cBox: A decentralized system for resources sharing that enables peer communication in an heterogeneous environment

Konstantinos Akasoglou, Alexandros Baltas, Emannouil Gkatziouras, Nikolaos Kapetanos, Dimitrios Karavias, Konstantinos Moustakakis, Georgios Oikonomou, Nikolaos Palaghias, Georgia Papaneofytou, Nikolaos Triantafyllis , Filippos Vasilakis, Orestis Akribopoulos, Christos Koninis, Marios Logaras and Ioannis Chatzigiannakis

[1] Research Academic Computer Technology Institute (RACTI), Patras, Greece
[2] Computer Engineering and Informatics Department (CEID), University of Patras, Patras, Greece
Email:{akasoglo, ampaltas, gkatzioura, kapetanos, karavias, moustakaki, oikonomou, palaggias, papaneofyt, triantafyl, vasilakis}@ceid.upatras.gr
{akribopo, koninis, logaras, ichatz}@cti.gr

Abstract

The cBox system is a collection of services, libraries and applications that can be used to share resources, such as Internet connectivity, with other members of an ad-hoc network, in a transparent and safe way. The system features include delay tolerant networking, caching of the web requests and fully decentralized operation. The system design exploits well established technologies such as ZeroConf, mDNS to allow the devices to discover compatible services. Also it utilizes a wide variety of existing technologies and devices to facilitate communication. Furthermore, we provide a set of applications and a use case scenario showing the usage of the cBox system.

Keywords: Wireless Sensor Networks, Android, Delay Tolerant Networking

1. Introduction

In the last decade it has become obvious that we have entered the knowledge society and everyone must have access to participate. That is why Internet access in some countries(e.g. Finland) has been ruled *a base human right* and in other countries is considered part of their basic infrastructure like roads and water. However there are occasions, where Internet connectivity is lost or unavailable due to technical reasons or as a consequence of natural disasters. In other cases the access may be partially or fully blocked due to different ISP policies. In addition many users are becoming increasingly concerned about their privacy and recognize the need to ensure the security of their confidential data. In some situations exchanging data through direct channels is not preferred by the user because the main infrastructure has been compromised resulting in privacy inadequacy and security leaks. In such cases, alternative routes need to be

found or even created, exploiting various communication channels (i.e. wired, wireless, broadband, mobile).

In a representative scenario where alternative and multiple communication paths need to be found, consider a user that tries to send a small message from his laptop(i.e. tweet or mail). His broadband Internet connection is not operational. However, another user within range of his wifi is online via his smartphone with a 3G connection and he is willing to share his connection. In this case, we need a way to facilitate the communication between the two devices, in order to connect the offline user to the Internet.

The cBox software system is designed and developed to address those issues in a reliable, secure and efficient way. Based upon building and maintaining a peer network among trusted nodes willing to participate, the cBox software can be executed in a large variety of devices with no special hardware requirements, i.e. Android based devices, Personal Computers or small, low power devices such as Beagleboard [Beagleboard, 2011], Dreamplug [GlobalScale, 2011]. Resilient routing techniques ensure cBox's high reliability inside a network where nodes can be added or removed and act as Internet gateways.

cBox employs different adapters to facilitate the communication depending on the hardware that it runs. When executed on a device carrying a Bluetooth adapter, cBox software carries out any communication tasks using the Bluetooth interface –if that is possible. If cBox is executed on a more powerful device with Wifi, Ethernet and Bluetooth interface, then the software utilizes all three of them accordingly. The services and connections provided by the participating nodes depend on users preference and are fully customizable.

2. cBox System Architecture

In Figure 1 we present the high level architecture of our system, broken down in multiple layers. At the lowest is the Network Abstraction Layer (NAL) which conceals the differences of the underlying communication interfaces (i.e Bluetooth, Ethernet), providing functions for sending and receiving data in an standard way. NAL includes as well, the Android Connectivity module, which is the implementation of NAL for Android-based devices. The layer also includes a database for the accessing discovered services, that are needed by the upper layers. The next layer, (i.e. Protocols) includes Routing and ZeroConf. Routing layer maintains routes to every node participating in the network and delivers the packets to their appropriate destinations. ZeroConf is responsible for assigning unique hostnames to the nodes and discovering neighboring nodes running compatible services. Proxy is the following layer providing anonymity and caching. The higher layer is the Application, that provides multiple ways to interface the cBox system with popular and useful applications and interact with the end users.

