

Utilizing Remote Storage through USB over IP on SoD System Environment

Kyuchang Kang*, Jinho Yoo**, and Jeunwoo Lee*

**Green Computing Research Department, IT Convergence Tech. Research Lab., ETRI*

161 Gajeong-dong, Yuseong-gu, Daejeon, 305-350, Korea

***Division of Information and Communication, Baekseok University*

115 Anseo-dong, Cheonan Chungnam, Korea

**Tel +82-42-860-1695, 5037, {k2kang, ljwoo}@etri.re.kr*

***Tel +82-41-550-0493, yoojh@bu.ac.kr*

Abstract— This paper addresses method for utilizing a remote storage through USB over IP on the SoD system environment. Toward this goal, a file system expected to be used as a remote storage is generated by a file-backed storage gadget, and it is connected to a USB/IP gadget. Then, the file system is viewed as a USB mass storage device in a virtual machine of host computer. The goal of this paper is to provide a private storage device on demand with a virtual machine running on the SoD system environment.

Keywords— USB emulation, USB over IP, a file storage, a remote storage, a file storage gadget

I. INTRODUCTION

New trend in a personal computing environment is changed from PC (personal computer)-based computing to device-collaborated ubiquitous computing. To support this trend, we need to provide individual computing environment enabling anytime, anywhere computing service by means of collaborating distributed ubiquitous devices.

One of approaches implementing this computing environment is a system on-demand (SoD) system. The SoD system is implemented by following procedures: (1) recognizes and manages functional resources from distributed computing devices as virtual functional resources, (2) according to the user's needs, provides virtual machines by combining virtual functional resources. From technology perspectives, we need virtualization technologies translating and enabling physical resources into virtual resources in implementing the SoD system.

In the result, an increasing ubiquitous computing infrastructure promises mobile users an attractive carry-nothing computing model, in which powerful computers for personal use can be found anywhere, anytime.

This paper addresses one of technical approaches, especially a remote storage service, for implementing the SoD system.

The purpose of this paper enables personal terminals such as a PDA or an Android phone to be used as a general USB storage device whenever a network connection is established.

In this paper, we present the initial basic mechanism of a remote storage service and its implementation details. We describe how to implement a remote storage service: (1) translating a general file system into a USB storage device, (2) connecting a USB storage device through IP network between a host computing device and a peripheral terminal, and (3) utilizing a connected USB storage device in virtual machine.

A rest of the paper is organized as follows. In section II, we explain background technology. The section III presents implementation of a remote storage service based on USB over IP and a file-backed storage gadget (FSG). The section IV describes test results and some discussing issues. Finally, we summarize and address future works in section V.

II. BACKGROUND TECHNOLOGY

This section describes the background technology about USB, USB over IP, and FSG enabling a remote storage service on the SoD system environment.

A. SoD (System on-Demand) System

We define the SoD system as virtual computing environment providing device-collaborative virtual machines by integrating peripherals distributed nearby users, as shown in Figure 1.



Figure 1. The SoD system environment

From the perspective of physical entities, the SoD system consists of a SoD station and variable peripherals. The SoD station makes and manages a SoD zone and provides following functionalities:

- Management of computing resources
- Recognition of user and peripheral devices
- Dynamic allocation of resources
- Provision of collaborated virtual machine environment

All computing resources and peripheral devices in SoD zone are registered into a resource manager of SoD station. The basic computing environment such as operating system except peripherals is configured by hypervisor [1-5] such as Xen [6].

B. USB and USB over IP

USB (Universal Serial Bus) [7] is a way of setting up communication between a computer and peripheral devices. USB physical device is a piece of hardware on the end of a USB cable that performs some useful end user function. USB client is software that executes on the host, corresponding to a USB device. This client software is typically supplied with the operating system or provided along with the USB device. USB system software is software that supports the USB in a particular operating system. The USB system software is typically supplied with the operating system, independently of particular USB devices or client software. As a host side bus interface, USB host controller is the hardware and software that allows USB devices to be attached to a host, as shown in Figure 2.

