논문 2011-06-32

Native Client 네트워크 기능 확장을 통한 웹기반 I/O 서비스 지원

(Supporting Web-Based I/O Service by Extending Network Communication to Native Client)

성 백 재, 박 세 진, 박 찬 익*
(Baegjae Sung, Sejin Park, Chanik Park)

Abstract : A user desktop service can be made available on internet or local area network with the help of virtualization and cloud technologies. The service is usually called a virtual desktop or a desktop cloud. However, a user interface is limited to I/O capabilities of a user's mobile terminal. In order to enhance a user interface on a remote virtual desktop, it is important to connect full-featured I/O devices which are founded locally. Our previous work called SoD (System-on- Demand) has proposed a technique to associate local full-featured I/O devices with a remote virtual desktop in Xen. On the technique, it is required to install a SoD client agent in a user's mobile terminal for connecting a remote virtual desktop. In this paper, we propose a new framework called Web-SoD that does not require any explicit installation to make SoD service available. The SoD client agent is provided by the web technology so that the agent can be installed transparently, and the platform independency is also achieved. Due to insufficient network socket performance of current web technologies, we extend Native Client (NaCl) proposed by Google to support a network functionality by modifying a NaCl library and a service runtime. With conducted experiment, we show that the network extension supports a full socket functionality over the compromised overhead on the web environment.

Keywords : Virtual desktop, Dynamic device configuration, Web service, I/O devices, Native client

I. 소 개

최근 들어, 가상화는 서버 플랫폼에서 핵심적인 기술로 자리매김하였다. 최근 들어 많은 기업들은 이러한 가상화 기술을 서버 부문에 적용하여, 클라우드 서비스를 제공하기 시작했다 [1-2]. 이러한 기술의 발전과 함께, 인터넷이나 근거리통신망을 통해 사용자의 데스크탑을 서비스하는 것이 가능해졌다. 이러한 서비스를 일반적으로 가상 데스크탑 또는 데스크탑 클라우드라고 지칭한다. 따라서 사용자는 어떤 장소에 있더라도 모바일 터미널 (노트북, 스마트폰, 태블릿 PC 등) 을 사용하여, 가상 데스크탑에 접속하여 사용할 수 있게 되었다. 하지만 현재 기술에서 사용자 인터페이스는 사용자의 모바일 터미널의 I/O 능력만 사용하도록 제한되어 있다. 즉, 사용자는 현재 사용하고 있는 모바일 터미널의 자원만 사용가능 하다. 원격 가상 데스크탑에서 사용자의 인터페이스를 향상시키기 위해서, 사용자 주변의 I/O 장치들을 활용하는 것이 필요하다. 최근 I/O 장치들 (키보드, 비디오 디스플레이, 마우스 등) 은

* 교신저자(Corresponding Author)
논문접수 : 2011. 01. 31., 수정일 : 2011. 03. 05.,
채택확정 : 2011. 04. 18.
성백재, 박세진, 박찬익 : 포항공과대학교
※ 본 연구는 지식경제부 및 정보통신산업진흥원의 대학 IT연구센터 지원사업 및 IT R&D 프로그램의 연구결과로 수행되었음.
(NIPA-2011-C1090-1131-0009)
(2008-S034-01, Development of Collaborative Virtual Machine Technology for SoD (System on-Demand) Service)

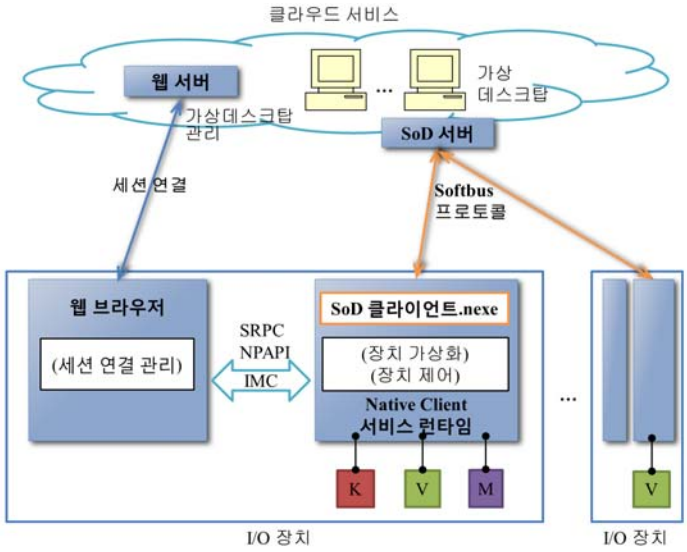

그림 1. Web-SoD 시스템 디자인: 두 대의 사용자 주변 레거시 I/O 장치를 네트워크화 된 I/O 장치로 구성

Fig. 1. Web-SoD system design: transforming two local legacy I/O devices into network-wide I/O devices