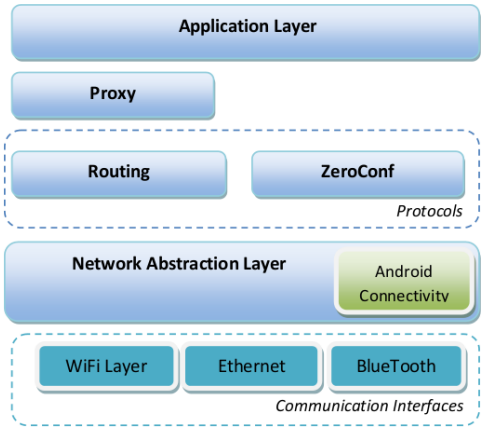

Fig. 1. The Layers of the cBox

Wiselib library

The Wiselib [Baumgartner et al., 2010] is an algorithm library initially designed for sensor networks but expanded to more general wireless networks. We decided to implement our algorithm using Wiselib: a code library, that allows implementations to be OS-independent. It is implemented based on C++ and templates, but without virtual inheritance and exceptions. Algorithm implementations can be recompiled for several platforms and firmwares, without the need to change the code. Wiselib can interface with systems implemented using C (Contiki), C++ (iSense), and nesC (TinyOS). Also currently it is ported to support SmartPhones operating systems like Android and iOS.

2.1 Adding support for Android OS to Wiselib

Android's mobile operating system is based on a modified version of the Linux kernel. The Android open-source software stack consists of Java applications running on top of Java core libraries executed on a Dalvik virtual machine featuring JIT compilation. Although most applications are written in Java code, libraries written in C, C++ and other languages can be compiled to native code and installed with the Android Native Development Kit, using a rather mature framework, the JNI [ORACLE, 2011]. This enables Java code running in a JVM to call and to be called by native applications and libraries written in other languages, such as C, C++ and assembly. Native classes can be called from Java code running under the Dalvik VM using the System.loadLibrary call, which is part of the standard Android Java classes.

As with other C++ template libraries like the Boost [Boost, 2011] or CGAL [Cgal, 2011], the fundamental design parts in Wiselib are concepts and models. A concept is an interface description, only existing in documentation. It describes exactly both the

types that are defined in a class, and the provided methods with expected parameters. A model in turn is the implementation of one (or more) concepts - each type and each method that is described in the concept must be provided by the model.

As a first step in order to be able to compile existing or new algorithms for this platform, we developed the models for android platform. During the development of cBox, most concepts were implemented for Android. Specifically, *Debug*, *Clock*, *Timer* and *Radio* models, as being the most important, were fully implemented. The Debug model just prints a string given in native C++ code to an alert message inside the application using the Java alert box. Clock model is a boost-like clock for android platform, providing the applications with time related information. Timer model, is used to schedule the events to be executed in the future. The timer has been implemented using setitimer system call which sets an interval timer and when expired a Unix signal(SIGALRM) is sent to the process. Finally, the radio model implements the radio interface. When the application starts, the radio is enabled. Upon receiving a new message, Android's Java code calls through JNI a native function that notifies any registered callback that a message was received. To send a message, native functions store in an XML-like string the message properties and sent it to the Java code through JNI. Java code restores all the necessary info(IP,header,data etc) and sends the packet.

2.2 Network Abstraction Layer

Network abstraction layer (NAL) is located at the lowest system layers, along with Android Connectivity Component. The main goal of this layer is to provide a highly abstract network interface to upper levels. Network interfaces on a computer system may use different network technologies (i.e. Bluetooth, Ethernet, Wifi, GSM) and interfacing them requires the use of different techniques. Network Abstraction Layer abstracts all the details and differences and provides a simple, unified way to access every available network interface.

The NAL API provided is quite minimalistic, consisting of few simple functions that are used to access every available Network Interface. NAL is implemented using the Wiselib in which a new Wiselib Radio concept was conceived and developed for every supported Network Interface (Bluetooth, Ethernet, Wifi).

Sending data is a straight forward procedure: a new thread is created with a buffer holding the data to be send. The thread opens a socket connection and sends the data to the destination. On the other hand, Data Reception is asynchronous. In order for an application to receive data, a callback function must be registered with NAL, which will be called when new data is available. Based on Wiselib's requirements, delegates mechanism is used to facilitate the callback mechanism.