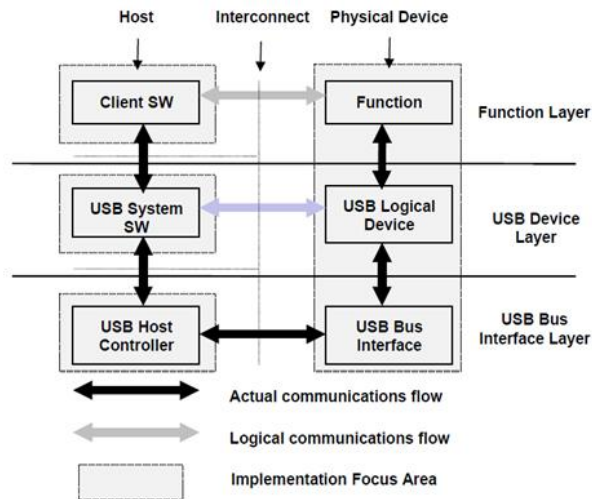


Figure 2. USB software implementation

On the other hand, USB over IP software [8] allows us to share USB devices and access them remotely over fast network such as gigabit ethernet. Using this software, USB devices connected to remote computers in local network or internet can be accessed as if they are locally plugged in. USB over IP is a kind of device emulation which consists of software modules. Actually there is no physical device on

server side. There is just the software modules connected to USB device on client side over internet protocol. The role of this device emulation module makes two IP channels for transferring USB four transfer modes. And this module has management channel over internet protocol for transferring USB data. This management channel module manages data flows of USB data over internet protocol. Thus this device module implements a service access point between one USB device and other USB device driver over internet protocol. So applications which work with the device can then be run without the device being locally present.

In detail, the network connection between server side and client side consists of virtual connection and data connection over TCP/IP with nagle disabled. The virtual connection is for doing services of listing, requiring information and sending command of attaching, detaching and so on. This virtual connection also communicate virtual USB module. And the USB module on server side consists of packet receiver, packet transmitter and exception handler of this module. Packet receiver and transmitter is related to transferring packets. Packet transmitter and receiver are kinds of a stub role. if error occurs of this module, all errors is processed by their exception handler. This USB module is stacked on USB core drivers. USB core drivers control USB host controller driver which is on USB host controller. USB module on client side consists of packet receiver, transmitter and exception handler like server side. By the way the USB core driver is on this client module in client side. The USB device drivers is stacked on USB core driver. USB device drivers communicate with device control manager of userland for virtual connection management.

C. File-backed Storage Gadget

The file-backed storage gadget (FSG) [9] provides support for the USB mass storage class. It can appear to a host as a set of up to 8 SCSI disk drives called logical LUNs, although most of the time a single LUN is all we will need. The information stored for each LUN must be maintained by the gadget somewhere, either in a normal file or in a block device such as a disk partition or even a ramdisk. This file or block device is called the backing storage for the gadget, and we can tell FSG where the backing storage is when we load the gadget driver. Figure 3. shows the connection by FSG between computing devices.

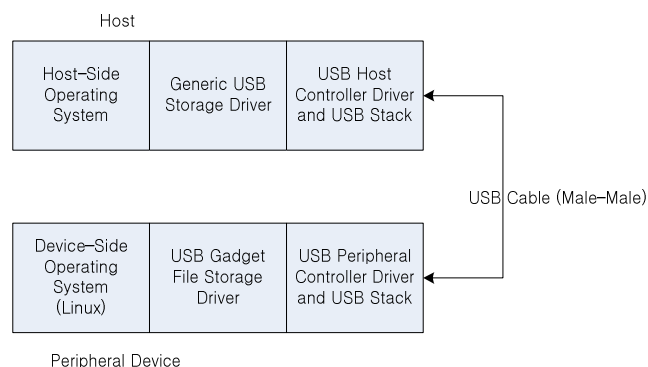


Figure 3. FSG and its connection

III. REMOTE STORAGE SERVICE

This section describes the implementation of a remote storage service based on Linux.

A. Overview and Scenario

The service scenario using a remote storage service, as shown in Figure 4, is composed of two tasks. Firstly, we can recognize a smart phone as a portable storage disk on the virtual machine through translating non-USB device to USB device. Secondly, we can exchange files existed in each smart phones.



