MOnKey - A Portable Middleware On Key

Jérémie Albert, Jérôme Castang, Serge Chaumette

MONKEY - A PORTABLE MIDDLEWARE ON KEY

Jérémie Albert
LaBRI
University of Bordeaux
351 cours de la Libération
33405 Talence, France
email: jeremie.albert@labri.fr

Jérôme Castang
LaBRI
University of Bordeaux
351 cours de la Libération
33405 Talence, France
email: jerome.castang@labri.fr

Serge Chaumette
LaBRI
University of Bordeaux
351 cours de la Libération
33405 Talence, France
email: serge.chaumette@labri.fr

ABSTRACT
More and more computing resources are available everywhere. They are also more and more connected. The drawback is that these resources and network connections are volatile and heterogeneous. To cope with these problems we propose a portable middleware that supports the characteristics of these versatile and unstable configurations. It is called MOnKey and it is the topic of this paper. This work is partly supported by the French Agence Nationale de la Recherche under contract ANR-05-SSIA-0002-01.

KEY WORDS
Mobile ad hoc networks, middleware, portable applications.

1 Introduction and rationale for a middleware on key

In the last decade the number of computers has increased significantly, and there is hardly any place you can go where you will not be able to get access to either a computer or a mobile terminal. Furthermore, these resources are most likely to be equipped with a communication technology (WiFi [1], Bluetooth [2], etc.). This is a good news because you can access a number of services from anywhere, but it also raises a number of problems.

Applications must be available everywhere. For a single user, having to deal with several computers implies being faced with several different software configurations. This means that it may be the case that part of his software environment is not available or working differently depending on which computer he is using. To avoid these problems, portable applications have been created. Portable applications are applications that are designed and compiled so as to remain insulated within a portable device, such as a USB key. It basically consists in making so that every file access refers to the portable device itself and that the applications are statically linked. Furthermore, dedicated pieces of hardware have been designed for these applications, such as the U3 USB memory keys. These keys provide an autorun mechanism that usually starts a minimum desktop-like environment so that no interaction at all is required with the host computer. Many software suites now offer releases that are compliant with this approach (Firefox, Thunderbird, GIMP, OpenOffice, etc.). There are also a number of companies or organizations that provide compilations of portable applications (Framakey [3], Liberkey [4]). Thanks to this approach, a user can always carry with him the different pieces of software he is used to work with, in the precise configuration he is accustomed to, and that will run on any binary compatible computer[1].

Heterogeneity of devices must be supported. The price to pay for this wide availability of computing resources is heterogeneity. The capabilities of these devices (PC, PDA, mobile phones, sensor nodes, etc.) can be really different. This is true at the hardware level (battery capacity, etc.), at the level of the operating system (Linux, Windows, Symbian [5], etc.) and at the level of the supported communication technologies (Ethernet, WiFi, Bluetooth, ZigBee [6], etc). The portable applications as described above solve the problem of having them available at many places, but this only works provided the machine features remain the same. We believe that such a limitation is not realistic in today constellation of mobile terminals: it should be possible to run the applications everywhere. This thus requires the support of an underlying middleware that will hide the differences between the different platforms. This middleware is then portable, the term "portable" referring to the "portable applications" as described above.

The instability of the communication links must be dealt with. In this constantly evolving context (users are moving, machines are moving, new users are joining or leaving the community), it is impossible to assume stable communication links. At any given time, any communication link between two nodes in the network may be broken because of some external, unpredictable event (some noise or interferences for a wireless connection, a device in a car entering a tunnel that thus loses the network, a physical link failure for a wired connection, etc.). When dealing with stable networks, it is possible to install a service at some given node, being sure that it will remain available during

[1] It should also be noted that this makes it possible to avoid installing a large number of software on all computers to fulfill the needs of all users.

a reasonable amount of time. For instance this is a basic assumption in a PKI [7] that makes it possible to setup a certification authority. In the real mobile and unstable context in which we are working, peer-to-peer is the only approach that makes sens. This is the approach that has to be implemented in the portable middleware described above if we want to be effective in a versatile network.

To summarize our contribution, we have designed a portable middleware on top of which it is possible to develop applications that will be supported on any device with any communication technology. We call this middleware MOnKey, that stands for Middleware On Key.