The implementation of NAL is in C++ using Wiselib extensions. For the Ethernet and Wifi Interface, standard socket API is used, which is provided by every Linux distri-

bution. For the Bluetooth Interface, library Bluez [Bluez, 2011] is used which provides UNIX-like sockets for Bluetooth adapters.

In the following example an upper layer using the NAL API to broadcast a message to every neighbor:

```
Ethernet e; //Instance of Ethernet NAL implementation
e.enable(); //Enable and initializes interface
//Register data receiver handler
e.register_receiver<Layer,&Layer::handler_function>(this);
e.send(``CBox!``,BROADCAST); //Send a (roadcast message
//When communication is no more needed, disable is called
e.disable();
```

Android Connectivity

The main purpose of Android Connectivity is to provide to the higher layers an abstract and simple way to send and receive data using an Android phone.

Considering the connectivity through Wifi we had two ways to achieve our goal either through UDP or TCP. The decision was to use UDP in occasions where the size of the data to be transferred was less than 512 bytes. This transfer is achieved through a client server model between Android phones and cBox devices using the DatagramSocket, DatagramPacket classes from the Java.net library. In an android phone the client is achieved with a "send" function sending a UDP message to a UDP server of another cBox device. For the Android phones we use Android services and threads for the UDP server. By this way the user is able to run the cBox UDP server on the background, thus being able to use his mobile phone as usual.

Considering the connectivity through Bluetooth the only way provided from an Android phone was the rfcomm layer. cBox follows the client server concept from the Wifi connectivity using the Android Bluetooth API. However in order to transfer data using the Android Bluetooth API, it is mandatory to first pair the devices, establishing a connection between them before data transfer is allowed. The connectivity through Bluetooth is fully multi-threaded, using classes from the Android Bluetooth API.

Another issue is how the Android Connectivity deals with the data storage. The data can be either necessary information for the higher levels or files of various formats, probably used by applications integrating with cBox. For the first situation cBox takes advantage of an Android phone's sqlite database and content providers. The information needed by the upper levels is stored and retrieved through SQL queries. Information needed by other applications integrating with cBox are available through a public content provider. Also the files to be received or to be sent are stored on the sdcard directory which is public.

2.3 Routing

Routing is the layer responsible for creating and maintaining the network, while forwarding every message to its designated destination. In particular, the following four tasks are executed: a) Create the network and allow peers to join at any time, b) Assign each node with a unique address, c) Design a self-repairing mechanism, that preserves the network consistency when peers leave or crash and d) Route data messages.

Network Construction: Our routing algorithm uses a tree structure as the backbone of the network, with each node representing a peer. One node initializes the tree by making itself the root node, awaiting for nodes wishing to enter the tree. Let N_i be a node that is part of the tree network and N_j a node that wishes to join the network. The *join* algorithm operates as following: N_j broadcasts a join request message and N_i replies. Then N_j sends back a reply to let N_i know that N_j is its child and updates its status.

Unique Address Assignment: This protocol uses IPv6 [Deering and Hinden, 1998]-like addresses to identify each node by assigning a different address to each peer participating in the tree structure. Each node utilizes two hexadecimal digits, while inheriting its parent address as a prefix (see Figure 2). Non valid digits are ignored. Root utilizes only 2 digits, nodes with height 1 utilize 4 digits etc. For example, if a node has the address `0x0003af` then node `0x0003` is its parent. Each node can have up to 256 children and height up to 15. For this property to be possible, the *join* algorithm implements the following mechanism. When N_j sends the join request, N_i replies with a message indicating that N_j can be its child with a specific address computed by N_i. Then N_j has to inform N_i that it obtained the address and is now taken.

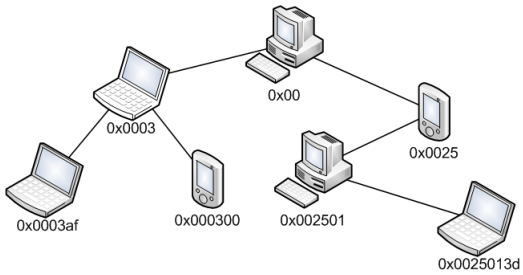

Fig. 2. IP assignment at a cBox network that consists of computers and Android devices.