Figure 4. Service scenario

Figure 5. shows corresponding software entities and relationship to provide a remote storage service. The remote storage service consists of three procedures as following.

- Procedure 1: creating and configuring a file system
- Procedure 2: configuring USB over IP connection
- Procedure 3: connecting USB device to a virtual machine

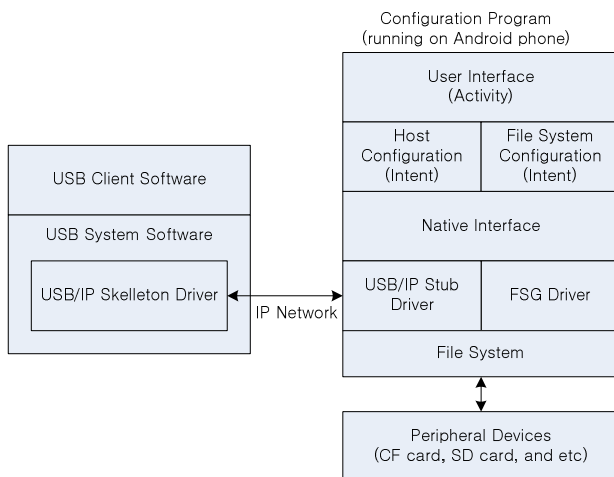


Figure 5. Overview of remote storage

B. Creating and Configuring a File System

To translate a file system to USB storage, we use file-backed storage gadget (FSG). The translating procedure consists of four steps.

Firstly, we need to create a backing storage file. Backing storage requires some preparation before FSG can use it. To start with, if the backing storage is a regular file then the file must be created beforehand, with its full desired size. Because FSG won't create a backing storage file and won't change the size of an existing file. As an example, we can create `'/root/data/backing_file'` to represent a 64MB drive using a command something like this:

```
# dd bs=1M count=64 if=/dev/zero \
of=/root/data/backing_file
```

This has to be done before we can load `g_file_storage`, but it only has to be done once. If the backing storage is a block device or disk partition such as `'/dev/hda7'` then we do not have to create it beforehand, because it will already exist.

Secondly, we need to partition the backing storage. Sometimes, creating the backing storage is not enough. It is like having a raw disk drive; we still need to partition the disk and install a file system before we can use it. To partition the backing storage, we create a partition table by using the `fdisk` program.

Thirdly, we need to make a file system. After `fdisk` operation, a new partition has been created but it does not yet contain a file system. The easiest way to add a file system is to load `g_file_storage`, connect the gadget to a USB host, and use the host to do the work. With a Linux host we can run `mkdosfs`; with a Windows host we can double-click on the drive's icon in the "My Computer" window.

Finally, we need to install a file system as follows.

```
# modprobe g_file_storage file=/root/data/backing_file
```

This command tells FSG to provide a single LUN with backing storage maintained in `/root/data/backing_file`. If we wanted to have two LUNs, where the second LUN used `/dev/hda7` as its backing storage, we would do:

```
# modprobe g_file_storage \
file=/root/data/backing_file,/dev/hda7
```

C. Configuring USB over IP Connection

As a USB connection over network, we use open source, 'USB/IP Project' [10].

The USB/IP Project aims to develop a general USB device sharing system over IP network. To share USB devices between computers with their full functionality, USB/IP encapsulates USB I/O requests into IP packets and transmits them between computers.