The price we have to pay to reach this goal is that the resulting middleware can only provide low level primitives that it will be possible to support on any platform and that will make sens whatever the context. This also leads to limited communication capacities and broadcast is thus the basic communication primitive. Nevertheless we have shown in [8] that it is still possible to develop realistic applications. An example is the topic of section 3 of this paper.

The rest of this paper is organized as follows. In section 2, we motivate and show the relevance of our work based on the context described above an on a description of some of the significant look-alike existing systems. We then present the MOnKey middleware itself. Thereafter, so as to illustrate our work, we show in section 3 a proof of concept application and, in section 4, our first results. We eventually conclude in section 5 and introduce some future research directions.

2 Middleware On Key - MOnKey

In this section we present MOnKey, the portable middleware that we have designed.

Let us recall that we are in the particular context of ad hoc networks. We do not make any assumption about the underlying network and the device used. These could be PC, PDA or even mobile phones. We also consider that links between nodes, when they exist, are not permanent. A link failure can occur at any time for several reasons: a device can move out of range, the key where the MOnKey middleware is installed can be untimely removed, etc. We assume that each device has an unique identity and at least one network interface. We do not make any other assumption about devices capabilities and network characteristics. Note that the security problems are out of the scope of this paper but are one of our major research directions (see "Future work" in section 5).

Related work. Most of the middleware designed for this kind of networks hide the mobility of the nodes (figure 1). They claim they deal with the dynamic aspect of the nodes and communication links, but they allow an application to

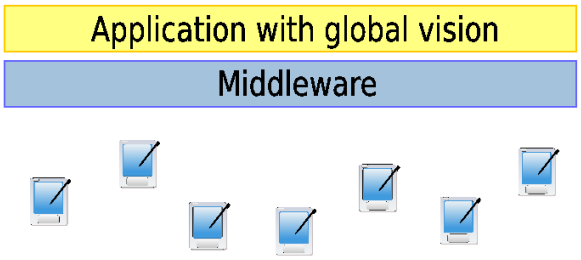

Figure 1. Usual approaches

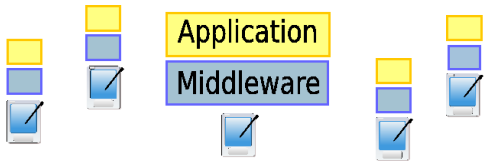

Figure 2. Our view

be supported as if it were in a static environment. They rely on routing algorithms such as AODV [9] or OLSR [10]. We claim that in a really unstable network, where any node and/or communication link can appear/disappear at any time, no routing can be achieved. It can even be the case that the network is splitted.

Our approach. Contrary to other the approaches, we neither provide routing between the nodes of the system nore hide the volatility of the underlying system. We rather allow a program to be aware of the dynamic aspect of the underlying network and then to take it into account, by providing contextual information based on a callback mechanism.

For example, to deal with the volatility of nodes, a MOnKey node announces itself thanks to a beacon mechanism. It collects the beacons coming from its neighbours and reports the presence of these neighbours to the application by means of two callbacks which are *onNodeDeclaredPresent* and *onNodeDeclaredMissing*.

MOnKey also offers two primitives to send/receive a packet to/from the network interface(s). The arrival of a message causes *whenReceived* to be invoked. Emission is supported by means of a one-way broadcast primitive named *drop*.

The main objective of MOnKey is to provide a minimal and universal set of primitives that we call Neighbourhood and Context Interaction Primitives (NCIPs) [8].

Implementation choices and current prototype. We have developed a prototype of MOnKey that is written in Java[2]. We have chosen this language because it is available

[2] Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. The

on most of the devices that we are considering. Even on the smallest of them, such as sensor nodes (SunSpot [11]), mobile phones or smart cards, light versions of Java are provided.

Till now, the prototype implementation focuses on PCs. In this context, so as to avoid to make any assumption about the availability of a Java Virtual Machine on each node of the network, we use the compiler *Excelsior JET* [12] that produces binary code.

As of writing, MOnKey supports several communication technologies such as Ethernet, WiFi (over UDP/IP) and Bluetooth.

3 Proof of concept - A portable P2P Application

To emphasize the usability and usefulness of our middleware MOnKey, we have designed a peer-to-peer application, the goal of which is to allow users to share/send files.