Self-Repairing Mechanism: The network is highly dynamic; nodes can leave the network or can crash at any time. The protocol does not implement any *leave* algorithm

so leaving and crashing nodes will be treated equally. In order to detect the possible absence of its parent, each node periodically pings its parent and waits for response. If no response is received during a predefined time, the node broadcasts a re-join message using an algorithm similar to the *join* one. The node, then, forces the update of all children addresses. When a node address is updated by its parent, it recursively updates its children with consistency to the protocol explained above.

Message Routing: The way addresses are assigned gives the peers the ability to route every message with the optimal way, by only knowing their own IP address and the destination IP address. When a node S, creates a message, it compares its IP, IP_S, with the destination peer IP, let IP_D. If IP_D is shorter than the IP_S, S forwards the message to its father. Otherwise, if the IP_S is a prefix of IP_D and the peer with IP_D is a member of a subtree with S as a father, so it forwards the message to its children. If IP_S is not a prefix to IP_D, S forwards the message to its father. When node A receives a message, it compares its own IP_A with the IP_D. If IP_A is equal to IP_D, node A is the destination and handles the message. If the two addresses are of the same size, A drops the message. If the two addresses are not of the same size, A executes a routing algorithm similar to the one described above.

A node has also the ability to send a message to all other peers. The message gets forwarded by an algorithm similar to breadth first search, as each node forwards the message once to every neighbor.

2.4 ZeroConf

ZeroConf [Aboba et al., 2002] is a module incorporated in the Routing layer. It is a variation of the common ZeroConf protocol and it is implemented in Wiselib. The two main tasks of the protocol are: a) the uniqueness of the nodes' hostnames and b) the implementation of the sharing of the services.

Hostname uniqueness: Each node is characterized by a unique IP (routing is responsible for its validity) and a hostname. When a device finds an existing network and a suitable IP is assigned to it, the device "asks" the network if its name already exists. If no node has that specific name, the device joins to the network and the hostname is equal to its name. If another node has the specific name, it sends a "conflict" message to the node-candidate which now has to change its name to a different one. The search procedure for a unique hostname is the same for the new name, too.

Service Discovery: When a node joins the network, it advertises its services with broadcast messages. The services which are provided from a node are listed to a folder that the node reads. When a node receives a record of a service that exists in the network,

it stores it temporarily in its database where all of the available services are stored. The record includes some information for the service, i.e, IP address, the node's hostname, the port that uses. Moreover, the record has a TTL (time-to-live) value, which is used for validity of the service. So, a service can be used only if its TTL is a positive value (fresh service). If a service is about to expire, the node can ask the provider of the service if it is still available. The provider will respond with a new TTL if the service exists. The database of each node deletes the record of a service if its TTL is not suitable (stale service).

There is also the ability of a node to ask other nodes if they have a specific type of service. Every node receiving that message, looks locally for this type of service and in case that the specific service exists, a response message is broadcasted.

A node which wants to use some services from the network, needs only to select a given service from its database. Then it sends its specialized query to the destination that the record includes.

2.5 Proxy and Application Layer

One of the most important services that the cBox system can offer to the end-user is transparently caching web content and utilizing multiple nodes caches in order to reduce the generated Internet traffic and increase the response time. In the most typical scenario the cBox system is running in a client with Ubuntu Linux distribution. Due to this major advantage and based on software reuse, we decided that Squid [Squid-Cache, 2011] Proxy Server was the most suitable choice. Squid is an open source, high performance proxy caching server for the Web, supporting HTTP, HTTPS, FTP and more. It provides many capabilities such as: *Anonymity*, and *Caching*. User is able to perform transactions with the Web, without his personal information being published. The Proxy server is caching and reusing frequently requested web pages and has also the ability to configure hierarchical caches.

The Application Layer is the highest layer in our project and includes the implementation of two clients: one for Twitter and one for Flickr that use the cBox system. Each one of those clients consists of two parts: Android Mobile Application and cBox Caching Proxy Application. The functionality of each client is similar, on the way they interact with the Web. The Android Application part of the Twitter Client allows the user to sign in to Twitter from various accounts using OAuth [E. Hammer-Lahav, 2010] (open protocol allowing secure authorization in a simple method). Users are able to send their own update status to Twitter, with ensured arrival guarantee, regardless the availability of Internet connectivity. A push notification mechanism is used for delivering notifications to the mobile phone. Moreover the Twitter Client provides all the functionality of the Twitter website, i.e, Direct Messages, Timeline view, Friends view etc.