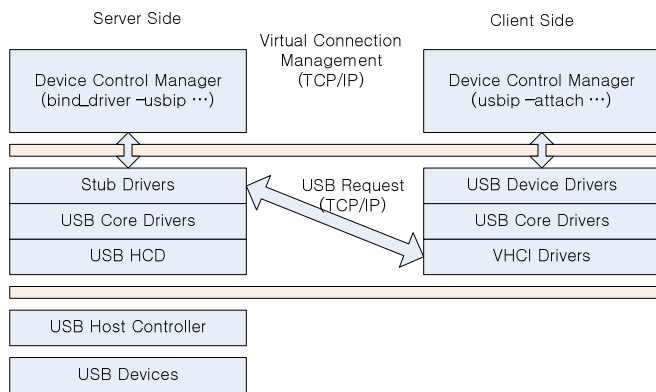


Figure 6. USB/IP software architecture

Figure 6. shows software architecture of USB/IP. And configuring procedures of a client side and a server side in Linux is briefly shown as follows.

```
-Server side (attach device)
$ insmod usbip_common_mod.ko
$ insmod usbip.ko
$ usbipd -D
$ bind_driver -usbip 1-2
<busid 1-2 is unbundled, exported>
** (process:24621): DEBUG: 1-2:1.0 -> none
** (process:24621): DEBUG: write "add 1-2" to
/sys/bus/usb/drivers/usbip/match_busid
** Message: bind 1-2 to usbip, complete!
```

```
- Client side
$ insmod usbip_common_mod.ko
$ insmod vhci_hcd.ko
$ usbip -attach <server ip addr> 1-2
$ usbip -port or lsusb
< busid 1-2 is displayed>
Port 00: <Port in Use> at Full Speed(12Mbps)
Prolific Technology, Inc. : unknown product (04bb:0206)
1-2 -> usbip://10.0.0.3:3240/1-1 (remote bus/dev 001/004)
1-2:1.0 used by usb-storage
/sys/class/scsi_device/0:0:0:0/device
/sys/class/scsi_host/host0/device
/sys/block/sda/device
$ usbip -detach 0
```

D. Connecting USB Device to a Virtual Machine

After USB/IP connection, a host system embedding Xen hypervisor recognizes a remote file system as a USB mass storage by identifying process.

Before a USB device can be added to a Xen HVM domainU guest, it must first be connected to the host system, powered on and identified. USB devices on Linux systems are identified using *vendor* and *product* ID values in the form *vendor:product*.

To add the USB device to the Xen domainU guest to address a long term requirement the device should be specified in the domainU configuration file. The first step is

to enable USB support which is achieved by adding 'usb=1' to the configuration.

With USB support enabled, the next step is to specify the device to be added. Devices are referenced using *host:vendor:product* where *vendor* and *product* are the ID values which can be found by the *lsusb* command.

Once these changes have been made, restart the domainU system. The USB device should now be accessible within the guest operating system.

If access to a USB device is needed temporarily the device can be added to the running domainU guest without shutting it down and modifying the configuration file. This feature is achieved using the QEMU monitor. Before the QEMU monitor of a running Xen HVM domainU guest can be accessed, however, a number of configuration changes are necessary. Firstly, the monitor needs to be enabled in the domainU configuration file. Secondly, the graphical console of the domainU guest must be using SDL rather than VNC. Therefore, the configuration file of domainU has following items:

```
usb=1
monitor=1
sdl=1
vnc=0
```

Once these changes are made, restart the domainU and from now on USB devices may be added and removed dynamically without having to restart the system.

At the (HVMXen) prompt a USB device may be added using the *usb_add* command together with the *host:vendor:product* value pair of the required device. The device may be subsequently removed from the domainU guest when no longer required using the *usb_del* command combined with the appropriate device identification values, for example:

```
usb_add host:03f0:a004
usb_del host:03f0:a004
```

IV. RESULTS AND DISCUSSION

This section describes results of a prototype implementation for a remote storage service.

Figure 7. shows test environment using Wi-Fi enabled Android G1 phone as a peripheral terminal and a notebook computer as a host computing device.



Figure 7. Test environment

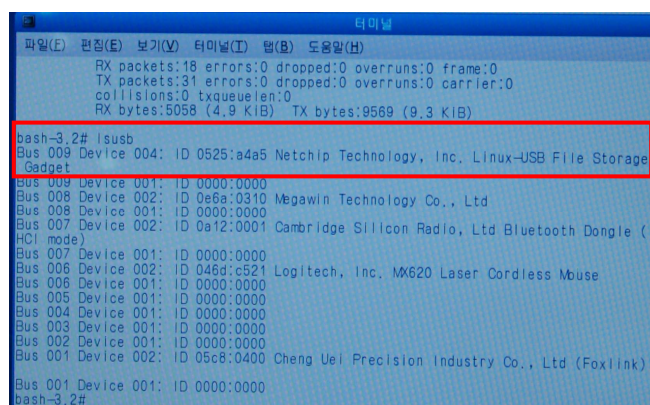
Figure 8. describes the view of configuration program running on Android G1 phone.