네트워크 모듈을 장착하여 출시되고 있으며, 원격의 가상 데스크탑에 직접 연결할 수 있게 되었다. 이전 연구인 SoD (System-on-Demand) [3-4]에서 Xen [6]환경의 원격 가상 데스크탑과 사용자 주변의 I/O 장치들을 연결하는 방법을 제안하였다. 하지만 사용자의 모바일 터미널에 SoD 클라이언트 응용프로그램을 설치해야 하는 번거로움이 있었다.

본 논문에서는 첫 번째로 새로운 프레임워크인 Web-SoD를 제안한다. Web-SoD에서는 SoD 서비스를 웹기술을 기반으로 제공하기 때문에, SoD 클라이언트의 명시적인 설치과정이 생략된다. 또한 플랫폼 독립적으로 수행가능하다는 장점을 가지게 된다. 두 번째로 SoD 클라이언트 구현을 위해, 현재 웹기술은 네트워크 소켓 성능의 한계가 있으므로, Native Client (NaCl) 기술 [7] 상 네트워크 기능을 추가하여 사용하였다. Google에서 제안한 Native Client 기술은 보안의 손실 없이, 웹브라우저 상에서 네이티브 코드를 수행하는 것을 제공한다. 현재 Native Client는 보안 정책 상의 이유로 네트워크 서비스를 지원하지 않으므로, SoD 클라이언트 상 요구되는 네트워크 기능 지원을 위해 NaCl 라이브러리, 서비스 런타임을 수정하였다.

본 논문은 이전 포럼논문 [5] 의 확장버전으로, 아이디어 정립 단계에서 확장되어 본 시스템을 직접 구현하고 실험을 통한 성능 검증을 진행하였다. 특히 네트워크 기능을 확장한 Native Client 기술은 범용적으로 사용되어 웹기반 I/O 서비스를 지원할 수 있으므로, 본 기술에 대해 초점을 맞추어 구체적으로 기술하고 있다.

2장에서는 Web-SoD 기술 및 Native Client 네트워크 기능 확장 기술을 설명하고, 3장에서 Native Client 네트워크 기능 확장을 위한 구현 내용을 설명하며 실험을 통해 적은 오버헤드로 웹환경에서 네트워크 소켓 기능을 지원함을 보인다. 이후 4장에서 본 Native Client 네트워크 기능을 통한 Web-SoD 클라이언트 구현에 대해 소개한다.

II. 본 론

1. Web-SoD

그림 1은 본 논문에서 제안하는 Web-SoD가 어떻게 작동하는지 개괄적으로 보여준다. 이는 어떻게 사용자 주변의 레거시 I/O 장치가 네트워크화 된 I/O 장치로 변형되는지 나타내며, 이후 네트워크화 된 I/O 장치를 가상 데스크탑에 동적으로 연결 구성하는 것을 보여준다. 먼저 하단 왼편의 I/O 장치는 키보드, 비디오 디스플레이, 마우스를 가지고 있

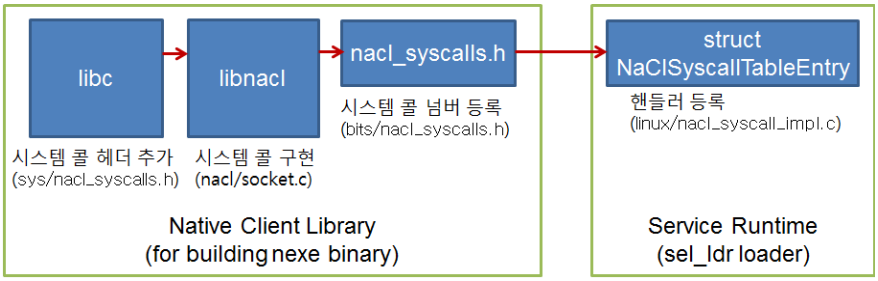

그림 2. Native Client 시스템 콜 추가 방법
Fig. 2. The method of adding native client systemcall