The cBox Caching Proxy Application part of the Twitter Client provides the ability to cache Twitter status updates on Internet connectivity loss. The Twitter status update may originate either from the above application or during the use of a web browser. The implementation for the Flickr follows the same techniques and strategies as the Twitter client we described above.

3. *Use Case*

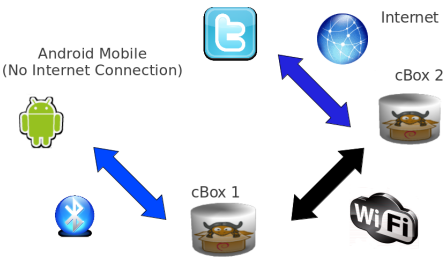

Fig. 3. Simple Use Case Scenario

Suppose a user of an Android SmartPhone is in proximity of a cBox device which has Bluetooth and wifi interfaces. The mobile device does not have Internet access and the user opens the cBox twitter app and tries to send a status update. The request is cached in a database of the local device until it reestablishes Internet connection. In the meantime, it sends the status update to the neighboring cBox device. If neither the cBox device has Internet connectivity, it caches the status update and it also forwards the status update to an other cBox device, see Fig 3. The moment, one of those devices regains Internet connectivity, it sends the status update. If all of the devices regain Internet connectivity, the status update is sent three times but Twitter is able to handle the specific issue, so that the message is not displayed more than one time. When the status update is delivered, Twitter's application server sends to the mobile device a push notification, via "Google's Cloud to Device Messaging Framework", in order to inform the user that the status update was successfully delivered.

4. *Conclusion and Future Work*

cBox is an open source reliable and decentralized system that can be used to facilitate communication anytime, anyhow, anyplace. It provides the choice to connect and share resources independently. As we saw in the use case, such networking is functional

and a transition in a large scale scenario is meaningful. Further work can be made on expanding the variety of services that can be offered. More application clients can be implemented i.e. email and ftp clients. Improvements can be also made on the security, such as adding cryptographic functions for the message delivery. Our goal is the wide usage and acceptance of our software from the open source community as is publicly available[3].

References

Aboba, B., Thaler, D. and Guttman, E. [2002], 'Zeroconf multicast address allocation protocol (zmaap)'.
URL: *https://tools.ietf.org/html/draft-ietf-zeroconf-zmaap-02*

Baumgartner, T., Chatzigiannakis, I., Fekete, S. P., Koninis, C., Kröller, A. and Pyrgelis, A. [2010], Wiselib: A generic algorithm library for heterogeneous sensor networks, *in* 'Proceedings of the Seventh European Conference on Wireless Sensor Networks (EWSN 2010)', Springer-Verlag LNCS 5970, Coimbra, Portugal, pp. 162–177.

Beagleboard [2011], 'Low cost, high performance, low power omap3 based platform'.
URL: *http://beagleboard.org/*

Bluez [2011], 'Library for the core bluetooth layers and protocols'.
URL: *http://www.bluez.org*

Boost [2011], 'peer-reviewed portable c++ source libraries'.
URL: *http://www.boost.org*

Cgal [2011], 'Computational geometry algorithms library'.
URL: *http://www.cgal.org*

Deering, S. and Hinden, D. [1998], *RFC 2460 Internet Protocol, Version 6 (IPv6) Specification*.
URL: *http://tools.ietf.org/html/rfc2460*

E. Hammer-Lahav, E. [2010], 'The oauth 1.0 protocol'.
URL: *http://tools.ietf.org/html/draft-hammer-oauth-10*

GlobalScale [2011], 'Developer kit for the marvell pxa510 highly integrated soc'.
URL: *http://globalscaletechnologies.com*

ORACLE [2011], 'Java native interface specification contents'.
URL: *download.oracle.com/javase/6/docs/technotes/guides/jni/spec/jniTOC.html*

Squid-Cache [2011], 'Squid internet object cache'.
URL: *http://www.squid-cache.org*

[3] The source code of all the developed components is publicly available in the github repository: `https://github.com/CEID-DS/cbox`