INTELLIGENT AUTOMATION & SOFT COMPUTING, 2017
http://dx.doi.org/10.1080/10798587.2017.1280262

A Lightweight Approach to Access to Wireless Network without Operating System Support

Yonghua Xiong[a,b,d], Jinhua She[a,b,c] and Keyuan Jiang[d]

[a]School of Automation, China University of Geosciences, Wuhan, China; [b]Hubei key Laboratory of Advanced Control and Intelligent Automation for Complex Systems, Wuhan, China; [c]School of Engineering, Tokyo University of Technology, Hachioji, Tokyo, Japan; [d]Department of Computer Information Technology & Graphics, Purdue University Northwest, Hammond, USA

ABSTRACT

Wireless network is crucial for the Mobile Transparent Computing (MTC), in which a mobile device without any Operating System (OS) support needs to load the demanded OSes and applications through accessing the wireless network connection. In this paper, a lightweight approach based on the Boot Management System (BMS) was proposed to ensure the wireless network connection before booting OS. In BMS, the Virtual File System (VFS) technology was used to drive the wireless network card and establish a stable network connection. A prototype of the BMS was tested on ARM11 hardware platform and the results demonstrate the validity of the BMS.

KEY WORDS

BMS; Mobile transparent computing; Wireless network driver; Virtual file system

1. Introduction

As a new kind of computing method, transparent computing (TC) was proposed first in Zhang (2004) that provides users with resources and services, including operating systems and applications, from a server anywhere and anytime through any type of computing clients. In TC, all services and resources are stored in servers on a distributed network and when an application is called, an invisible computer may responds to a client's need by unobtrusively calling up a corresponding service from a fixed or mobile device nearby.

MTC is the development of TC with special concern on mobile equipment. While portable mobile devices are easy to carry around, their hardware resources and energy are limited (Lima, Leeb, & Jai-Hoon Kim, 2013). Taking these characteristics into consideration, all the resources including OSes and applications are stored and managed in remote servers and the clients are left to be almost bare hardware without any operating systems or applications being installed in advance. That is, all resources are accessed by clients through wireless networks in MTC. This not only lowers the requirement for the storage of a mobile device, but also improves the security of the system to some extent (Zhang, Wang, & Xu, 2013).

As all resources are accessed through wireless networks, it is essential for MTC to connect to servers before an operating system starts. The system based on virtual machines virtualizes network resources and disks, and have been widely used in PCs (Wang, Von Laszewski, Chen, Tao, & Kunze 2010). Generally, the system uses the virtual support at Intel VT (Intel Virtualization technology) hardware level and the XEN virtualization technology to run an OS for a client in a virtual machine. By means of calling up virtual disks and a network driver in a device model in the client space of the management domain, it assigns the requests of the access of client space and network I/O to servers over the network so as to run multiple operating systems remotely from one computer. On the other hand, full virtualization generates additional computational loads largely, and leads to the result that the overall system performance is much lower than the performance of the PCs in the system. To solve this problem, a TC system was proposed based on a lightweight virtual machine (Zhou, Zhang, Hao, et al., 2012). It only virtualizes network equipment, thus reduces the computational load.

Since the above virtual-machine-based methods only consider the situation with wired network connection, they are not suitable for mobile devices as mobile devices are connected to the network through wireless means, such as WIFI, GPRS and 3G. Addressing this problem, a meta-OS was developed for mobile devices (Zhang & Zhou, 2007). It rewrites a wireless network driving program in the recovery partition and uses Meta-OS to manage it. This technique implements online upgrade of OS and seamless migration of all configurations. However, in this method, the whole meta-OS is stored in the local storage, which requires large amounts of hardware resources. However, mobile devices are limited with hardware resources.

In this study, we developed a BMS using the VFS that not only solves the above problem, but also simplifies the system development by devising a uniform method to drive wireless equipment. The driving source codes for the wireless network devices in our approach only needs a little or even no change, so it is directly compiled to modules and stored in the VFS, ready for the dynamic calling of network device drivers. By this way, the wireless network equipment is able to connect to the server without OS for loading resource including OSes and application from the server, and thus realize remote boot. We do experiments in an ARM11 hardware platform, and the result shows the approach works well to achieve the goal.

The rest of the paper is organized as follows: Section 2 explains the concept of mobile transparent computing, and outlines the related work in this research field. Section 3 describes the design of BMS. Section 4 we present the implementation

CONTACT Yonghua Xiong xiongyh@cug.edu.cn

and verification of this method. Concluding remarks and future work are given in Section 5.

2. Background and Related Works

This section explains the concept of MTC, and outlines the related work in this research field.

2.1. Mobile Transparent Computing (MTC)

TC provides a new kind of computing model that is composed of servers and clients. But different from other computing models, TC allows a device to choose and run applications under multiple OSes. The OSes and applications are stored as resource in Web servers and the application computing is performed in the client. This reduces the server load and accelerates the response speed. TC separates the storage, operation, and management logically or physically. This ensures the computing to be a personalized service that is unconscious and controllable for clients.