다. K, V, M 그림은 이를 표시한다. 이 I/O 장치는 모바일 터미널로 스마트폰이나 노트북이 될 수 있을 것이다. 본 시스템 상에서 웹 브라우저는 세션 연결을 관리하는 역할을 하고 있다. 사용자는 웹브라우저를 통해 로그인을 하며, 이러한 과정이 끝난 뒤 세션이 연결되고 유지된다. 이후 클라우드 서비스의 웹 서버는 로그인한 사용자의 가상 데스크탑을 할당해준다 - 데스크탑을 부팅하여 SoD 서버를 대기 상태로 만들어주는 것을 의미한다. 그리고 웹 브라우저는 웹 서버로부터 SoD 클라이언트 바이너리 코드(SoD 클라이언트.nexe)를 받아오게 된다. 이 바이너리 코드, 즉 SoD 클라이언트는 Native Client 서비스 런타임에 의해 실행되며, 이 Native Client 서비스 런타임은 웹브라우저 플러그인 형태로 NPAPI 같은 API를 사용하여 웹브라우저와 교신한다. SoD 클라이언트는 키보드, 비디오 디스플레이, 마우스 등의 장치를 가상화하고 제어하는 역할을 수행하며, 최적화된 I/O 전송을 지원하는 Softbus 프로토콜을 통해 SoD 서버와 교신한다. SoD 서버는 Xen 하이퍼바이저 상에 존재하며, 연결된 I/O 장치를 가상 데스크탑에 서 사용하도록 지원해 준다. 즉 사용자 주변의 레거시 I/O 장치는 네트워크화 된 I/O 장치로 변형되어, 원격의 가상 데스크탑에서 사용할 수 있다. 동일한 과정을 거쳐 비디오 디스플레이를 가진 하단 오른편의 I/O 장치도 동일한 가상 데스크탑에 연결 가능하다. 즉, 여러 I/O 장치를 가상 데스크탑에 동적으로 구성 가능하며, 이는 기존 가상 데스크탑 연결 방식 (VNC [8] 등) 과 달리 사용자 인터페이스를 크게 향상시킨다.

2. Native Client 기술 상 네트워크 기능 확장

Web-SoD를 구현하기 위하여 SoD 클라이언트 응용프로그램의 명시적인 설치과정을 제거하는 기술이 요구된다. 이러한 목적을 위해, 이전에는 RIA (Rich Internet Application)플랫폼 (Flash, Silverlight [9, 10] 등) 을 활용하였다. 이러한 환경을 기반으로 SoD 클라이언트 프로토타입을 제작하였으나, 이 환경 상 소켓 성능이 떨어져 시스템 전체의 성능을 저하시키는 문제가 발생하였다. 사용자의 데스크탑 서비스를 제공하기 위해서 이러한 환경은 적합하지 않다고 판단하였고, 이후 SoD 클라이언트를 구성하는데 Native Client 기술을 선택하였다. 이 기술은 보안의 손실 없이, 웹브라우저 상에서 네이티브 코드를 수행하며, 이 특징은 본 시스템에 적합하다고 판단되었다. 추가적으로 Native Client 기술은 운영체제 비의존성을 제공한다. 그림 1에서 Native Client 서비스 런타임 상에서 바이너리 코드 (SoD 클라이언트.nexe)가 실행되는 것을 개괄적으로 표현하고 있다.

현재 Native Client는 보안 정책 상의 이유로 네트워크 서비스를 지원하지 않고 있다. 하지만 SoD 클라이언트는 네트워크 기능이 요구되므로, 우리는 기존 RIA 플랫폼에서 지원하는 보안 정책 기법을 활용하여 Native Client 상 네트워크 기능을 지원하도록 수정하였다. 기존 RIA 플랫폼의 경우, 정책 파일을 활용하여 로컬 파일시스템과 네트워크 포트를 제한적으로 제공하고 있다. 이 정책 파일은 접근가능한 주소(Same Origin Policy)와 포트를 정의하고 있으며, 웹 응용이 소켓 등의 기능을 사용할 때, 이 정책파일을 활용하여 실행가능 여부를 결정한다. 이러한 방식은 동일하게 Native Client 기술에 적용가능하며 NaCl 라이브러리, 서비스 런타임을 수정하여 이러한 정책에 따라 네트워크 기능을 사용할 수 있도록 수정하였다.