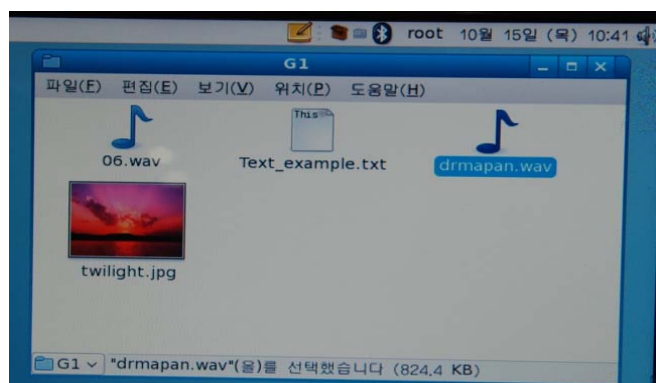
Figure 8. Configuration application in G1 phone

After enabling FSG and USB/IP, the result of 'lsmod' linux command shows FSG related g_file_storage module and two USB/IP related modules.

Figure 9. presents the result of enabled remote storage from 'lsusb' command in a host computing device.



(a) The result of 'lsusb'



(b) The result view of attached G1 phone storage

Figure 9. Result view in host computing device

The emulated USB storage of G1 phone has a plain text file, wave file, and image file. These files are accessed and played in a host computing device through USB/IP connection as if those files saved in the host computing device.

Because of the bandwidth limitation of 54Mbps Wi-Fi in a test environment, a large volume of storage is not adequate through USB protocol. However, we can expect that this limitation can be solved by means of UWB (Ultra-Wide Band) connection.

V. CONCLUSIONS

This paper addresses one of technical approaches, especially a remote storage service, for implementing the SoD system.

In this paper, we present the initial basic mechanism of a remote storage service and its implementation details. We describe how to translate a general file system into USB storage, how to connect USB storage through IP network between a host computing device and a peripheral terminal, and how to utilize a connected USB storage device in a virtual machine.

The purpose of this paper enables personal terminals such as PDA or Android phone to be used as a general USB storage device whenever a network connection is established.

We need to allow for following issues as the future research works to evolve into a full-fledged system. Firstly, we allow for various operating systems used for terminals because the prototype is implemented on Linux including Android. Secondly, we also need to deal with performance issues while a storage connection through IP network.

ACKNOWLEDGMENT

This work was supported by the IT R&D program of MKE/KEIT, [2008-S-034-01(09MC2610), Development of Collaborative Virtual Machine Technology for SoD (System on Demand) Service].

REFERENCES

- [1] Sun Microsystems, Sun xVM, <http://www.sun.com/software/-products/xvm/index.jsp>
- [2] IBM, z/VM, <http://www.vm.ibm.com>
- [3] VMware, VMware ESX Server, <http://www.vmware.com/products/vi/esx/>
- [4] Microsoft, Hyper-V, <http://www.microsoft.com/servers/hyper-v-server/default.msp>
- [5] KVM (Kernel-based Virtual Machine), <http://kvm.qumranet.com/kvmwiki>
- [6] XenSource, Xen, <http://www.xen.org>
- [7] USB Implementers Forum, Inc., USB 2.0 specification, <http://www.usb.org/developers/docs/>
- [8] Takahiro Hirofuchi, Eiji Kawai, kazutoshi Fujikawa, and Hideki Sunahara, "USB/IP: A Transparent Device Sharing Technology over IP Network," IPSJ Transactions on Advanced Computing Systems, Vol. 46 No. SIG12 (ACS11), pp.349-361
- [9] Backing Storage for the File-backed Storage Gadget, http://www.linux-usb.org/gadget/file_storage.html
- [10] Source Forge, "The USB/IP Project," <http://sourceforge.net/projects/usbip/>