MTC is an extension of TC by applying the technologies of TC to solve the problem in mobile computing by making use of the characteristics of mobile devices (Huang, Wu, & Xiong, 2014; Xiong, Huang, Wu, & She, 2014). The structure of the MTC is shown in Figure 1. MTC has the following characteristics:

- *Remote storage and local computing*

In MTC, mobile devices do not have any OSes and applications pre-installed. Only some boot managers are pre-installed in them. When a client submits required services through the interface provided by a mobile device, the pre-installed communication protocol and task manager in the mobile device automatically loads the corresponding resource from the transparent server, perform calculations locally, and saves the calculation results and data on the server.

- *Support of multiple OSes*

Clients can select any OSes (for example, Android, Linux QT), tools, and applications that are available to their mobile devices.

- *Dynamic loading of resources*

Clients can check the list of available resources on the network. The resources are loaded into a mobile device only when the client needs them.

In MTC, a mobile device only needs a small amount of space to store the management program. So, a mobile device can be made very small. This allows a small CPU to complete a variety of functions, and reduces the cost of hardware. A terminal only accesses different OSes in the servers and runs applications in its memory when necessary, it completes, in fact, a virtual computing (Schaffer, Averitt, Hoit, et al., 2009; Wannous & Nakano, 2010). Mobile devices are connected with servers through a wireless network, thus computing is ubiquitous.

2.2. Related Works

A considerable number of studies have been made on PC-based TC. OSes are dynamic loaded and run on the clients. So, the connection and the protocol of network are the key to performing TC. The MRBP (multi-OS remote booting protocol) protocol was proposed in Zhang, Xu, Wei, Yang, and Zhou (2008) and Schmelzer, von Suchodoletz, Schneider, et al. (2011) that supports multiple OSes for remote boot. It defines the operating environment of clients before the start of an OS, and a file download protocol-APTP (active program transfer protocol). While the MRBP protocol supports Linux and Windows 98 OSes, it does not support the remote boot of Windows 2000/XP. A remote boot protocol-MRBP2 (Yang & Zhang, 2006) was devised to solve this problem. It supports the dynamical loading and running of different OSes including Windows for a terminal from a server through Ethernet. According to this protocol, different OSes in the server are downloaded to and run in a client through the OSPM (OS pre-boot mechanism) mechanism and the SFTP (sector based file transfer protocol).

In addition, lots of effort has been made to develop new protocols that may support TC. iSCSI (Internet small computer system interface) (Baekjae, Sejin, & Woojoong, 2008) is a new protocol developed by the IETF (Internet engineering task force). It converts the data of the small computer system interface (SCSI) to network packets over IP networks. PXE (pre-boot execution environment) (Jinhui, Ke, & Fang, 2011; Tiago & Paulo, 2010) was developed by Intel. It works in a C/S network model. It allows a workstation to download images over the network from a remote server, and supports the boot of an OS on the workstation over the network. iSCSI and PXE are widely used protocols that implement TC.

The above methods and protocols for TC have been successfully applied to PCs. However, they are not suitable for mobile devices, for the distinctive characteristics when compared with PCs. That are, the storage space is quite limited, and the processing capacity is very low. And most importantly, the access that mobile devices connect to transparent server is wireless networks. So, the PC-based TC is hardly available to MTC. The primary problem we have to consider in MTC is how we can connect mobile devices to the network before the boot of OS.

TNOS (Zhang & Zhou, 2007) was proposed for MTC by focusing on the mobile feature. It reformed Android from the hardware layer to the application layer, constructed a Meta-OS in the recovery partition, and ported the drivers of wireless network cards into the Meta-OS. TNOS features the security and the integrity of the OS, because the Meta-OS can directly upgrade the OS in a mobile device in an online fashion and perform seamless migration of system configuration. However, this method is only applicable to upgrade Android. And the upgraded Android destroys the recovery partition, and makes the Meta-OS not available anymore. The Pre-OS was presented in Huang et al. (2014) to solve this problem. The Pre-OS runs in the BootLoader layer, and ports the drivers of wireless network devices in it so as to manage them. It initializes hardware devices and connects wireless network devices to the networks without OS environment, and then loads the resources for the boot of required OS. Although this method accomplishes MTC, the implementation is very complicated.

Specially, when compared with TNOS and Pre-OS, BMS has obvious advantages. For TNOS, the upgraded Android will destroy the recovery partition, and make the Meta-OS unavailable. But, in BMS, there is no such problem, the OS will be upgraded without destroying the recovery partition and the BMS can be used again. In Pre-OS, the versatility of the method is poor and the programs in a mobile device have to be ported after the replacement of network equipment. While in BMS, different kinds of WNC drivers can be compiled to drive modules and added to the VFS, and the driver will be loaded dynamic if needed.