III. Native Client 네트워크 기능 확장

표 1. Native Client 추가 시스템콜
Table 1. Added native client systemcall

```
accept(int, struct sockaddr *, socklen_t *);
bind(int, const struct sockaddr *,
        socklen_t);
connect(int, const struct sockaddr *,
        socklen_t);
getpeername(int, struct sockaddr *,
        socklen_t *);
getsockname(int, struct sockaddr *,
        socklen_t *);
getsockopt(int, int, int, void *, socklen_t *);
listen(int, int);
recv(int, void *, size_t, int);
recvfrom(int, void *, size_t, int, struct
        sockaddr *, socklen_t *);
recvmsg(int, struct msghdr *, int);
send(int, const void *, size_t, int);
sendto(int, const void *, size_t, int, const
        struct sockaddr *, socklen_t);
sendmsg(int, const struct msghdr *, int);
sendfile(int, int, off_t, size_t, struct sf_hdtr
        *, off_t *, int);
setsockopt(int, int, int, const void *,
        socklen_t);
shutdown(int, int);
socket(int, int, int);
socketpair(int, int, int, int *);
```

1. 구현

2장에서 설명한 바와 같이 본 연구에서 Web-SoD를 구현하기 위하여, Native Client 실행 환경 상 네트워크를 지원하게 수정하였다. Native Client는 네이티브 코드의 안전한 수행을 위해 모든 시스템콜을 검증한다. 즉 허용하지 않는 시스템콜을 사용하는 바이너리의 실행은 안전하지 않다고 판단하여 실행을 중단한다. 이러한 환경을 2가지 영역에서 지원하고 있다. 첫 번째는 빌드환경 (nacl-gcc 등)으로 허용하는 시스템콜을 사용하는 소스만 컴파일 되어 실행파일(nexe 바이너리)을 생성한다. 두 번째는 Service Runtime으로 바이너리를 로드하고 실행 중 허용하지 않는 시스템콜을 사용하는 경우 실행을 중단한다. - 자세한 사항은 [7] 참조

이러한 Native Client 환경에서 네트워크 기능을 확장하기 위해서는 네트워크 관련 시스템콜을 모두 추가해 주어야 한다. 이는 빌드환경의 라이브러리와 서비스 런타임의 소스 수정을 통해 이루어지며 그림 2는 Native Client에서 시스템콜을 추가하기 위해서 소스 수정이 필요한 경로를 나타내고 있다. 이는 시스템콜 헤더 추가, 시스템 콜 구현 (네이티브 운영체제의 시스템 콜 연계), 시스템 콜 넘버 등록,

표 2. Native Client 추가 코드 라인 수
Table 2. Added line of code of native client

	라인 수 (LOC)	내용
라이브러리	528	소켓 관련 시스템콜 추가
서비스 런타임	395	소켓 관련 시스템콜 핸들러 등록

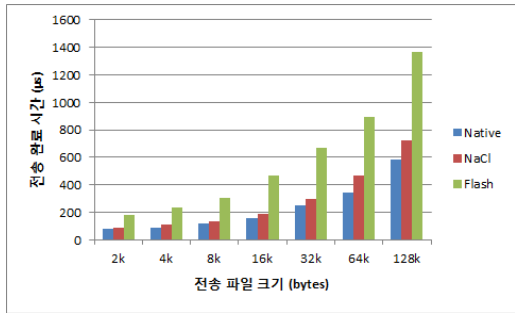

그림 3. 파일 송수신 실험
Fig. 3. Experiment: file up/download

시스템콜 핸들러 등록의 과정으로 이루어진다. 표 1은 추가된 시스템콜을 보여주며, Libc에서 제공하는 POSIX 시스템콜 형식이다. 또한 표 2은 추가된 소스의 라인 수를 나타내고 있다. 전체적으로 1k 라인 정도를 추가하였으며, 전체적인 변경 사항은 전체적인 Native Client의 보안 정책을 준수하는 범위에서 이루어졌다.

