB and 25,120 KB of RAM, respectively, because the Symbian OS compiler, at least from Symbian version 9, generates very small binaries and DLLs for ARM, partly by stripping away unnecessary symbolic data such as function entry points. For example, the binaries of Python 2.5.4 for PyS60 2.0.0 have a total of 480 KB, while for Windows x64 they have 2,800 KB. The virtual memory management in Symbian is also very simplistic and strict: it uses usually 4 KB pages, no demand-paging - so an entire code or data segment is allocated when the application starts executing, nor swapping and the heap space is bounded accordingly to the amount of RAM by the OS [26].

On Android 2.1 iCam requires a lot more memory from the start, namely 28,360 KB. The reason is that on Android there are a few software components cooperating to make the Python script work: besides the iCam application, which is started as an Android SDK Java application and invokes our Python script, there are also the Python virtual machine and the *Scripting Layer for Android (SL4A)* taking care to access the mobile APIs of Android.

Independent of the platform we use, we achieve a very low overhead for the interpretation of the Python code itself since we perform few computations in pure Python in iCam and most of the time-consuming operations performed are simple multimedia and I/O commands, which are handled directly by the OS.

PyS60 provides many Python bindings through *.pyd* extension modules to the native Symbian services such as multimedia, cellular, Bluetooth and sensors. *Nokia 6120* experiences an average CPU load when running iCam with the normal configuration with video resolution of 176x160 at 15 fps, of 9.8% in a 10 minutes interval.

On *Raspberry Pi model B+*, iCam, which does not perform any multimedia capture operations and acts only as a Bluetooth gateway to the Internet, uploading media to both the dedicated server and *YouTube*, achieves an average CPU load of 1.5% when running for 22 hours.

On Android, when iCam performs Android specific operations it invokes the *SL4A* layer and this call takes us from Python and native code to the Android SDK Java API, through the use of a *Remote Procedure Call (RPC)* mechanism [27], which adds a negligible overhead. On the Samsung Galaxy S phone, the average CPU load for running iCam for an interval of 150 seconds is 5.5% for a video of resolution 640x480, at 20 fps.

D. Energy Consumption Analysis for the PyS60 iCam

By using the *Nokia Energy Profiler* application, also called *Juice* [23] or other similar tools, people have reported the power breakdown of Nokia Symbian S60 mobile phones. Its major components are the wireless modems (WLAN, 3G), the CPU and the display backlight [23]. For example, *Nokia 6120* consumes at least 8 mW in complete standby mode and a maximum of 2219 mW during our experiments. Similarly, *Nokia E7* consumes at least 20 mW in complete standby mode, and, on average, 236 mW when sending data via Bluetooth, 816 mW when using WiFi, 1860 mW when using 3G and 287 mW when the LCD display is on.

The power consumption when running iCam with the standard configuration on *Nokia N82*, using WiFi or Bluetooth to transmit the media data is reported in Figures 7 and 8, respectively. For all traces, the phones wait for about twenty seconds before starting to record the 30 seconds long videos. The average power is 380 mW and 331 mW, respectively, which means the WiFi transceiver consumes 50 mW more power than the Bluetooth on average in the 120 seconds intervals we use for our experiments, all the other conditions being similar. Also, the power consumption on *Nokia 6120* is reported in Figure 6, with an average value of 335 mW.