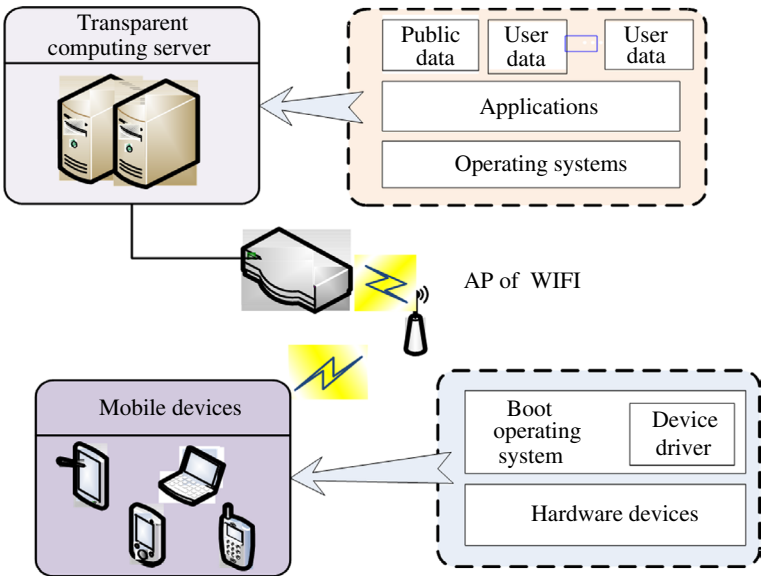

Figure 1. Structure of MTC.

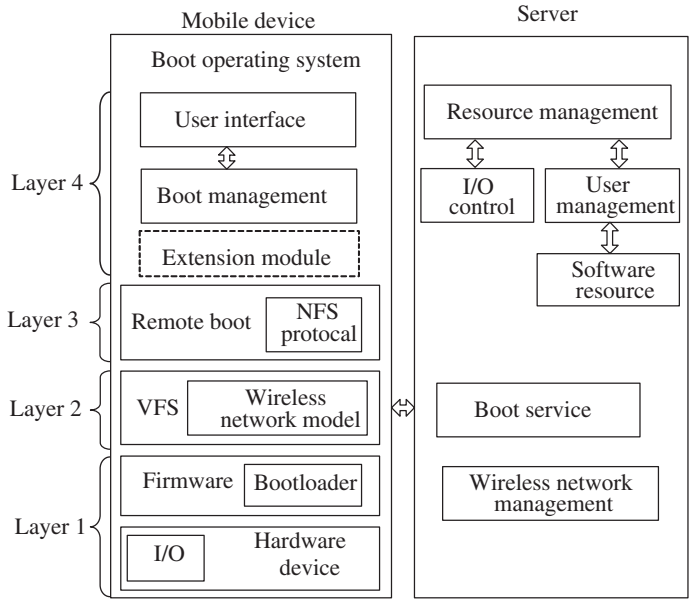

Figure 2. Diagram of Structure of BMS System.

When it comes to the application of VFS technology, there have existed several related researches. A mobile distributed short file sharing system was created by Nikolaos and Dimitris (2004). It uses the NFS (Network File System) protocol that is a method of sharing documents on the Internet to share and transfer files like on the client's local hard disk. An information management architecture named as NLI-enabled (natural language interface) was designed by Zhoua (2007) to lead to improve efficiency and effectiveness of managing information on mobile devices. A flexible intermediate library named Stampi was proposed by Tsujita (2007) to realize seamless remote MPI-I/O (Mechanism Provided Parallel-Input/Output) operations on interconnected computers. And a parallel virtual file system (PVFS) was supported in the remote MPI-I/O mechanism for data-intensive applications. The above applications have not considered the characteristics of MTC; therefore it is difficult to extend them into MTC.

3. Design of BMS

The BMS, with the C/S architecture, was designed to provide mobile devices with transparent computing services. In which, the server are to provide mobile clients with a variety of resources and services over the wireless network. And a mobile device does not need to install any OSes or applications in advance. All the resources that a client required are loaded to it through the module of wireless network driver that provided by the BMS remotely.

3.1. Framework of BMS

In order to drive the wireless network card of the mobile device and load the needed OS and applications as required, the BMS was designed in a hierarchical mechanism. The structural diagram of the system is shown in Figure 2. It is divided into four layers; the hardware layer, the network layer, the remote start layer, and the management layer.

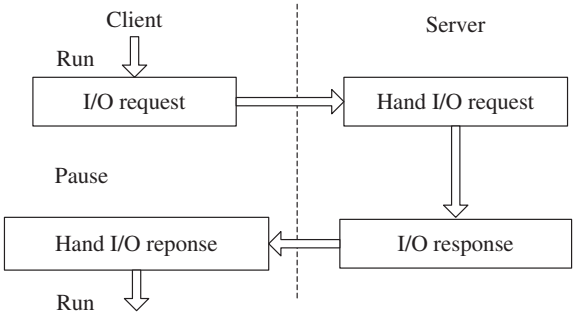

Figure 3. The Remote Request Process.

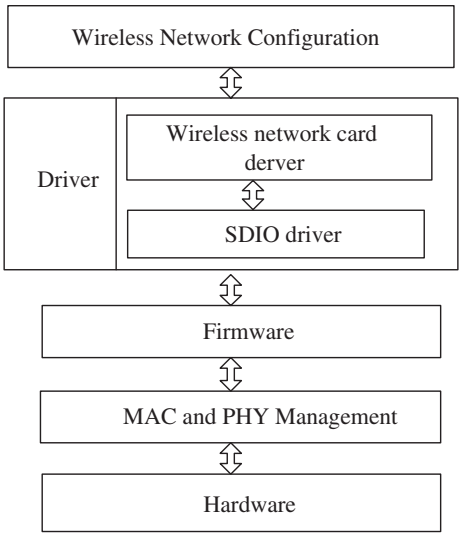

Figure 4. Wireless Network Card Driver Structures.