2. 성능평가

구현된 Native Client의 네트워크 성능 평가를 위하여 간단한 마이크로벤치마크를 수행하였다. 서버와 클라이언트에서는 TCP프로토콜을 사용하여 특정 크기의 바이너리파일을 주고 받으며 총 걸린 시간을 측정한다. 해당 마이크로벤치는 Native Client 환경에 구현된 socket(), send(), recv() 와 같은 소켓 관련 시스템콜의 오버헤드를 측정하는데 중점을 둔다. Flash의 경우 millisecond가 시간 측정 최소 단위임으로, 시간 측정은 서버에서 microsecond단위로 측정되었다. 실험환경은 Intel Core i7 950 (3.06 GHz), 8GB 메모리, 1TB 하드디스크(7200 RPM) 장비에 VMware 상 2GB 메모리를 할당 한 가상머신에서 테스트하였으며, 네트워크 오버헤드를 없애기 위해 동일한 운영체제에 서버와 클라이언트를 설치하였다. 운영체제는 Ubuntu

그림 4. Web-based SoD Client
Fig. 4. 웹기반 SoD 클라이언트

9.10 (커널 2.6.31) 운영체제를 사용하였다.

그림 3는 실험 결과를 보여준다. Native는 리눅스 소켓을 사용한 그래프이며, NaCl은 Native Client 상에 구현된 소켓을 사용한 그래프, 또한 Flash는 Flash 소켓을 사용한 그래프이다. 바이너리파일의 최소 전송 단위는 2KB 단위이며 각 실험값은 천 번씩 수행하여 평균을 구한 값이다. 그래프 첫 번째의 2KB 크기의 파일을 보내고 받는 경우, send(), recv() 시스템 콜을 각 한 번씩 사용하며, Native는 79 μs, NaCl은 91 μs, Flash는 177 μs의 성능을 보인다. NaCl은 Native와 비교하여 15% 정도의 오버헤드를 가지며, 이 오버헤드는 각 시스템콜을 호출할 때 마다 발생하는 NaCl 컨텍스트 스위칭, NaCl 시스템콜 처리루틴 등이 원인이다. 또한 Flash와 비교해서 약 2배 정도 전송속도가 빠른 것을 보인다. 128KB 크기 파일의 경우 Native는 587 μs, NaCl은 725 μs, Flash는 1364 μs로, NaCl은 Native에 대비하여 23%정도의 오버헤드를 가지며, Flash에 대비하여 약 2배 정도 빠른 전송속도를 보여준다.

IV. 웹기반 SoD 클라이언트 구현

1. 구현

기존 SoD 클라이언트는 C언어로 작성되었다. Native Client 환경의 장점은 이러한 네이티브 코드를 그대로 활용하여 웹에서 사용 가능하다는 것이다. 하지만 SoD 클라이언트의 경우 비디오 출력, 키보드 입력, 마우스 입력하는 코드를 사용하므로, 이러한 부분은 Native Client에서 제공하는 API를 사용하도록 변경되었다. nacl_video_init(), nacl_video_update(), nacl_video_poll_event() 와 같은 API들을 사용하여 구현되었다. 소켓과 관련된 API들은 Native Client 상 네트워크 기능을 추가하였으므로, 그대로 사용하였다.

2. 구동 테스트

Web-SoD 시스템을 위하여, 웹기반 SoD 클라이언트를 구현하였으며, 그림 4는 이를 보여주고 있다. Ubuntu 9.10 (커널 2.6.31) 운영체제 상에 파이어폭스 웹브라우저에 네트워크 기능을 확장한 Native Client 플러그인을 설치하였다. 그리고 구현한 SoDClient.nexe(바이너리 파일)을 이 환경 상에서 실행하였다. 본 SoD 클라이언트는 원격지의 가상 데스크탑에 연결되어, 비디오, 키보드, 마우스 장치를 연결시킨다. 현재 원격 가상 데스크탑에는 윈도우XP가 설치되어 있으며, 일반적인 원격 데스크탑과 유사하게 사용 가능하다. 원격 데스크탑 기술의 성능은 화면의 FPS(Frames per Second)에 따라 차이가 나타난다. 즉 30 FPS은 화면의 갱신이 빨라 부드러운 화면을 제공하지만 20 FPS 이하의 경우 끊기는 현상이 나타난다. 하지만 원격 데스크탑 상 FPS를 측정하는 것은 쉽지 않으므로, 그림 3의 실험결과를 토대로 데스크탑의 성능을 가늠해 볼 수 있다. 즉 본 SoD 클라이언트는 웹기반으로 작동하나 네이티브 프로그램들과 유사한 수준의 성능이 나올 것으로 기대하며, 기존 Flash와 같은 RIA 기술로 작성된 웹기반의 원격 데스크탑 프로그램보다 약 2배 정도의 우수한 성능을 보여 줄 것으로 기대된다.