Goals. This application enables a user to send a file to his direct neighbours. We call direct neighbour a node accessible in one hop (single broadcast), whatever the network technology[3]. The file is transferred using the "best" technology available between the two nodes. The selection of the best communication technology depends on several criteria such as power availability and network characteristics (bandwidth, latency, range of radio technology, etc). From these criteria, it is possible to create policies to automatically select the best technology to use for a given transfer. As of writing, the only criteria that is taken into account is bandwidth.

Furthermore, since a node can appear/disappear at any time, we have to handle a resume mechanism.

Scenario. In the scenario described in figure 3, we assume the peer-to-peer application is installed on a USB key plugged in a PC, and on a memory card inserted inside a PDA. One of the users initiates a transfer (figure 3, top) and then moves out of range of the other user/device. The transfer resumes automatically (figure 3, bottom) when they are once again direct neighbours. Also note that MOnKey provides an abstraction of the network layer so that a transfer can be initiated and started using Bluetooth and then completed using WiFi or vice versa.

Neighbourhood awareness. Let us recall that we do not make any assumption about the network topology or the mobility of users. We thus need to handle neighbourhood

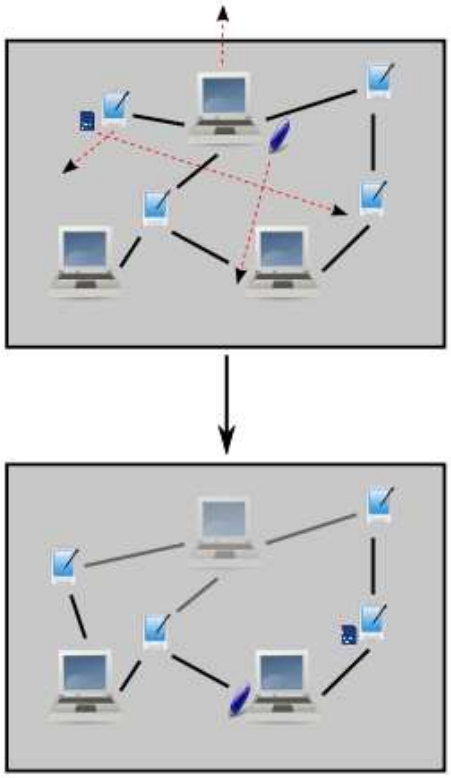

Figure 3. Mobility scenario

evolution. Therefore, we use the MOnKey API and more precisely the functions *onNodeDeclaredPresent* and *onNodeDeclaredMissing* to manage a set of *supposedly* accessible direct neighbours. Again, we call direct neighbour a node that can be reached in one hop, whatever the network technology. These direct neighbours are *supposedly* accessible because in the context of mobile ad hoc networks, we cannot guaranty that the presence information, even though it is true when collected, is still true when it reaches the user level. A node detected as gone can be back when the information reaches the top level of the middleware.

IO operations. To reach direct neighbours, we use the *drop* primitive provided by the MOnKey API. This primitive broadcasts a packet and there is no guarantee about its delivery. Symmetrically, the *whenReceived* callback is called when a packet is received. This callback function has to be implemented by the application. Algorithm 1 shows how received packets are managed depending on their type in our peer-to-peer application.

Prototype. As explained in section 2, we have a prototype running. It has been implemented in Java and we have used *Excelsior JET* [12] that supports static linking and binary code generation. We have run the middleware and the application in several different configurations and this has

authors are independent of Sun Microsystems, Inc. All other marks are the property of their respective owners.

[3] In case of a wired network, this thus does not necessarily mean a direct link.

Algorithme 1 : whenReceived(byte[] b)

```
Input : byte[] b: data received
Global : The transfer session set
SideEffect : A new transfer session can be
             created; Packets can be dropped
begin
    switch type of b do
        case "TSInitPacket"
            create a new transfer session;
            drop all packets of this transfer session
            on the network;
        case "TSDataPacket"
            save data from this packet in a
            persistent memory;
            get the transfer session corresponding
            to this packet;
            add the received packet identifier to
            the received packet identifier set;
            if all expected packets have been
            received then
                write the file in persistent memory;
                remove this transfer session from
                the current transfer sessions set;
                notify application that the transfer
                succeeded;
        case "TSRedropPacket"
            get the transfer session corresponding
            to this packet;
            if transfer session does not exist then
                create a new transfer session;
            get the packet asked for;
            drop this packet on the network;
end
```