First, after mobile devices powers up, the u-boot is one kind of Bootloader in Layer 1 that runs and initializes the hardware. Then the management system starts to runs and displays the list of the available OSes stored in server and prompts user to choose through the user interaction interface where different keys correspond to different OSes in Layer 4. Once user chooses one OS, BMS will load the kernel and VFS images of the OS into RAM for running. The VFS is the main part of Layer 2 and it will start to run after the kernel boots for achieving configuring the wireless network card, searching server and connecting to the server. After the completion of all these preparations, the VFS hands over the right of system control to NFS in Layer 3. Obtaining the permission of system control, the main process of the NFS will run to find a server on the network and load concerned resources that are necessary for completing the booting of the OS.

- *Hardware layer (Layer 1)*

The hardware layer provides the foundations of the network connection and program execution for a client and the server. After the device is turned on, a curing program initializes all hardware, for example, setting the frequency of the system clock, the address of memory, etc., to prepare the runtime environment of programs.

- *Network layer (Layer 2)*

The virtual file system was used in the network layer to supervise and control the wireless module. It is responsible for scanning the available wireless network, configuring the parameters of a wireless network and connecting to the transparent server, etc. The network layer is a bridge between the client and the server. It is mainly responsible for finding an appropriate server and connecting to it after the mobile device is launched. It provides connection service for the remote start layer when loading an OS in a real time fashion.

- *Remote start layer (Layer 3)*

The remote start layer is mainly responsible for the remote loading of the resources, OSes and applications. After a client connects to the network, the remote start module automatically finds a server on the network and loads resources. The BMS catches the I/O request packages it into NFS formatted packets, and then sends it to the server. When the server receives a message, it extracts the request, answers it, and then compiles the result into a response message and sends it back to the client. The client recompiles the message and finds the result. The process is shown in Figure 3.

- *Management layer (Layer 4)*

The management layer is primarily responsible for the client interaction. According to the client's selection, it carries out necessary initialization, and then forwards the client's selected information to the remote start layer. The remote start layer loads the resources of OSes and services according to the information. In addition, the management layer contains an extensible module for the use of dynamically adding functions required by clients and to reduce the influence of the adding operation on other modules.

At the beginning, all resources of OSes, applications, data, etc. are stored in the server and shared by all clients. A client can choose any necessary applications to install. As time went by, the data (client programs, data and applications) that produced by different clients with different OSes, will be updated to the server as private data, which can only be changed by the same user authentication. The server allocates storage space and permission for each client. A client is free to access resources within the permission.

3.2. Wireless Network Driver

A wireless network driver plays an important role in MTC. It is a key to determining whether or not a client can connect to the server. The result of porting the driver to the BMS directly determines the rate and accuracy of packet transmission. In this study, a Marvell 8,686 wireless card is used. The Marvell 88w8686 is a low-cost, low-power highly-integrated IEEE 802.11a/b/d MAC/Baseband, designed to support IEEE 802.11a or 802.11 g payload data rates 6, 9, 12, 18, 24, 36, 48, and 54 Mbps, as well as 802.11b data rates of 1, 2, 5.5, and 11Mbps. Its structure is shown in Figure 4. The MAC (media access control) and PHY (physical layer) are implemented by the hardware and software of a wireless network card (WNC), respectively. A MAC protocol processing chip was designed based on IEEE 802.11 standard. It processes data received from a radio frequency module to a required format, converts packets received from the driver to 802.11 standard packets, and sends them to the front end of the radio frequency module. The structure is divided into a WNC, a driver of the card, firmware, and the management of the WNC configuration. The firmware is the basic control system of the WNC. It implements the control and management of the network card based on the MAC chip. The firmware completed the bottom, the most complex transport/delivery module function, provided a

physical interface for above layer, and offered a programming interface for next layer.

When an SDIO (secure digital input and output) interface of a WNC is connected to the system, the CPU first sees the SDIO bus, and then the network card chip. An SDIO driver is a bridge between a client and the WNC driver. The program of the WNC driver is basically the official source. We made some necessary modifications so as to build it as a module in the BMS. This allows us to add different kinds of driver modules for network cards in the BMS, and to find the corresponding driver number if necessary. This ensures the one-to-one relationship between a card and a driver. A program of a WNC driver mainly contains the initialization of the card, and sending and receiving of packets.

The initialization of a network card is mainly to check the existence of a network card, fill device structure, and register the card:

- Detect the existence of a network card based on the characteristics of its hardware, and call the driver program.
- Detect the I/O address and interrupt of the card, and call request_irq and request_region to register the base address of I/O and the interrupt of the card.
- A hardware device adds its own hardware frame in front of the header of packets, and then sends the data. The driver programs use the hard_header method. The length of the hardware frame head is filled in dev->hard_header_len. The protocol layer calls hard_header before sends a packet, and reserves a storage space for the hardware frame at the beginning of packets. The hard_header method calls skb_push and then fills in the hardware frame.
- Use the wifi_setup method to set the equipment device structure members for wireless network equipment.

Packets are sent using the following procedure. According to the OSI seven-layer protocol, when data are sent from the application layer, packets are delivered from the top layer to the bottom one based on the network protocol, and a frame head is added to the packet at each layer. Finally, packets are delivered to the network interface using the dev_quene_xmit method. The network interface passes packets to the hardware device of the network cart that carries out the physical transmission of packets.