V. 결 론

본 논문에서 새로운 프레임워크인 Web-SoD 를 제안하였으며, 이는 기존 시스템에 비해 두 가지의 장점을 제공한다. 첫 째는 사용자 주변의 레거시 I/O 장치를 네트워크화 된 I/O 장치로 변형하는 것이다. 이 네트워크환 된 I/O 장치는 SoD 클라이언트에 의해 관리가 이루어지며, 동적으로 원격의 가상 데스크탑에서 사용 가능하도록 구성가능하다. 이

를 통해 사용자 인터페이스를 크게 향상 시킬 수 있었다. 두 번째는 클라이언트 응용프로그램의 명시적인 설치과정을 제거하는 것이다. 이후 Web-SoD를 구현하기 위하여, Native Client 실행 환경 상 네트워크 기능을 추가하였다. 그리고 기존 보안 정책을 준수하도록 정책파일을 활용하여 구현하였다. 이후 성능을 측정하였으며 네이티브 리눅스에 대비하여 20%정도의 오버헤드가 발생하였으나, 기존 RIA 기술인 Flash와 비교하여 약 2배 정도 빠른 전송 능력을 보여주었다. 마지막으로 웹기반 SoD 클라이언트를 구현하여 Web-SoD 환경을 검증하였다.

참고문헌

[1] Xen Desktop, http://www.citrix.com/virtualization/desktop

[2] VMware View, http://www.vmware.com/products/view/overview.html

[3] S. J. Park, S. W. Kang, C. I. Park, "Design and implementation of a framework to convert legacy device to network-wide device", Korea Computer Congress 2009.

[4] S. W. Kang, S. J. Park, C. I. Park, "Design and implementation of a device virtualization framework to control virtual desktop", Journal of KIISE : Computing Practices and Letters, Vol.16. No.6, pp. 631-751, 2010.

[5] B. J. Sung, S. J. Park, C. I. Park, "Web-SoD: supporting web-based I/O services in xen environment by native client with network extension", Ph.D Forum, Int. Sym. on Wearable Computers 2010.

[6] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization", In proceedings of the 19th ACM symposium on Operating Systems Principles, pp.164-177, Bolton Landing, NY, 2003.

[7] B. Yee et eal., "Native client: a sandbox for portable, untrusted x86 native code", Security and Privacy, 2009 30th IEEE Symposium, pp.79-93, May. 2009.

[8] Tight VNC, http://www.tightvnc.com.

[9] Adobe RIA , http://www.adobe.com/kr/resources/business/rich_internet_apps

[10] Microsoft Silverlight, http://www.silverlight.net

저 자 소 개

성 백 재 (Baegjae Sung)

2006년 : 인하대학교 컴퓨터공학 학사.
2009년 : 포항공과대학교 정보통신공학 석사.
현재, 포항공과대학교 컴퓨터공학 박사과정.
관심분야 : Embedded systems, File systems, Virtualization.
Email : jays@postech.ac.kr

박 세 진 (Sejin Park)

2007년 : 금오공과대학교 컴퓨터공학 학사.
현재, 포항공과대학교 컴퓨터공학 통합과정.
관심분야 : Virtualization, Operating system.
Email : baksejin@postech.ac.kr

박 찬 익 (Chanik Park)

1983년 : 서울대학교 전기공학 학사.
1985년 : KAIST 전자전기공학 석사.
1988년 : KAIST 전자전기공학 박사.
1991~1992년 : IBM Thomas J. Watson Research Center visiting scholar.
1999~2000년 : IBM Almaden Research Center visiting professor.
1989년~현재, 포항공과대학교 컴퓨터공학과 교수.
관심분야 : Storage systems, Embedded systems, Pervasive computing.
Email : cipark@postech.ac.kr