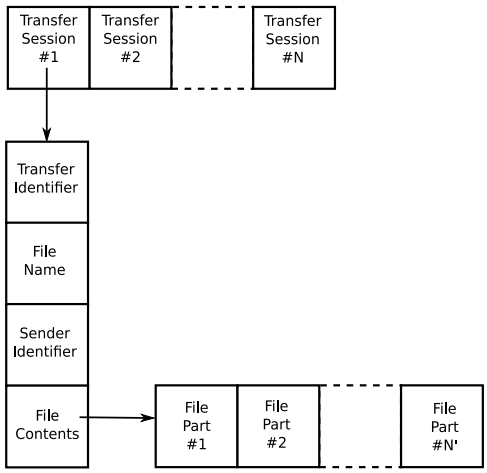

Figure 4. Data structure

shown that it is working properly. We have not done efficiency measurements to date, but we are currently setting up such an experiment. Furthermore, we are also preparing a large scale simulation using the Madhoc [13, 14] engine.

Data structure. Figure 4 shows how the transfer management is implemented. Each receiving node stores the set of transfer sessions which are not completed. This allows the receiving node to ask again for a file for which it misses packets until the transfer completes. A transfer (*Transfer Session* field) has an identifier (*Transfer Identifier* field) that makes it possible to have several transfers at the same time with an identical file name (stored in the *File Name* field).

4 First results and a first possible improvement direction

We have a prototype running. We have developed a prototype of our middleware on key that provides an API which contains some of the Neighbourhood and Context Interaction Primitives (NCIPs) defined in [8]. More precisely, it provides the functions *onNodeDeclaredPresent*, *onNodeDeclaredMissing*, *whenReceived* and *drop* discussed in section 2.

We have also developed a peer-to-peer application on top of the MOnKey middleware as a proof of concept. It allows users to share/send files between each other. The time necessary to transfer a file obviously depends on its size and on the communication technology[4].

The transfer resume process also works, whatever the communication technology, that is to say that a transfer can be-

[4]Let us recall some bandwidths: Ethernet is 100/1000 Mbps, WiFi 802.11g is 54 Mbps, Bluetooth v1.2 is 1 Mbps, v2.0+EDR is 2.1 Mbps, and v3, coming soon, would be 480 Mbps.

gin with a first communication technology, can be stopped (because the node can become unreachable, because the application is stopped or because the communication technology used is no more available) and then could be resumed using another communication technology in a transparent way.

When MOnKey is running on top of Bluetooth, it is affected by the limitations of its neighborhood discovery. Bluetooth technology raises several problems in term of neighborhood discovery. First of all, when a Bluetooth device is discovering its neighborhood, it is not discoverable itself. Furthermore, a neighborhood discovery lasts more than 10 seconds. Then, to identify services, a node has to ask each discovered neighbour, one by one, for the services it provides.

This is why neighbourhood (and services) discovery is inefficient in Bluetooth. Furthermore, to optimize the number of discovered neighbours, it is necessary to leave some time between each discovery request because if all the nodes wanted to discover their neighbours all the time, they would all keep on listening for beacons (not sending any), what would make the process ineffective. This is an obstacle to implement a responsive transfer resume functionality.

Let us summarize here the successive steps that are required to resume a transfer using Bluetooth. First, we wait a certain time (that depends on the number of present neighbours, and we have no idea about for the first discovery!). Thereafter, the optimum time interval between two successive inquiries depends on the number of neighbours and on the mobility of the nodes. Then, we perform a device discovery that requires at least 10 seconds [15]. To finish, we query each discovered Bluetooth device one by one in order to learn about the services that it provides.

We thus propose an optimization of the Bluetooth neighbourhood discovery. The problems related to the Bluetooth discovery process and its optimization have already been studied and discussed, for instance in [15, 16]. To deal with this problem, we propose to use an additional communication technology: ZigBee [6]. We choose ZigBee instead of WiFi because we think it is more convenient for power-limited devices such as PDA and mobile phone. Of course, if ZigBee technology is not present on the considered device, we apply the procedure described below using the WiFi technology.