Packets are received as follows: When a network card receives a packet, it sends a hardware interrupt request to the BMS. The BMS uses a driver to receive packets. To handle an interrupt, first, the BMS determines the hardware types of the WNC, and the length of a packet frame head based on the frame control bits. Next, it prepares a buffer, sk_buff and saves the packets read from the hardware device in it. Then, it fills in the sk_buff: skb - > dev = dev, judges the type of the frame protocol and fills in skb: skb - > protocol = hotns (). After it sets the pointer skb - > mac.raw to the data in the hardware device, it discards the hardware frame head. Finally, the BMS calls the netif_rx method to send the data to the protocol layer for processing.

3.3. Virtual File System

The virtual file system works on the network layer. It was introduced in BMS to simplify the implementation. As it is really difficult to port the WNC drivers, includes the SDIO interface driver, the network card driver, the stack of the network protocol, and the management procedure, etc., to the environment without an operating system. And even though all these needed drivers were successfully ported, all the tasks have to be done again if a network device or a mobile device is replaced. This is a waste of efforts and not practical. To solve this problem, we introduce the concept of the virtual file system. A Linux file system is divided into two layers, the upper layer is the virtual file system layer, and the lower is the real file system (RFS) layer (for example, NFS, ext2, ext3, etc.). Other functional layers and the file system send communication requests to the VFS layer. The VFS unifies the calling interface for the system accessing files. When applications or other functional layers call the interface of the VFS, the function of the VFS interface calls the function of the RFS, which is the function to really read or write files.

The VFS links available file systems to a single tree, which displays all file systems as an entity. Any type of a file system is assembled into a proper directory in the tree structure. The system can only see the latest file in the entity.

The VFS slightly modifies the wireless network driver according to the hardware features, and then compiles it as a driver module but not a program in the kernel. When the BMS starts up, it detects the hardware device of the network card and searches for the corresponding driver. Then, it checks the VFS. If it finds the corresponding driver in the VFS, then it loads the driver into the memory dynamically and connects the WNC to the network. It is able to compile a variety of commonly used WNC drivers to drive modules and add them to the VFS using the BMS to dynamic load hardware drivers from the VFS. This makes it possible to support a variety of WNCs. Since drivers are dynamically loaded, unused network card drivers do not consume the resources of the limited memory. So, it is convenient to use the VFS to execute scripts, manage the wireless network, and complete the connection to the server.

3.4. Management of Wireless Network

It is essential for the MTC that the wireless network is connected stable. The management of the wireless network has two phases: Search for the network when a client starts, and the access to the RFS in the server.

When the BMS at the client side starts, it searches a server on the network based on the client's requirement. The BMS first wakes up a wireless network equipment device, which is inactive in default. Then, it uses wireless tools to search hot spots of the network. If it detects a target server, then it configures a wireless network account and a password and connects to the server. A failure at any places in the process of network connection results in the break of the network connection. This forces a restart of the mobile device and a repeat of the connection process. Since this is not convenient to clients, we built a function of detecting the state of network execution in the wireless network management module. Note that every step in the network connection returns a value (0: true, 1: false). We use the function to detect the execution result at each step. If an error occurs at a step, then the function repeats the step itself until success. It allows us to just repeat the failure step but not the whole access process. So, it saves the startup time and improves clients' satisfaction.

The BMS loads the sources for the boot of an OS from a remote server. Then, it passes the authorization of system administration to the RFS to complete the boot process. The BMS life cycle usually ends at this point. Note that the

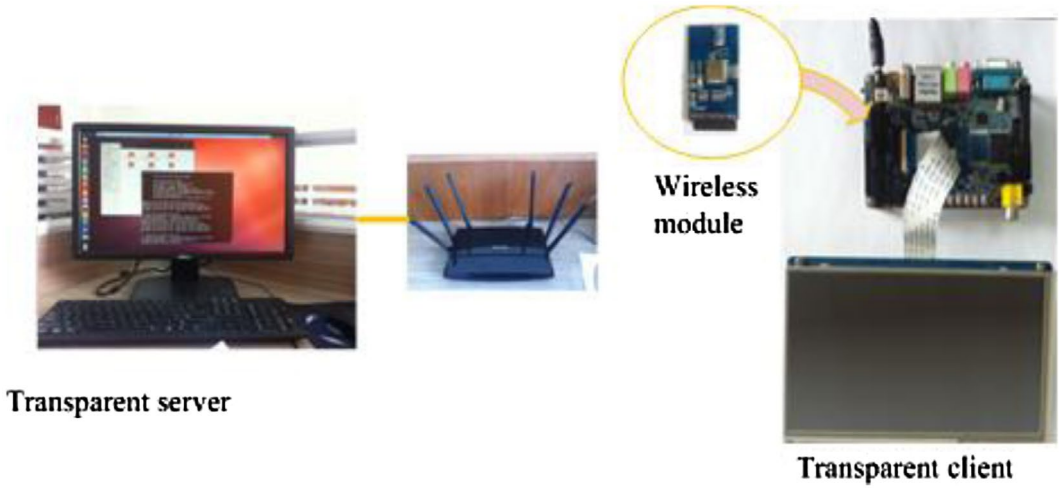

Figure 5. Diagram of Construction of the Testbed.