We note that both *Nokia N82* and *6120* phones get into low-power mode in the biggest part of the interval when they do not perform recording or transmission. The phones consume on average during this time 103 mW and 94 mW, respectively, and at the minimum 68 mW. However, we notice the power consumption increases under the form of spikes at certain moments during these sleep intervals for all traces and we try to understand why. Fortunately, the *Nokia Energy Profiler* reports besides power consumption the CPU load and the WiFi traffic - we show this information in Figures 6, 7 and 8. The main reason we have spikes in power consumption during the sleep interval in Figure 7 is that the WiFi performs radio link management even if it is not transmitting data. Similarly, in Figure 6, we notice that there is an increase in power consumption about 15 seconds before the video recording starts because the CPU load gets high, close to 100% - the reason being the viewfinder is turned on, an operation that takes a fair amount of time on this phone model. Also, we take into account that the CPU power manager of Symbian is somewhat unpredictable - we do not know the exact CPU manager algorithm running on the Nokia phones, even if it is documented [28] and the last version of Symbian OS is now open-source. It is also worth mentioning that during the $pauseInterval$ = 120 seconds duration, iCam sleeps for just 30 seconds and performs some simple checks and then gets back to sleep, over and over again, until it starts recording.

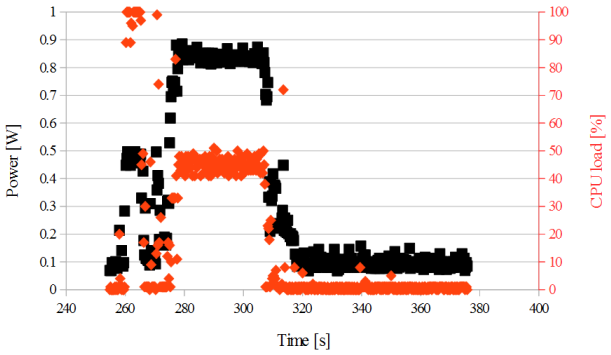

Fig. 6: The power consumption (black points) of *Nokia 6120* running iCam, obtained with the *Nokia Energy Profiler* application, together with the CPU load (red points).

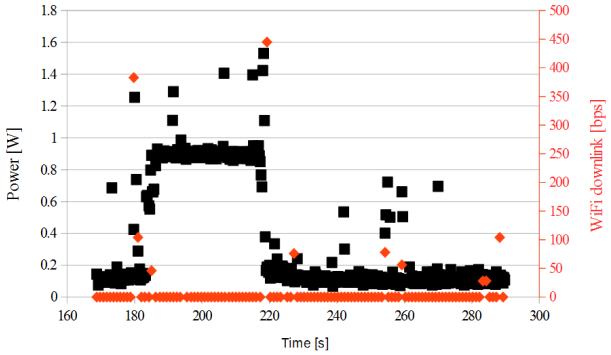

Fig. 7: The power consumption (black points) of *Nokia N82* when running iCam with WiFi Internet connection, together with the Internet download traffic (red points).

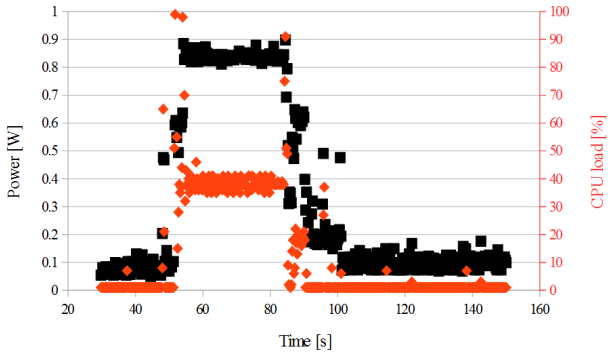

Fig. 8: The power consumption (black points) of *Nokia N82* when running iCam as a Bluetooth client, together with the CPU load (red points).

An experiment we performed with a different occasion was to run the *iCam* client on the *Nokia 6120* without the charger connected to it, the battery being somewhat weared out. In this scenario, we notice that the phone running *iCam* is draining the battery considerably

faster, in 12 hours, than when running without the *iCam* application started, in which case it depletes the battery in more than one day. The iCam client is basically in standby mode because it is waiting for the charger to be connected and it performs recording and media transmission via Bluetooth every $pauseInterval$ = 10 hours. Also, it is performing some simple checks, as already discussed, and writing short logging messages to the microSD card every 30 seconds, the latter being more costly. Because of the above operations, the client consumes itself on average around 60 mW, which explains why running iCam depletes the battery faster.