ZigBee is a high level approach based on the IEEE 802.15.4 which is a standard that specifies its physical layer. It is a low-cost, low-power and very low-bandwidth[5] wireless technology. Several new hardware items have already been designed that integrate ZigBee, such as mobile phones and (U)SIM cards. These last will allow every mobile phone to use ZigBee, an important feature of which is to provide very fast neighbourhood discovery and we thus propose to use it in a bootstrap phase to exchange Bluetooth mac addresses. By doing so, we will bypass the Bluetooth discovery process and gain in reactivity.

This is currently only a proposal that still needs to be experimented.

[5] ZigBee bandwidth is 250 kbps.

5 Conclusion and future work

In this paper we have presented the rationale for a middleware built on a portable device that should deal with heterogeneous platforms and versatile networks. We have implemented a prototype in the form of a middleware on key called MOnKey. To illustrate the use of this prototype we have developed a peer-to-peer file transfer application on top of it. We have also discussed the problems raised by this approach that we have faced (and are still facing for some of them).

There are a number of directions/features that we want to investigate further.

One of the current missing features of the system is security. There is neither identity management, nore encryption of communications. Based on our experience in the domain of identity and security management in mobile ad hoc networks [17] we will provide some support for this aspect in the near future. This will be built in the lowest layers of our middleware so as to be integrated in the set of "universal" primitives[8] that can be provided on any platform (see section 2).

We would also like to go further in the notion of portable middleware by even suppressing the dedicated device that the user has to carry along. Of course there is no magic in that, there still remains a piece of hardware, but the idea is to use one that most people carry along all the time: their mobile phone. The concept is to put the middleware inside the memory card of the mobile phone and to have a nearby PC execute this code (today most phones are not powerful enough to execute large codes themselves, and for some codes, IOs are required that are not supported by mobile phones - screen, real keyboard, etc.-).

MOnKey would then become a seamlessly portable middleware.

References

[1] IEEE Standards Association. Ieee 802.11,2007 specification. Available at `http://standards.ieee.org/`.

[2] Bluetooth Consortium. *Bluetooth Core Specification v2.0*, november 2004.

[3] Framakey. Available at `http://www.framakey.org/`.

[4] Liberkey. Available at
http://www.liberkey.com/en/.

[5] Symbian os. Available at
http://www.symbian.com/.

[6] Zigbee alliance. Available at
http://www.zigbee.org/.

[7] Karlo Berket, Abdelilah Essiari, and Artur Muratas. Pki-based security for peer-to-peer information sharing. In *P2P '04: Proceedings of the Fourth International Conference on Peer-to-Peer Computing*, pages 45–52, Washington, DC, USA, 2004. IEEE Computer Society.

[8] Jérémie Albert and Serge Chaumette. Rationale for defining ncips (neighborhood and context interaction primitives). In *Ambient Intelligence Developments (AmI.d'07)*, pages 144–153, Sophia Antipolis, French Riviera, September 2007. Springer.

[9] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc On-Demand Distance Vector (AODV) Routing. RFC 3561 (Experimental), July 2003.

[10] T. Clausen and P. Jacquet. Optimized Link State Routing Protocol (OLSR). RFC 3626 (Experimental), October 2003.

[11] Project sun spot. Available at
http://www.sunspotworld.com/.

[12] Excelsior jet. Available at
http://www.excelsior-usa.com/.

[13] Luc Hogie, Pascal Bouvry, and Frederic Guinand. An overview of manets simulation. In *MTCoord - International Workshop on Methods and Tools for Coordinating Concurrent, Distributed and Mobile Systems*, pages 81–101, 2006.

[14] Luc Hogie. *Delay Tolerant Networks: Modelling, Simulation and Broadcast-based Applications*. PhD thesis, Le Havre University (France) and Luxembourg University (Luxembourg), april 2007.

[15] Ryan Woodings, Derek Joos, Trevor Clifton, and Charles D. Knutson. *Rapid Heterogeneous Connection Establishment: Accelerating Bluetooth Inquiry Using IrDA*. Brigham Young University Provo, Utah 84602.

[16] Igor Sedov, Stephan Preu, and Clemens Cap. Time and energy efficient service discovery in bluetooth. In *in Proceedings of the 57th IEEE Vehicular Technology Conference (VTC), Jeju, Korea*, 2003.

[17] Eve Atallah. *Une solution pour l'établissement non planifié de groupes sécurisés permettant des communications sûres dans les réseaux MANets*. PhD thesis.