Figure 6. Test of Remote Boot of OSes: (a) List of Available OSes and (b) Boot Result of Linux OS.

Table 1. Comparison of Start-up Time.

	Local booting	Remote booting
Linux QT	26.50 s	72.17 s
Android 2.3	68.00 s	175.07 s

configuration information for the wireless connection is saved in the BMS. The ending of the BMS life cycle results in the break of the wireless network connection. To solve this problem, we do not quit the BMS after it passes the authorization of system administration to a RFS. Instead, we keep it as a directory of the RFS. This ensures that the process of authorization transmission does not break the network connection and that the OS starts up smoothly.

4. Implementation and Verification

In this section, a prototype of the BMS was built based on OK6410-B ARM11 hardware platform. In order to evaluate the proposed BMS, we conducted two sets of experiments: The remote booting of multiple operating systems and the performance testing of the wireless network connection.

The construction of testbed is shown in Figure 5. The server for TC is a PC (CPU: Intel I5-3470, 3.20 GHz; memory: 1 GB; OS: Ubuntu 12.04). It provides the services of the TFTP (trivial file transfer protocol), the PORTMAP and the NFS. The clients are mobile devices that based on OK6410-B boards (CPU: S3C6410, 533 MHz; DDR2 memory: 256 MB; NAND flash memory: 2 GB; wireless module: Marvell@ 88W8686). The OK6410-B is a highly-integrated board that designed for the use in mobile devices, such as a mobile phone, a PDA, a video camera, etc.

The wireless module Marvell 88W8686 integrates a radio frequency wireless transceiver that operates in both, 2.4 GHz and 5 GHz, a physical layer, a media access controller, and an ARM processor. The wireless network module supports standard industrial SDIO interfaces and uses IEEE 802.11 b network standard. Data are transferred at the rate of 54 Mbps with WEP encryption.

A mobile device without any OSes and applications installed in advance, that only is equipped with the BMS to launch a management program. The OSes of Linux QT and Android 2.3, as well as clients' data are stored in the server that is responsible for the management and maintenance of data. All clients' required data are loaded from the server through wireless networks. Computational results are stored in the server.

We tested the performance of the BMS at the side of a single client and at that of the server. First, we ran multiple OSes (Linux and Android) in a client's mobile device, and then ran applications to verify the validity of the BMS. Besides, we also tested the performance of the wireless network during the boot of the OSes and the execution of applications, and analyzed the experimental results.

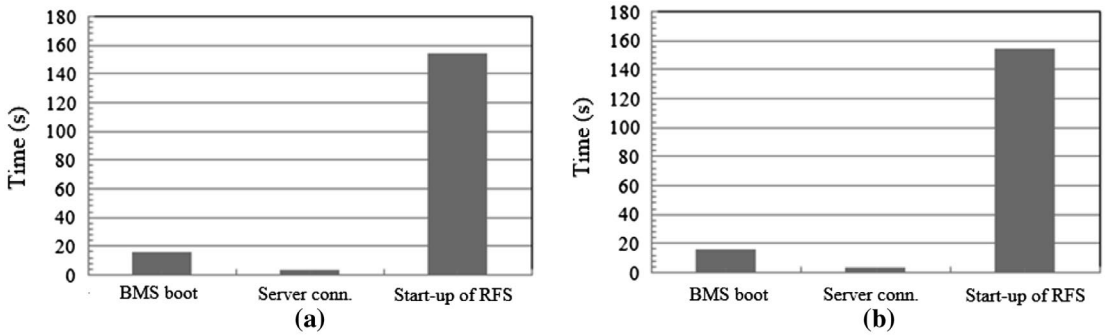

Figure 7. Distribution of Start-up Time at Different Stages: (a) Linux QT (b) Android 2.3.

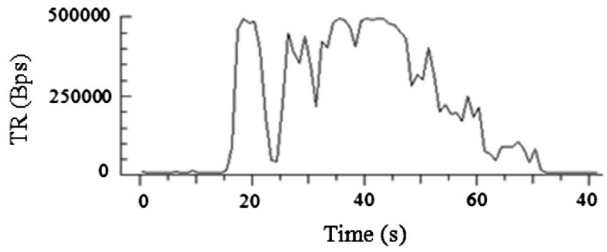

Figure 8. TR of Wireless Network During Start-up Process of Linux QT.

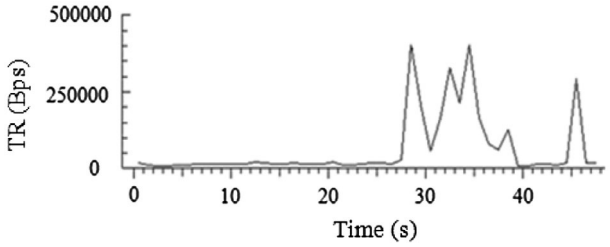

Figure 9. TR of Wireless Network During Start-up Process of Android 2.3.