We have determined above that the average power consumption of a Nokia smartphone running iCam in standard operation mode is around 400 mW, as we can see for example in Figures 6 and 7. This implies that powering our device with a decent-sized (e.g., 10x10 cm^2) solar panel, as people have reported for WSNs - see, for example, Heliomote [29] - is not feasible. Doing standard media (video or photo) recording, even with a fair amount of power management performed on the phone, can be sustained only by larger solar panels of around 10 W, normally of dimensions 0.3x0.3 m^2.

VI. Conclusions

In this paper, we presented the iCam Internet-enabled video surveillance solution running on smartphones and Raspberry Pi, for almost seven years already. Our proposed iCam system uses a client-server architecture, where a web server, such as *YouTube*, receives media content from the devices running iCam and stores it to be viewed later. In the case we use at a certain location several phones with iCam, in order to cut down costs, the devices can form a star network topology, with only one gateway node connected to the paid Internet service via 3G, communicating with the other "local area" phones via Bluetooth or WiFi.

We wrote down also some considerations regarding fault tolerance, resource consumption and power management for the devices running the iCam application, the server and the iCam remote viewers.

Acknowledgments

We would like to thank the following people for their help and suggestions: Ricardo Mendonca Ferreira, Vlad Patras, Hamish Willee, Michael Haar, Răzvan Mihăilescu, Dan George Şotală and Ştefan Şuşu. We would also like to thank the anonymous reviewers and Nokia for kindly sponsoring us with a *Nokia E7* smartphone in 2011.

References

[1] G. Halfacree and E. Upton, *Raspberry Pi User Guide*, 1st ed. Wiley Publishing, 2012.

[2] P. Bolliger, M. Köhler, and K. Römer, "Facet: Towards a Smart Camera Network of Mobile Phones," in *Proceedings of Autonomics 2007 (ACM First International Conference on Autonomic Computing and Communication Systems)*, Rome, Italy, Oct. 2007.

[3] A. Kansal, M. Goraczko, and F. Zhao, "Building a Sensor Network of Mobile Phones," in *Proceedings of the 6th international conference on Information processing in sensor networks*, ser. IPSN '07. New York, NY, USA: ACM, 2007, pp. 547–548. [Online]. Available: http://doi.acm.org/10.1145/1236360.1236433

[4] A. Kansal and F. Zhao, "Location and Mobility in a Sensor Network of Mobile Phones," in *Proceedings of the 17th International workshop on Network and Operating Systems Support for Digital Audio & Video (NOSSDAV)*. ACM Press, Jun. 2007. [Online]. Available: http://www.nossdav.org/2007/files/file-31-session6-paper1-kansal.pdf

[5] S. Santini, B. Ostermaier, and R. Adelmann, "On the Use of Sensor Nodes and Mobile Phones for the Assessment of Noise Pollution Levels in Urban Environments," in *Proceedings of the 6th international conference on Networked sensing systems*, ser. INSS'09. Piscataway, NJ, USA: IEEE Press, 2009, pp. 31–38. [Online]. Available: http://dl.acm.org/citation.cfm?id=1802340.1802347

[6] Devisubox, "Boitier Photo Solaire," *information available at http://www.devisubox.com*.

[7] EarthCam, "EarthCam.net Jobsite Construction Cameras and Live Webcams," *information available at http://www.earthcam.net*.

[8] IPVM, "Video Surveillance Book 2013," *available at http://ipvm.com/report/surveillance_guide_2013*.

[9] M. Rahimi, R. Baer, O. I. Iroezi, J. C. Garcia, J. Warrior, D. Estrin, and M. Srivastava, "Cyclops: In Situ Image Sensing and Interpretation in Wireless Sensor Networks," in *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems*, ser. SenSys '05. New York, NY, USA: ACM, 2005, pp. 192–204. [Online]. Available: http://doi.acm.org/10.1145/1098918.1098939

[10] Google, "YouTube Data API," *https://developers.google.com/youtube/v3/getting-started*.