4.1. Test of Remote Boot of OSes

We first demonstrate the effectiveness of the BMS for MTC. We load the BMS in the flash with a fixed address in a client's mobile device. Considering that both of Android 2.3 and Linux QT are not only open-source systems but also used popularly, we chose them as the example of experimental OSes so that the proposed method is of more scalability and commonality. When we turned the device on, the BMS initialized the hardware devices, displayed a list of the available OSes (Figure 6(a)), and waited for the client choosing one to start up. The boot result of Linux QT is shown in Figure 6(b) .

After the start-up of the device, we used the device to run some applications through MTC, such as Microsoft Office, multimedia data, and network applications. We felt that the applications were run just like they were in the client's local memory. All data was easy to reload when the device restarted or the client replaced the device with a new one. Since the client does not need to care about the location of the storage of the OS and applications, and data that produced in the computing are stored in the server; it is safe and easy to manage.

We also tested Android 2.3 OS and the same applications, the test results are as good as those of the Linux.

The start-up time of an OS is defined to be the time period from the switch-on of the device to the display of the desktop. The test results of the start-up time for the remote and local boot are shown in Table 1. It is clear from the table that the start-up time of remote boot is much longer than that of local boot.

We then analyzed the start-up process in detail. The process is divided into three stages: BMS boot, connection to the server, and start-up of the RFS. The time distribution of the start-up time for the two OSes is shown in Figure 7. Clearly, the start-up of the RFS spends most of the start-up time. In this stage, the BMS searches for file systems, loads required resources to the local device, and then pass the authorization of system administration to the RFS. Note that the Linux file system is only about 70 MB. The time is mainly spent to load the required resources over the wireless network. So, it is possible to shorten the start-up time by optimizing the file systems and reducing the required resources for the start-up. In fact, the test results show that the start-up time of the remote boot is in the acceptable range. Repeating the test showed that the start-up time is quite stable.

The above test results show that the BMS is effective to drive a wireless network and to implement MTC.

4.2. Test of Performance of Wireless Network Connection

Resources are different for an OS at different stages in the start-up process. This makes the transmission rate (TR) of the network changes at different stages. We measured the TR of the wireless network during the start-up process. The results for Linux QT and Android 2.3 are shown in Figures 8 and 9.

As shown in Figure 8, at the beginning (0–31 s), the BMS initializes the hardware device, configures the wireless network, and searches for available transparent server. It gives little effect on the TR of the network, because it mainly deals with the local device in this period. The client loaded resources from the server from 32 s. The maximum TR was 4 Mbps. But that TR was only used for a very limited period. It is clear that it is possible to shorten the start-up time by increasing the TR.

Our system used a TL-WDR4900 router that has the maximum TR of 450 Mbps. The transmission performance was good in an LAN. The use of WEP encryption in the experiments reduced the TR of the wireless connection. We found that, if we connected an 802.11n client to an 802.11n router, the TR was less than 54 Mbps even signals were very strong. This is because that the highest TR for WEP (Wired Equivalent Privacy) or WPA (Wi-Fi Protected Access) is 54 Mbps according to IEEE 802.11n standard. To improve the TR, we need to use WPA2 for the router or not use any encryption.

Figure 9 shows the measured result of the TR for playing music under Linux QT. The TR was 3.2 Mbps when the music data was

downloaded from the network, but it is less than the maximum TR. Experimental results show that it had little effect on the use of this kind of application even we increased the TR of the network.

5. Conclusion

This paper proposed a lightweight approach based on a boot management system to access to wireless network without OS support, which is a small system for the MTC to port wireless network driver and manage the wireless network connection. This system only needs a very limited modification of the original source code of the drivers. It has some extra features superior to others:

- It loads drivers dynamic if needed. Since there are several kinds of WNC drivers compiled to drive modules and added to the VFS, when the BMS starts up, it only loads drivers needed into the memory dynamically from the VFS.
- A new device is easily to be added to the system by compiling it to a module. The VFS encapsulates the calling interface as a layer between the Linux kernel and devices. As a result, it is easy to add a new device to the system by slightly modifying the device driver and compiling it as a driver module rather than a program of the kernel.
- It drives wireless devices and connects them to a server automatically without an OS. Before the OS booting, the BMS detects the hardware device of the network card, loads the corresponding driver that can be found in the VFS into the memory dynamically and achieves connecting the WNC to the network.
- Considering the features mentioned above, loading data in MTC is easily implemented by the system.

On the other hand, the TR of the wireless network needs to be improved. We also plan to examine the main factor that influences the TR of wireless networks, to shorten the start-up time, and to improve the user satisfaction level in our future work.

Acknowledgment

This research was supported in part by the National Nature Science Foundation of China under Grant No. 61202340, by the International Postdoctoral Exchange Fellowship Program under Grant No. 20140011, by the Fundamental Research Funds for the Central Universities, China University of Geosciences (Wuhan), by the Hubei Provincial Natural Science Foundation of China under Grant No. 2015CFA010, and by the 111 project under Grant B17040.

Disclosure statement

No potential conflict of interest was reported by the authors.

Notes on contributors

Yonghua Xiong received his M.S. and Ph.D. degrees in engineering from Central South University, Changsha, China in 2004 and 2009. He was a lecturer and then an associate professor of the School of Information Science and Engineering, Central South University from January 2005 to August 2014. He joined the staff of the China University of Geosciences in September 2014, where he is currently a professor of the School of Automation. He has been in the Department of Computer Information Technology & Graphics, Purdue University Northwest, USA, as a post-doctor since January 2015. His research interests include computational intelligence, scheduling algorithms, and cloud computing.

Jinhua She received his B.S. degree in engineering from Central South University, Changsha, China in 1983, and his M.S. and Ph.D. degrees in engineering from the Tokyo Institute of Technology, Tokyo, Japan in 1990 and 1993, respectively. In 1993, he joined the School of Engineering, Tokyo University of Technology, Tokyo, where he is currently a professor. His research interests include the application of control theory, repetitive control, process control, Internet-based engineering education, and robotics.

Keyuan Jiang received his B.S. degree in computer science from Southeast University, Nanjing, China in 1982, his M.S. in biomedical engineering from Shanghai JiaoTong University, Shanghai, China in 1985, and his Ph.D. degrees in biomedical engineering from Vanderbilt University, Nashville, USA, respectively. He joined the Department of Computer Information Technology & Graphics of Purdue University Northwest, where he is currently a professor and the Department Head. His research expands from bioinformatics, mining healthcare social media, and clinical research and healthcare information technology.

References

Baekjae S., Sejin P., & Woojoong L. (2008). Enhancing robustness of an iSCSI-based File system in wireless networks. In *Proc. of the IEEE conference on Computer Systems Architecture* (pp. 1–7). Hsinchu.

Huang S.Z., Wu M., & Xiong Y.H. (2014). Mobile transparent computing to enable ubiquitous operating systems and applications. *Journal of Advanced Computational Intelligence and Intelligent Informatics, 18*, 2014, 32–40.

Jinhui L., Ke Z., & Fang Z. (2011). Network center's highly-efficient management solutions based on intel PXE-based remote cloning system. In *Proc. of the IEEE International Conference on Advanced Computer Control (ICACC)* (pp. 408–411). Harbin.

Lima S.H., Leeb B.H., & Jai-Hoon Kim J.H. (2013). Voluntary disconnected operations for energy efficient mobile devices in pervasive computing environments. *Intelligent Automation & Soft Computing, 19*, 39–49.

Nikolaos M., & Dimitris K. (2004). Designing an NFS-based mobile distributed filesystem for ephemeral sharing in proximity networks. In *Proc. of 2004 4th Workshop on Applications and Services in Wireless Networks* (pp. 225–231). Boston.

Schaffer H.E., Averitt S.F., & Hoit M.I., Peeler, A., Sills, E.D., and Vouk, M.A. (2009). NCSU's virtual computing lab: A cloud computing solution. *Computer, 42*, 2009, 94–97.

Schmelzer, S., von Suchodoletz, D., & Schneider, G., Weingaertner, D., de Bona, L.C.E., and Carvalho, C. (2011). Universal remote boot and administration service. In *Proc. of the IEEE conference on Network Operations and Management Symposium (LANOMS)* (pp. 1–6). Latin American.

Tiago C., & Paulo S. (2010). Integration of PXE based desktop solutions into broadband access networks. In *Proc. of 2010 IEEE International Conference on Network and Service Management (CNSM)* (pp. 182–189). Niagara Falls.

Tsujita Y. (2007). Remote MPI-I/O on a parallel virtual file system using a circular buffer for high throughput. *Journal of International Journal of Computers and Applications, 29*, 2007, 291–299.

Wang L., Von Laszewski G., Chen D., Tao J., & Kunze M. (2010). Provide virtual machine information for grid computing. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans, 40*, 1362–1374.

Wannous, M., & Nakano, H. (2010). NVLab, a networking virtual web-based laboratory that implements virtualization and virtual network computing technologies. *IEEE Transactions on Learning Technologies, 3*, 129–138.

Xiong Y.H., Huang S.Z., Wu M., & She J.H. (2014). A Novel resource management method of providing operating system as a service for mobile transparent computing. *The Scientific World Journal,* 4 2014, 1–12.

Yang, H.J., & Zhang, Y.X. (2006). A remote transparent computing boot protocol MRBP2. *Mini-Micro Systems, 27*, 1657–1660.

Zhang, Y.X. (2004). Transparent computing: From concept to implementation. *Acta Electronica Sinica, 32*, 169–174.

Zhang Y.X., & Zhou Y.Z. (2007). 4VP: A novel meta OS approach for streaming programs in ubiquitous computing. In *Proc. of the IEEE International Conference on Advanced Information Networking and Applications* (pp. 394–403). Niagara Falls.

Zhang, X.J., Wang, Z.J., & Xu, F. (2013). Reliability evaluation of cloud computing systems using hybrid methods. *Intelligent Automation & Soft Computing, 19*, 165–174.

Zhang Y., Xu G., Wei L., Yang H., & Zhou Y. (2008). Method and computing system for transparence computing on the computer network. *U.S. Patent 7,467,293[P]*. 2008, 12–16

Zhou Y.Z., Zhang Y.X., & Hao L., Xiong, N., and Vasilakos, A.V. (2012). A bare-metal and asymmetric partitioning approach to client virtualization. *IEEE Transactions on Services Computing, 7*, 2012, 40–53.

Zhoua, L. (2007). Natural language interface for information management on mobile devices. *Journal of Behaviour & Information Technology, 26*, 197–207.