[11] ——, "GData Python client," *http://code.google.com/p/gdata-python-client/*.

[12] Digia, "PySide," *www.pyside.org*.

[13] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[14] SoX, "Sound eXchange," *http://sox.sourceforge.net*.

[15] D. Heun, "Getting Rid of Your Old Cell Phones Without Harming the Environment, http://www.pleaseconserve.com/getting-rid-of-your-old-cell-phones-without-harming-the-environment-2," 2010.

[16] EPA, "Environmental Statistic - Cell Phone for Soldiers," *http://www.att.com/Common/merger/files/pdf/CPFS_EarthDay/electronic_waste_fs.pdf*, 2008.

[17] ——, "Statistics on the Management of Used and End-of-Life Electronics," *http://www.epa.gov/osw/conserve/materials/ecycling/manage.htm*, 2009.

[18] C. Nicuţar, D. Niculescu, and C. Raiciu, "Using Cooperation for Low Power Low Latency Cellular Connectivity," in *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '14. New York, NY, USA: ACM, 2014, pp. 337–348. [Online]. Available: http://doi.acm.org/10.1145/2674005.2674983

[19] G. Shor, "How Bluetooth, UWB, and 802.11 Stack Up on Power Consumption," *EE Times, http://www.eetimes.com/document.asp?doc_id=1273546*, 2008.

[20] R. Holman, J. Stanley, and T. Ozkan-Haller, "Applying Video Sensor Networks to Nearshore Environment Monitoring," *IEEE Pervasive Computing*, vol. 2, no. 4, pp. 14–21, Oct. 2003. [Online]. Available: http://dx.doi.org/10.1109/MPRV.2003.1251165

[21] A. G. Marin and D. S. Tudose, "Energy Independent Wireless Sensor Network Design," in *2015 20th International Conference on Control Systems and Computer Science*, May 2015, pp. 267–272.

[22] F. Ingelrest, G. Barrenetxea, G. Schaefer, M. Vetterli, O. Couach, and M. Parlange, "SensorScope: Application-specific Sensor Network for Environmental Monitoring," *ACM Trans. Sen. Netw.*, vol. 6, no. 2, pp. 17:1–17:32, Mar. 2010. [Online]. Available: http://doi.acm.org/10.1145/1689239.1689247

[23] F. H. P. Fitzek and F. Reichert, *Mobile Phone Programming: and its Application to Wireless Networking*, 1st ed. Springer Publishing Company, Incorporated, 2007.

[24] A. Rome and M. Wilcox, *Multimedia on Symbian OS: Inside the Convergence Device*. Wiley Publishing, 2008.

[25] M. Malinowski, M. Moskwa, M. Feldmeier, M. Laibowitz, and J. A. Paradiso, "CargoNet: a Low-cost Micropower Sensor Node Exploiting Quasi-passive Wakeup for Adaptive Asychronous Monitoring of Exceptional Events," in *Proceedings of the 5th international conference on Embedded networked sensor systems*, ser. SenSys '07. New York, NY, USA: ACM, 2007, pp. 145–159. [Online]. Available: http://doi.acm.org/10.1145/1322263.1322278

[26] A. S. Tanenbaum, *Modern Operating Systems*, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2007.

[27] P. Ferrill, *Pro Android Python with SL4A*, 1st ed. Berkely, CA, USA: Apress, 2011.

[28] J. Sales, *Symbian OS Internals: Real-time Kernel Programming*, ser. Symbian Press. Wiley, 2005. [Online]. Available: https://books.google.ro/books?id=eKMeAQAAIAAJ

[29] A. Kansal, D. Potter, and M. B. Srivastava, "Performance Aware Tasking for Environmentally Powered Sensor Networks," *SIGMETRICS Perform. Eval. Rev.*, vol. 32, no. 1, pp. 223–234, 2004.